Design and Implementation of an FPGA-based Soft-Radio Receiver Utilizing Adaptive Tracking

John C. Davies IV

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

Master of Science
in
Electrical Engineering

Peter M. Athanas, Chair
Timothy Pratt
Jeffrey H. Reed

August 25, 2000
Blacksburg, Virginia

Keywords: FPGA, Configurable Computing, CDMA,
Software Radio, Adaptive Receiver, Adaptive Tracking

Design and Implementation of an FPGA-based Soft-Radio Receiver Utilizing Adaptive Tracking

John C. Davies IV

(ABSTRACT)

The wireless market of the future will demand inexpensive hardware, expandability, interoperability, and the implementation of advanced signal processing functions—i.e. a software radio. Configurable computing machines are often ideal software radio platforms. In particular, the Stallion reconfigurable processor's efficient hardware reuse and scalability fulfill these radio's demands. The advantages of Stallion-based design inspired an FPGA-based software radio—the proto-Stallion receiver. This thesis introduces the proto-Stallion architecture and details its implementation on the SLAAC-1V FPGA platform. Although this thesis presents a specific radio implementation, this architecture is flexible; it can support a variety of applications within its fixed framework. This implemented single-user DS-CDMA receiver utilizes an LMS adaptive filter that can combat MAI and constructively combine multipath; most notably, this receiver employs an adaptive tracking algorithm that harnesses the LMS algorithm to maintain symbol synchronization. The proto-Stallion receiver demonstrates the dependence of adaptive tracking on channel noise; the algorithm requires significant noise levels to maintain synchronization.

Acknowledgements

First, I would like to thank my graduate committee: my advisor Professor Peter Athanas, Professor Timothy Pratt, and Professor Jeffrey Reed. Second, I would like the many others who have aided this research. I am particularly grateful to Prinya Atiniramit; I could not have reached the ends of this thesis without standing on his shoulders. Also, I am indebted to Luke Scharf, Professor Mark Jones, Scott Harper, and the other members of the Configurable Computing Lab. At the MPRG, I owe thanks to Kathyayani Srikanteswara, Jody Neel, and Michael Hosemann. Third, I dedicate this thesis to my wife Tinelle.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Configurable computing machines (CCMs) bridge the gap between application specific integrated circuits (ASICs) and general-purpose microprocessors. They retain the flexibility of microprocessors while providing speed and power consumption more comparable to ASICs. CCMs represent a powerful alternative for certain applications, particularly communication systems.

The wireless market of the future will demand inexpensive hardware, expandability, interoperability, and the implementation of advanced signal processing functions—i.e. a software radio. The natural versatility of CCMs makes them the ideal baseband processor for software radio. Digital signal processors (DSPs) often lack the necessary speed to implement these algorithms, and additionally have higher power consumption—a significant drawback for mobile applications.

Field programmable gate arrays (FPGAs) are the flexible computational resource in most mainstream CCMs. FPGAs consist of simple computational units called combinational logic blocks (CLBs) linked by a configurable connective mesh. While FPGAs are extremely versatile, they have a significant drawback as a software radio platform: long reconfiguration times. To reduce the hardware needs for a given application—consequently reducing cost and power consumption—it is desirable that platforms support runtime reconfiguration. With configuration times on the order of milliseconds, such reconfiguration is impractical at typical wireless data

speeds. The Stallion CCM overcomes this problem while retaining much of the flexibility of an FPGA. The Stallion trades the CLB for a much more sophisticated 16-bit functional unit. A coarser mesh of larger elements is less flexible than the fine mesh of an FPGA, but is faster, both in terms of operation and reconfiguration. The wider data path also simplifies routing over a single-bit approach.

The key advantage of the Stallion as a soft-radio platform is its ability to switch rapidly among many configurations. This thesis presents an FPGA-based software radio receiver inspired by this capability of the Stallion device. A Stallion prototype was unavailable at the time of this research, but an FPGA-based design—particularly a Virtex-based design—can mimic many of the advantages of a Stallion-based architecture. The developed soft-radio—termed the proto-Stallion receiver—provides both flexibility and simplified control through its modular, swappable design.

1.2 Contributions of this Work

The proto-Stallion architecture for software radio is this research's primary contribution. The backbone of this architecture is a VHDL-described processing element that resembles a fixed Stallion implementation: the proto-Stallion module. Interface and control hardware surrounds these modules. While the proto-Stallion architecture is a harbinger of Stallion-based design, it is more useful for its advancement of FPGA-based soft-radio. Its proto-Stallion modules are building blocks; a library of these modules allows the implementation of countless radios configurations on a single FPGA device.

Following the FPGA-based receivers developed in [1] and [2], this research used the proto-Stallion architecture to implement an FPGA-based software radio prototype. This DBPSK DS-CDMA single-user receiver utilizes an LMS adaptive filter to combat MAI and constructively combine multipath. A key contribution of the radio's design is the incorporation of an adaptive tracking mechanism [3]. This algorithm offers significant advantages over the tracking systems implemented in previous FPGA-based receivers. In addition to more reliably following a drifting signal, this technique greatly reduces the number of computations required per symbol.

1.3 Organization of Thesis

Following this introductory chapter, Chapter 2 surveys the proto-Stallion receiver's hardware platforms, including the Stallion device and the SLAAC-1V FPGA development board. Also, this chapter compares the proto-Stallion receiver to similar soft-radios employing configurable computing. Chapter 3 provides background on the receiver's signal processing algorithms. Chapter 4 describes the receiver's architecture. This chapter includes a description of the proto-Stallion design methodology and provides details of the radio's implementation. Chapter 5 summarizes the results of this implementation, focusing on the behavior of the adaptive tracking algorithm. Chapter 6 concludes with an overview and suggestions for future work.

Chapter 2

Configurable Computing for Software Radios

2.1 Overview

Configurable computing machines are often ideal software radio platforms. In particular, the Stallion reconfigurable processor's efficient hardware reuse and scalability keenly fulfill the demands of software radios. The Stallion has not yet been fabricated, but the advantages of Stallion-based design inspired the FPGA-based software radio developed for this thesis—the proto-Stallion receiver. Although the Stallion stimulated this radio's design, the proto-Stallion receiver is a standalone device; it is viable far beyond its insights into Stallion-based architectures. This chapter first surveys the Stallion device, highlighting its advantages for signal processing applications. Next, this chapter details the proto-Stallion receiver's foundation: the SLAAC-1V FPGA platform. A discussion of CCM-based software radios concludes this chapter; this section focuses on two comparable FPGA-based soft-radios developed at Virginia Tech.

2.2 Stallion CCM

Software radio architectures demand versatile baseband processors. Not only must these processors adapt to a variety of existing algorithms and standards, they must scale to meet increased future requirements. FPGA-based systems can provide this level of flexibility.

Software radio systems benefit from an even more profound level of flexibility: run-time reconfiguration (RTR). Run-time reconfiguration allows radios to optimize processing to meet rapidly changing environmental conditions. Also, RTR permits hardware reuse; i.e. the reconfiguration of idle processing modules to more productive tasks. This reduces both component cost and power consumption. FPGA-based systems are unfortunately not adept at this level of flexibility. FPGAs have relatively long reconfiguration times and their design process inhibits run-time modifications to FPGA configurations files. FPGA implementations are typically entered using either a schematic or a hardware description language (HDL). A series of sophisticated tools translates the design into the files used to configure the devices (bit-files)—a process that can take hours. These bit-files do not intuitively resemble the initial design description, rendering partial modification extremely difficult. The JBits application programming interface (API) partially addresses this issue; it allows designers to write information directly to a Xilinx FPGA. Virtex FPGAs even allow partial reconfiguration [4]. JBits facilitates the reconfiguration of FPGAs, but RTR remains inelegant.

The limitations of FPGAs for RTR motivated the development of alternative CCMs that overcome these deficiencies. One example of such a device is the Sanders Context Switching Reconfigurable Computer (CSRC) [5]. The CSRC can store up to four configurations (contexts) and can rapidly switch among them. The device is limited, however, to these four configurations upon its programming. Another example device is the Stallion CCM. The Stallion retains the versatility of an FPGA while allowing efficient run-time reconfiguration; it is particularly attractive for soft-radio applications. The paragraphs below summarize the Stallion architecture; refer to [6] and [7] for the full details.

The key to the Stallion's efficient RTR is its stream-based architecture. A stream is a concatenation of a programming header with data; as the stream navigates the Stallion, the programming header locally configures a processing pipeline that subsequently operates on the trailing data. These streams are self-directed, eliminating the need for a global control structure. Multiple streams can partition the device to conduct parallel computations. Moreover, each computational pipeline can be reconfigured in run-time while the other pipelines continue to process data.

Figure 2.1 depicts the architecture of the Stallion. Six configurable data ports provide I/O for the device, while the 64 processing units are separated into two meshes bridged by a smart

crossbar. Each mesh contains two dedicated multipliers and 30 interconnected functional units (IFUs). The IFU in turn contains a single functional unit (FU)—the Stallion's computational building block—along with circuitry for control and connectivity. The FU can perform a variety of arithmetic and logical operations on its two 16-bit operands—plus shift, delay, and conditional operations. To program the Stallion, a stream enters at a data port, configures it, and then proceeds to configure a crossbar connection. The stream then establishes a processing pipeline through a series of IFUs before exiting the chip through the crossbar and another data port.

The two meshes support advanced connectivity that maximizes the usability of the Stallion's computational resources. The crossbar connects many of the processing elements and supports multicasting. In addition to the meshes' systolic busses, there is a skip bus that allows data transfer between non-neighboring IFUs in a single clock cycle. Such measures maximize computational density by eliminating the need to waste IFUs for routing purposes. Also, this versatile connectivity simplifies the allocation of the Stallion's computational resources. While

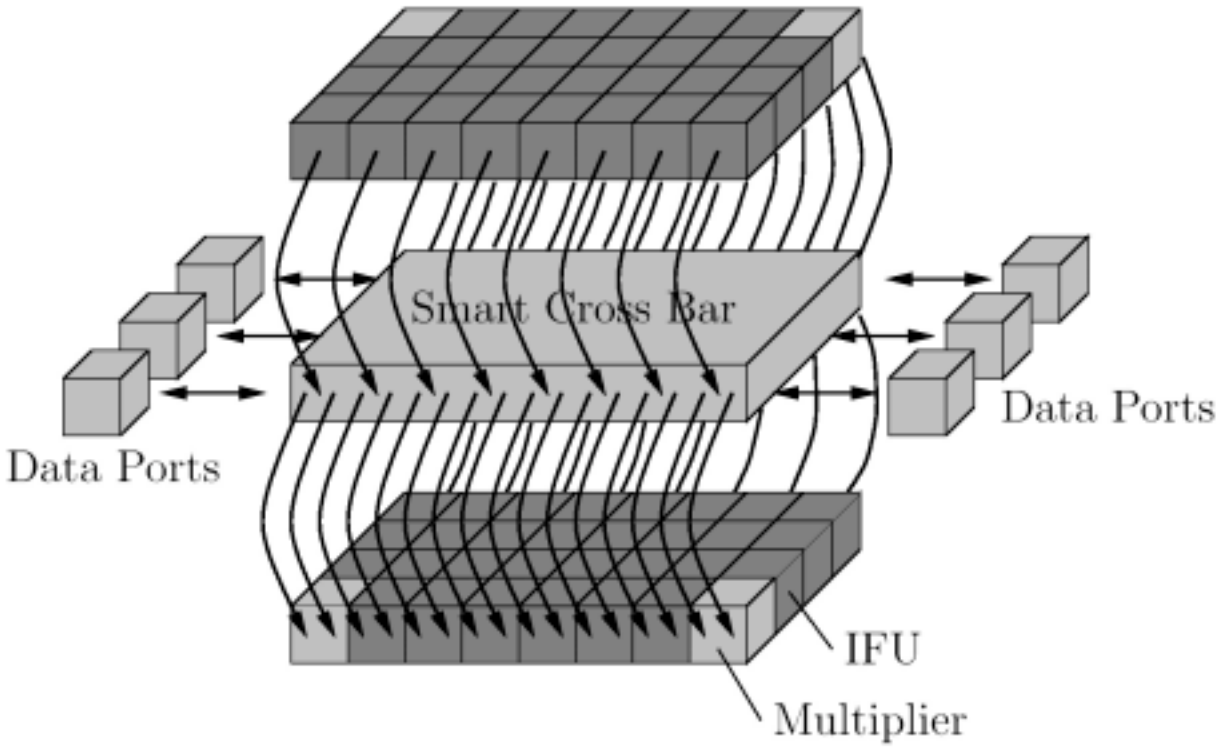

Figure 2.1. *Stallion Architecture [3].*

an FPGA design requires proximal CLBs to implement a computation, the Stallion supports processing pipelines implemented on IFUs scattered across the chip.

2.3 SLAAC-1V FPGA Platform

The Stallion offers many advantages for use as a software radio platform; however, it is not yet fabricated. Inspired by the potential advantages of a Stallion-based system, this thesis presents an FPGA-based proto-Stallion architecture. Chapter 4 details this architecture; this section describes the FPGA-based configurable computing platform on which this architecture is implemented: the SLAAC-1V. The size and flexibility of the SLAAC-1V's Virtex devices allow the implementation of a fast, powerful soft-radio that, given Virtex's support of partial reconfiguration, could eventually be run-time reconfigurable. While the Virtex's RTR will never match the speed of the Stallion's, it enables the development of a receiver that can support multiple standards on a single platform.

SLAAC-1V is an FPGA-based accelerator constructed on a full-sized 64-bit PCI board. This board features three Xilinx Virtex XCV1000 parts and ten 256Kx36 100MHz ZBT synchronous SRAMs. Figure 2.2 depicts the architecture of the SLAAC-1V. Two Virtex XCV1000s are dedicated processing elements (PEs)—X1 and X2. Each PE has a memory bank of four SRAMs. These memories are also accessible through an external memory bus. Two 72-bit ring connections connect the PE to its left and right neighbors. A three-port 72-bit shared crossbar provides access to the other FPGAs or to external I/O connectors.

Only one-half of the third Virtex XCV1000—X0—is user programmable. X0 features the same connectivity as X1 and X2: two 72-bit ring connections and a 72-bit shared cross bar connection. X0 has only two SRAMs, but bus switches allow single cycle memory exchange between X0 and X1/X2. The PE can exchange its first memory for any memory in X1's bank; similarly, it can exchange its second memory for any memory in X2's bank.

The remaining half of the Virtex XCV1000 handles the board interface. This interface chip (IF) implements 86-bit PCI-64/66, controls the clock, and monitors power consumption. The IF manages the host memory interface; its external memory bus accesses all board memories.

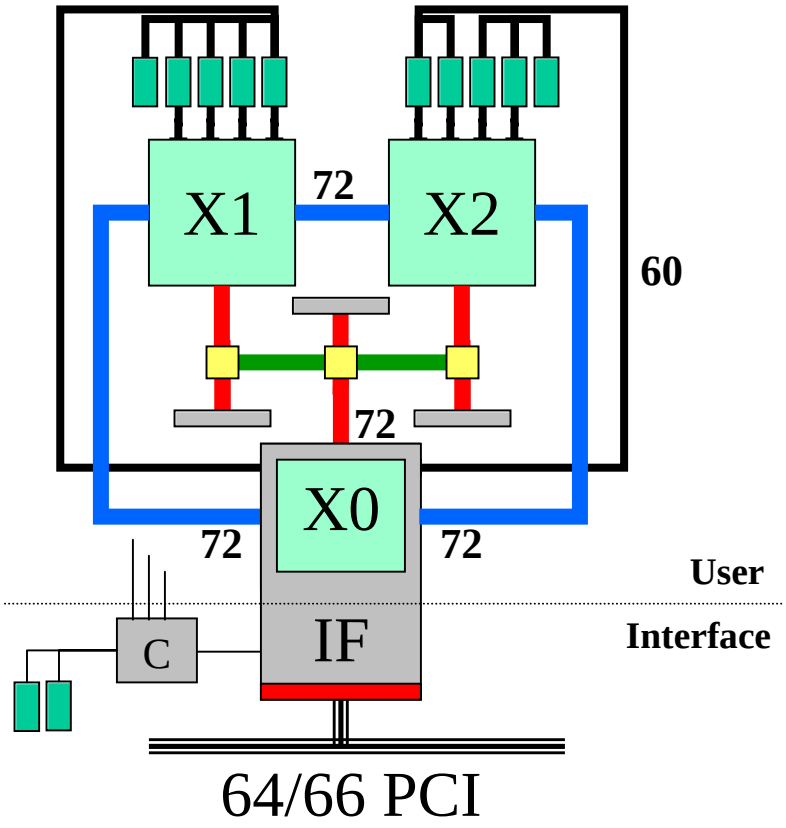

Figure 2.2. *SLAAC-1V Architecture [8].*

This device also includes a FIFO bank: X0 selects from four input and four output FIFOs—each is 68-bits wide, including a four-bit tag.

An additional Virtex XCV100 acts as a configuration controller; a pair of dedicated configuration memories (one FLASH, one SRAM) completes the configuration system. The configuration controller can perform a default boot from the FLASH memory or reboot configurations from the SRAM; this chip also controls partial runtime reconfiguration of X1 and X2. [8]

For information on the development of VHDL designs for the SLAAC-1V architecture, refer to [9]; [10] summarizes the SLAAC-1V's host interface.

2.4 CCM-based Software Radios

A number of recent research efforts have explored CCM-based software radios. [11] describes the implementation of loop-based carrier and timing synchronization on a Virtex FPGA. [12] introduces the Dynamically Reconfigurable Architecture for Mobile Systems (DREAM). This architecture's foundation is an array of reconfigurable processing units (similar to the Stallion's functional units) that provides both high performance and flexibility. Two FPGA-based soft-radios developed at Virginia Tech are particularly relevant to this research. Both of these stream-based, DS-CDMA DBPSK devices use the GigaOps G900 FPGA platform to perform the baseband processing of a software radio. Swanchara implemented a multi-user receiver employing parallel interference cancellation [1]. Atiniramit's radio is the direct predecessor to the proto-Stallion radio outlined in this thesis; he implemented a single user receiver using an LMS adaptive filter [2].

2.4.1 Multi-User Receiver Employing Parallel Interference Cancellation

In a CDMA system, multiple users transmit at the same time and frequency; other transmissions interfere when demodulating a single user. A wireless base station typically demodulates the transmissions of several users simultaneously; i.e. it incorporates a multi-user receiver. Using each transmission's initial estimate, a receiver can recursively improve the reliability of each user's demodulated symbol by removing the interference generated by the other users. This is termed interference cancellation; Swanchara structured his receiver around implementation of this technique.

Figure 2.3 depicts the architecture of Swanchara's receiver. This architecture is stream-based; programming information and data travel the same path. Each module incorporates a stream decoder that analyzes the incoming packets. Programming packets reconfigure the module, while data packets are processed according to the current configuration. This receiver processes up to four users serially using programming packets to customize the pipeline for each user. Dedicated hardware for each user would have increased the design's computational complexity and would not easily scale to a larger number of users.

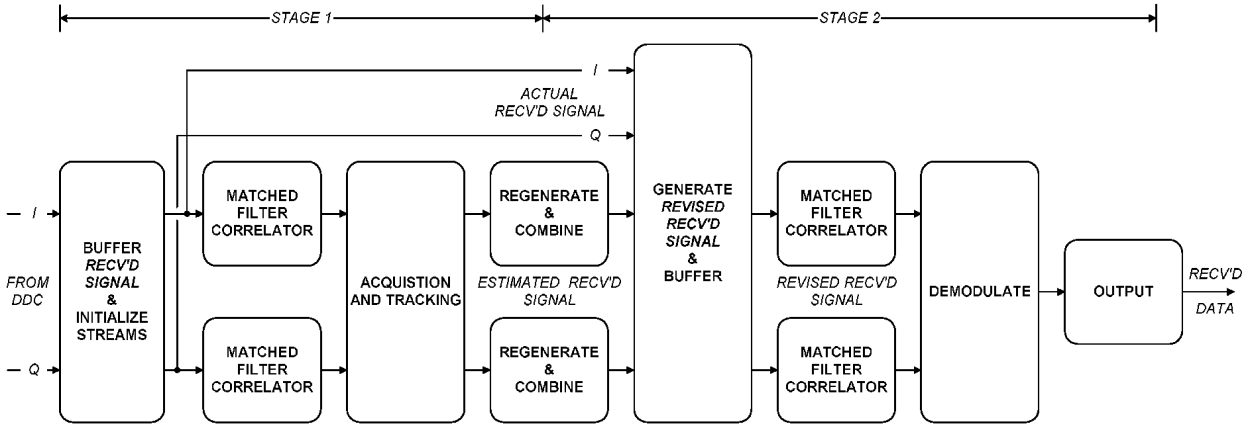

Figure 2.3. *Multi-User Receiver Architecture [1].*

Swanchara's receiver uses a traditional FIR filter structure (a tapped delay line and an adder tree) for its matched filters. Each filter multiplication reduces to an addition/subtraction since the coefficients correspond to either ±1. This simplification does not hold for the proto-Stallion receiver's adaptive filter, but its acquisition filter takes advantage of this reduction. The acquisition filter's structure is significantly different, however; it exploits oversampling to reduce both complexity and latency. Refer to Section 4.4.1 for a detailed comparison of these filter structures. Swanchara's tracking modules used an early-late loop to adjust the position of the data relative to the filter coefficients; the proto-Stallion receiver's adaptive tracking performs this adjustment automatically, eliminating timing error detection.

2.4.2 Single-User LMS Adaptive Receiver

Atiniramit implemented a single-user LMS adaptive receiver on the G900 FPGA platform. Figure 2.4 depicts the architecture of this receiver. As with Swanchara's receiver, this architecture is stream based. A bi-directional bus carrying both programming information and data connects neighboring modules. The modules operate on the incoming data as directed by the programming packets, then pass new data packets along the pipeline. The proto-Stallion receiver also implements a single-user LMS adaptive receiver; Chapter 3 details the operation of these algorithms.

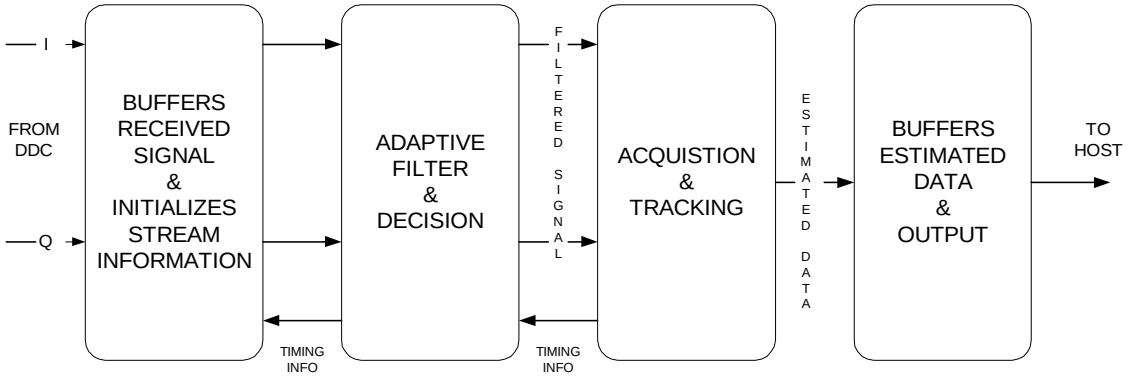

Figure 2.4. *Single-User Receiver Architecture [2].*

The proto-Stallion architecture eschews a stream-based approach. However, since it is also a single-user receiver employing an LMS adaptive filter, several of its modules resemble their counterparts in Atiniramit's receiver. To increase flexibility, the proto-Stallion receiver separates Atiniramit's Adaptive Filter and Decision module into thirds: a Correlate and Track module, an Update Coefficients module, and a Decision module. Although adaptive tracking altered the implementation (but not the underlying algorithms) of the first two modules, the Decision module is clearly the descendant of Atiniramit's. The Decision module borrows Atiniramit's integration of differential demodulation and scaled error calculation as well as his techniques for multiplier reuse. Both receivers used similar acquisition algorithms, but proto-Stallion's dedicated acquisition filter is faster and more efficient than analyzing the output of the adaptive filter. Like Swanchara's, Atiniramit's receiver used an early-late tracking loop that bears little resemblance to the proto-Stallion receiver's adaptive tracking.

While Atiniramit's radio performed well, the limitations of the G900 compromised the receiver's architecture. Atiniramit implemented the radio across several small (XC4028) FPGAs; the most computationally intense module required five FPGAs. The limited connectivity among these FPGAs forced time multiplexing of the busses, which slows processing. Additionally, although information transmission requires the calculation of only one filter output per symbol, this receiver calculated a filter output every sample for synchronization purposes. In order to meet this steep computational demand, the receiver clocked its multipliers at ten times the sample rate, ultimately reducing data throughput. By both switching to a more

powerful FPGA platform and incorporating a new synchronization technique, the proto-Stallion receiver provides significantly higher data rates than Atiniramit's radio.

Chapter 3

Algorithms

3.1 Overview

This chapter provides both the communications background and the algorithmic details necessary to understand the implemented FPGA-based receiver. This chapter includes an overview of spread spectrum systems and adaptive filters. The descriptions of the LMS adaptation algorithm and the synchronization algorithms are tailored to the proto-Stallion receiver implementation detailed in Chapter 4.

3.2 Spread Spectrum Systems

Initially developed for military applications because of their covertness and resistance to jamming, spread spectrum systems now draw considerable commercial interest. Spread spectrum multiple access—also known as code division multiple access (CDMA)—owes its popularity to its high bandwidth efficiency relative to other multiple access techniques. Other benefits of CDMA include accurate power control, good performance in fading channels through multipath combining, and improved cell boundary performance and reduction in dropped calls through soft handoff support. [13]

Unlike other multiple-access techniques that allocate a time or frequency slot to each user, CDMA spread each user's data over entire available bandwidth. All users transmit at the

same time and frequency. To distinguish among the signals, each user transmits using a pseudo-random spreading code—the transmitter modulates all information bits using this signature sequence. These codes have good auto- and cross-correlation properties, minimizing the interference from both multipath and other users' transmissions. A CDMA receiver can select the data encoded with the desired spreading sequence from among the other user's signals, which appear as additive white Gaussian noise (AWGN).

In direct sequence spread spectrum (DS-SS) systems, the user's spreading code modulates each transmitted information bit. For each bit valued "1", a DS-SS transmitter sends the spreading code bit sequence; for each bit valued "0", the transmitter outputs the negative spreading code bit sequence. Given a spreading sequence of length N, the transmitter sends N code bits, also known as chips, for each transmitted information bit. This correspondence of data and code period is termed code-on-pulse modulation.

At the receiver, the classical approach for demodulation a DS-SS signal is the matched filter. This filter has taps corresponding to the user's spreading code; when this user's data is properly aligned, the resulting filter output corresponds to the "1" or "0" transmitted. This process is termed despreading. The matched filter is the ideal receiver structure for demodulating a spread spectrum signal in an AWGN environment. Although the AWGN channel is a good model for many communications systems, it is an inadequate model for many of the environments—particularly mobile environments—in which CDMA systems are applied. Both multipath and multiple access interference (MAI) introduce non-AWGN interference to the channel. Many of the shortcomings of the matched filter receiver in a non-AWGN environment can be overcome through adaptive filter techniques.

3.3 Fractionally-Spaced Adaptive Filters

The characteristics of a communications channel are stochastic: channel behavior cannot be precisely determined prior to transmission. In the case of a DS-SS system, the transmissions of other users as well as the characteristics of a fading channel are time-varying; it is impossible to design a static filter to optimally compensate for these conditions. Adaptive filters, however, can overcome for such unpredictable sources of interference; they recursively update their coefficients to minimize output error.

Many adaptive filter structures address unpredictable sources of interference. When given a DS-SS signal and the following conditions

- Spreading code repeats every symbol
- The estimated signal is sampled once per symbol
- The symbol rate is harmonically related to the sample rate

the optimal receiver structure is an equalizer [14]. In particular, the linear fractionally-spaced adaptive equalizer offers many advantages in a DS-SS system. Figure 3.1 depicts a linear fractionally spaced receiver structure. In this structure, the received signal is

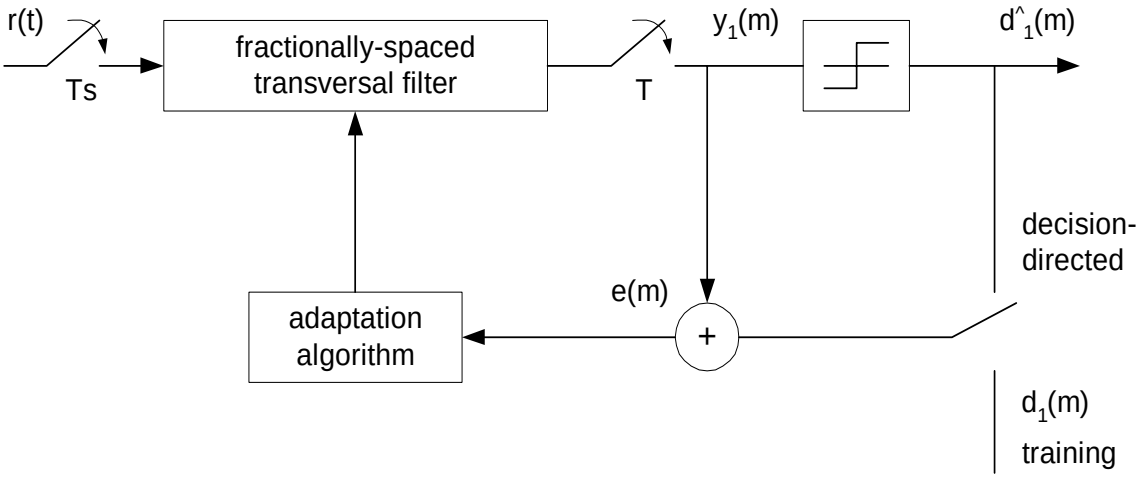

Figure 3.1. *Linear Fractionally Spaced Receiver [2].*

oversampled, i.e. it is sampled at a multiple of the chip rate; two to four samples per chip are common. The corresponding matched filter must increase in length to accommodate the complete oversampled symbol; it must have at least Np taps, where N is the length of the spreading code and p is the number of samples per chip. Receivers that exploit this oversampling are particularly beneficial in multipath channels and for asynchronous transmission [15]. Also, the decreased time between each sample makes the linear fractionally spaced receiver less vulnerable to sampling errors. When a DS-SS receiver uses an adaptive filter, the resulting filter not only depsreads the desired user's signal but can also combat MAI and constructively combine multipath.

3.4 LMS Adaptation Algorithm

A number of adaptation algorithms can operate within the linear fractionally spaced receiver. Following the successes of [2], this research utilizes the least mean squares (LMS) adaptation algorithm. This algorithm offers a reasonable convergence rate, low steady-state error, and low computational complexity relative to other adaptation algorithms

This receiver uses a modification of the LMS algorithm in order to compensate for a rotating symbol constellation. Each transmitted bit is encoded differentially, i.e. its encoded value is based on the value of the previous bit. The receiver detects the phase difference between successive symbols to recover the information bits; unlike a signal's absolute phase, phase difference changes little in the presence of a frequency offset. While this modification adds slightly to the complexity of the algorithm, the ability to compensate for a rotating symbol constellation is invaluable in a mobile environment. Figure 3.2 depicts a linear fractionally spaced receiver incorporating differential detection.

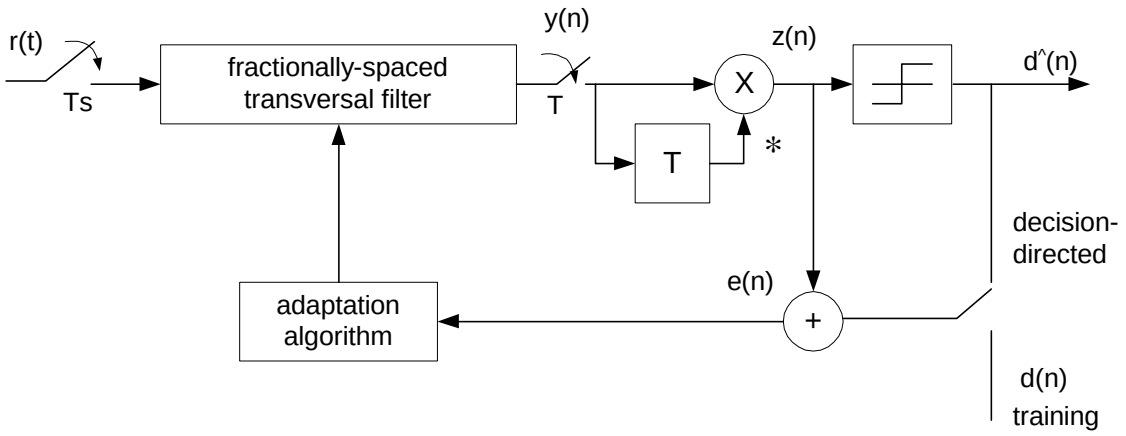

Figure 3.2. *LFSR Incorporating Differential Detection [2].*

The LMS algorithm first requires the correlation of the incoming data with the filter coefficients—initially the desired user's spreading code—or

$$y(n) = \mathbf{w}^H(n)\mathbf{r}(n) \tag{3.1}$$

The result is subsequently multiplied with the results of the previous symbol, decoding the differentially encoded information bit:

$$z(n) = y(n) \cdot y^*(n-1)$$

(3.2)

The proto-Stallion receiver operates in decision directed mode, i.e. it calculates the error based on its own determination of the ideal transmitted value. Correspondingly, the error is calculated by

$$e(n) = d(n) - z(n) \tag{3.3}$$

where *d* is the hard decision performed on *z*. This error value is used by the update equation

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \cdot e(n) \cdot y^*(n-1) \cdot \mathbf{r}(n) \tag{3.4}$$

to calculate the new filter coefficients.

3.5 Synchronization

All of the receivers described in this chapter require synchronization in order to despread a DS-SS signal accurately; i.e. the receivers need to recover the symbol timing of the transmitted signal. The task of synchronization is twofold: the receiver must detect presence of a user's signal and assesses the signal's initial code phase, plus it must update this assessment as the receiver processes this signal. User detection and initial code-phase estimation is termed acquisition, while the process of maintaining knowledge of the code phase is called tracking.

3.5.1 Acquisition

The acquisition process must both detect the presence of a user and provide an initial estimation of that user's symbol timing. Both these responsibilities can be fulfilled with a matched filter that produces a filtered output with every data sample. The data will align with the spreading code coefficients once and only once per symbol period. Assuming a spreading code with good autocorrelation properties, the magnitude of the filter output at this time will be significantly greater than the other outputs in the symbol window; this peak is the basis of matched filter acquisition.

A common acquisition technique sets a threshold on the output of a matched filter. If the filter output falls below the threshold, then the signal is purely noise or misaligned. If the output exceeds this value, the receiver assumes that the data is properly aligned with the spreading code. This scheme is simple to implement, but setting the threshold level is problematic; in a fading channel or in the presence of high noise levels, the acquisition algorithm can either overlook a user or falsely acquire when no user is present.

A maximum search for a persistent peak avoids the use of a threshold and its accompanying problems. This technique analyzes the filter outputs over each symbol period for the result with the largest magnitude. If this peak occurs consistently at the same phase for several symbol periods, a user is present; irregular maximum peak locations indicate the user's absence. While a maximum search over a single symbol window can recover the user's spreading code alignment, this result is valid only if the user is transmitting. Since the maximum search algorithm will always return a peak location irrespective of the presence of a user, a single maximum search has no means of distinguishing a user peak from a peak caused by noise. Even with a transmitting user, noise could produce an alternate peak. A noise source, however, will not produce a peak that consistently appears at the same code phase over several symbols. By considering the peak position over several symbols, the persistent peak algorithm features reliable user detection while providing an accurate estimate of the user's symbol phase.

3.5.2 Adaptive Tracking

The acquisition mechanism provides an initial estimate of the user's symbol phase, but this phase is not constant. Due to both Doppler shifts and clock mismatches, this phase will change slowly over time. Receivers incorporate tracking to compensate for this phase drift. Unlike acquisition mechanisms that only operate prior to the initial user detection, tracking algorithms initiate following user detection and continue as long as the receiver produces valid output.

Classical tracking schemes utilize a tracking loop: a tracking error mechanism detects the drifting signal in the output of the matched filter and restores the proper alignments through advancing or delaying the received signal. Unfortunately, this tracking scheme can add considerably to the computational complexity of the receiver. While only one matched filter

output per symbol is required to despread the transmitted signal, the tracking loop must monitor additional matched filter outputs with small phase offsets. At a minimum, the loop would monitor three filter outputs (advanced one sample, in phase, and delayed one sample), tripling the filter calculations per symbol. A novel approach in [3] uses the ability of an adaptive filter to track signal variations in a nonstationary environment to perform symbol tracking [16]. This method results in only a small increase in complexity and offers improvement over tracking loop schemes in fading channels.

The adaptive filter in the LFSR has two purposes: it performs both depreading and channel equalization. With a small extension in length, this filter can serve another purpose: symbol tracking. Hosemann shows that an LMS adaptive filter can respond rapidly enough to compensate for a permanent channel delay. While in a traditional tracking mechanism, the receiver detects the presence of a phase offset, this adaptive tracking technique does not attempt this detection. Rather, the adaptation algorithm inherently corrects this offset through its update process.

In order to contain a drifting symbol within its bounds, the matched filter must extend beyond the length of single symbol. This extension can be as small as two samples: one on each side of the filter for a slowly drifting symbol. Those coefficients that do not correspond to the current symbol should converge to near zero with a stationary signal; as the symbol drifts, the coefficients will shift to include the extended coefficient in the drift direction. Regardless of the length of the extended filter, a mechanism must center the most significant coefficients in the matched filter. Otherwise, the coefficients will slide off the edge of the filter after tracking a drifting signal over many symbols. Since the received signals drift slowly relative to the symbol rate, this mechanism does not need to operate frequently. Refer to Chapter 4 for a discussion of the implemented centering technique.

Chapter 4

Architecture and Implementation

4.1 Overview

This chapter describes the architecture and implementation of the proto-Stallion receiver. This system, inspired by a Stallion-based architecture, both explores that potential of Stallion-based receivers while showcasing the capabilities of the Virtex FPGA as a soft-radio platform. The proto-Stallion architecture exploits multi-level modularity: it has layered I/O-Control and Processing Units, with a Processing Unit that is itself composed of several smaller functional blocks (proto-Stallion modules). These modules represent a library of Stallion programs that can be swapped into a single Stallion device. In a Virtex with enabled partial reconfiguration, these modules could be reorganized at run-time. This chapter details the functionality and the implementation of the I/O-Control and Processing Units, including the details of the Processing Unit's proto-Stallion modules. Although this chapter presents a specific radio implementation, this architecture itself is flexible; it can support a variety of applications within its fixed framework. This implementation's most novel contribution is its use of adaptive tracking; several attributes of the proto-Stallion architecture facilitate the utilization of this technique.

4.2 Proto-Stallion Design

The potential advantages of a Stallion-based software radio (as described in Chapter 2) motivate investigating system architectures prior to the device's availability. Short of a fabricated chip, a

Stallion functional emulation is the most informative method for evaluation of a Stallion-based architecture. Consequently, a Virtex-based Stallion emulation is currently under investigation at the Virginia Tech Configurable Computing Laboratory. This emulation is a complete Stallion model; it responds to data and programming information exactly as the actual device would. Therefore, a system developed for the Stallion emulation could use a Stallion chip platform with little or no changes. But the Stallion emulation is not without its drawbacks: its data processing speed is limited, and its high gate count prohibits the consideration of multiple Stallion designs. Also, there is a considerable body of research on FPGA-based receivers developed at Virginia Tech; a purely emulation-based receiver design cannot directly draw from this research nor can it advance future FPGA-based prototypes. The architecture described in this thesis takes a road somewhat removed from Stallion based systems, termed proto-Stallion. VHDL-described processing elements with Stallion-equivalent computational capabilities (called proto-Stallion modules) form the foundation of this architecture. Control and data manipulation hardware, also described in VHDL, supports these proto-Stallion modules; architectures utilizing the Stallion device would require identical support hardware.

Each proto-Stallion module is far removed from a Stallion emulation. Rather, these modules more closely resemble a fixed Stallion implementation of an algorithm. Individually, these modules support little reconfiguration, but several together represent a collection of configurations that can be swapped in a single or multiple Stallion design. While this loss of general reconfigurability may seem severe, the first Stallion software radio architectures would likely employ a similar method by drawing from a fixed set of functional configurations. To insure that the proto-Stallion modules map easily to the Stallion device, their designs recognize the capabilities and limitations of the chip—both in terms of processing resources and data-flow. For example: the I/O is within the limitations of the Stallion's data ports; each module uses no more than four multipliers; and, since the Stallion cannot easily share data among its configurations, the proto-Stallion modules preserve no data once they have completed their computations.

The proto-Stallion architecture's requires more than the proto-Stallion modules: it needs support hardware including I/O handling, data manipulation, and control. The organization of this support hardware resembles the layered radio architecture (LRA) outlined in [18]. While proto-Stallion design is not fully compliant with this architecture, it borrows the LRA's layering

of processing and interface functions. The proto-Stallion modules form the Processing Unit, described in Section 4.4; both I/O and Control functions form the I/O-Control Unit, described in Section 4.3. The LRA's configuration layer is missing in this architecture; its functionality is splint into the other layers. The proto-Stallion modules are effectively pre-configured and require little additional modification.

While the evaluation of a Stallion-based receiver initially motivated the proto-Stallion design methodology, the developed architecture is quite valuable as an FPGA-based design. Stallion programs inspired the swappable modular processing elements, but the Virtex's partial reconfiguration capabilities create a similar computing environment. Although the Stallion's RTR is more versatile, a Virtex armed with a collection of processing module configurations could efficiently implement a wide variety of receiver structures.

4.3 I/O-Control Unit

The proto-Stallion receiver features two distinct functional units: Processing and I/O-Control. This engenders a more flexible receiver: changes in either the interface or the algorithmic implementations do not require a reworking of the entire radio. This section describes the I/O-Control unit, which is responsible for the receiver's system level behavior. This unit controls and configures the Processing unit, and formats both the incoming data and the outgoing bit stream. A finite state machine forms the core of the I/O-Control unit's implementation; each state drives or maintains the Processing unit in a mode of operation. While the Processing unit has a wide variety of operations at its disposal, it has no system level intelligence; i.e. it possesses no knowledge of the appropriate order or duration of its functions. With feedback from the Processing unit, the I/O-Control unit supplies this intelligence, specifying the order and duration of each operation to correctly extract data from the received signal. Refer to Figure 4.1 for a state diagram depicting this operational order.

The I/O-Control Unit uses a set of operational codes to establish the Processing unit's function. Each code corresponds to a specific Processing unit action. Rather than directly using the enables and control flags of the Processing unit's modules, the I/O-Control unit transmits

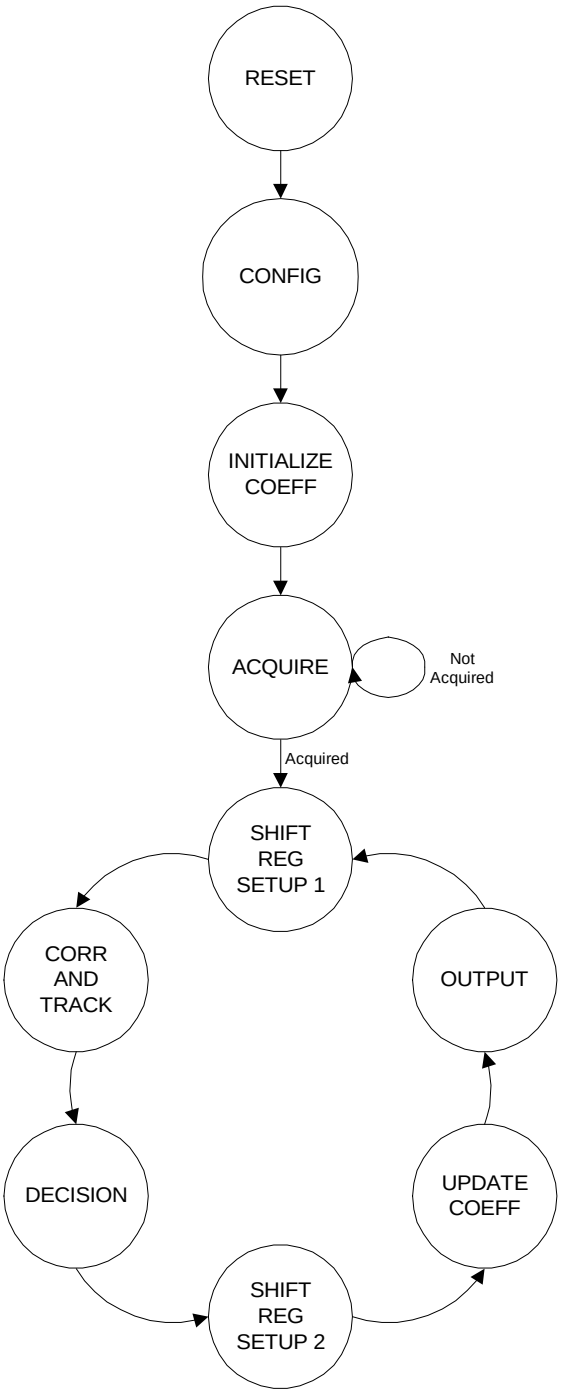

Figure 4.1. *I/O-Control Unit State Diagram*

only these defined mode codes, along with a flag that signals the Processing unit to switch to a new mode. This method both simplifies the I/O-Control unit and minimizes the bus width required to communicate between the two modules. Refer to Section 4.4 for a detailed description of each processing module mode.

The I/O-Control unit is also responsible for formatting input and output data. Input data can arrive from a number of sources: it can be internally generated test data, data from the host PC through the memory, or (in the future) from the Rockwell MRC. The unit sends the output bitstream to the memory where the host PC can access it.

4.4 Processing Unit

As its name suggests, the Processing unit handles the receiver's signal processing, implementing the algorithms detailed in Chapter 3. Figure 4.2 depicts the structure of the Processing unit. Nine modules compose this unit: four proto-Stallion modules, two data registers, two coefficient registers, and a parameter configuration module. Additionally, the Processing unit contains mode circuitry that translates the mode codes from the I/O-Control unit into the appropriate module control signals.

The four proto-Stallion modules each implement a basic function of receiver's baseband processing: the Acquisition module provides initial symbol synchronization; the Correlate and Track module performs matched filtering and channel equalization on the incoming spread spectrum signal, as well as maintaining symbol synchronization; the Decision module extracts the transmitted data bit and evaluates a portion of coefficient update process; and the Update Coefficients module completes the adaptation of the filter weights. A dedicated mode activates each of these proto-Stallion modules; the activated module then operates on the data set until the I/O-Control Unit initiates a new mode. For a radio implemented on a single Stallion device, this process of mode activation would be equivalent to loading a new Stallion program. This architecture's mode circuitry is simple; it decodes the I/O-Control Unit's instructions into the appropriate enables and control flags. In an architecture based on the Stallion device, however, this mode circuitry would need to be much more sophisticated. It would be responsible for storing and loading the Stallion's programs, effectively fulfilling the role of the LRA's

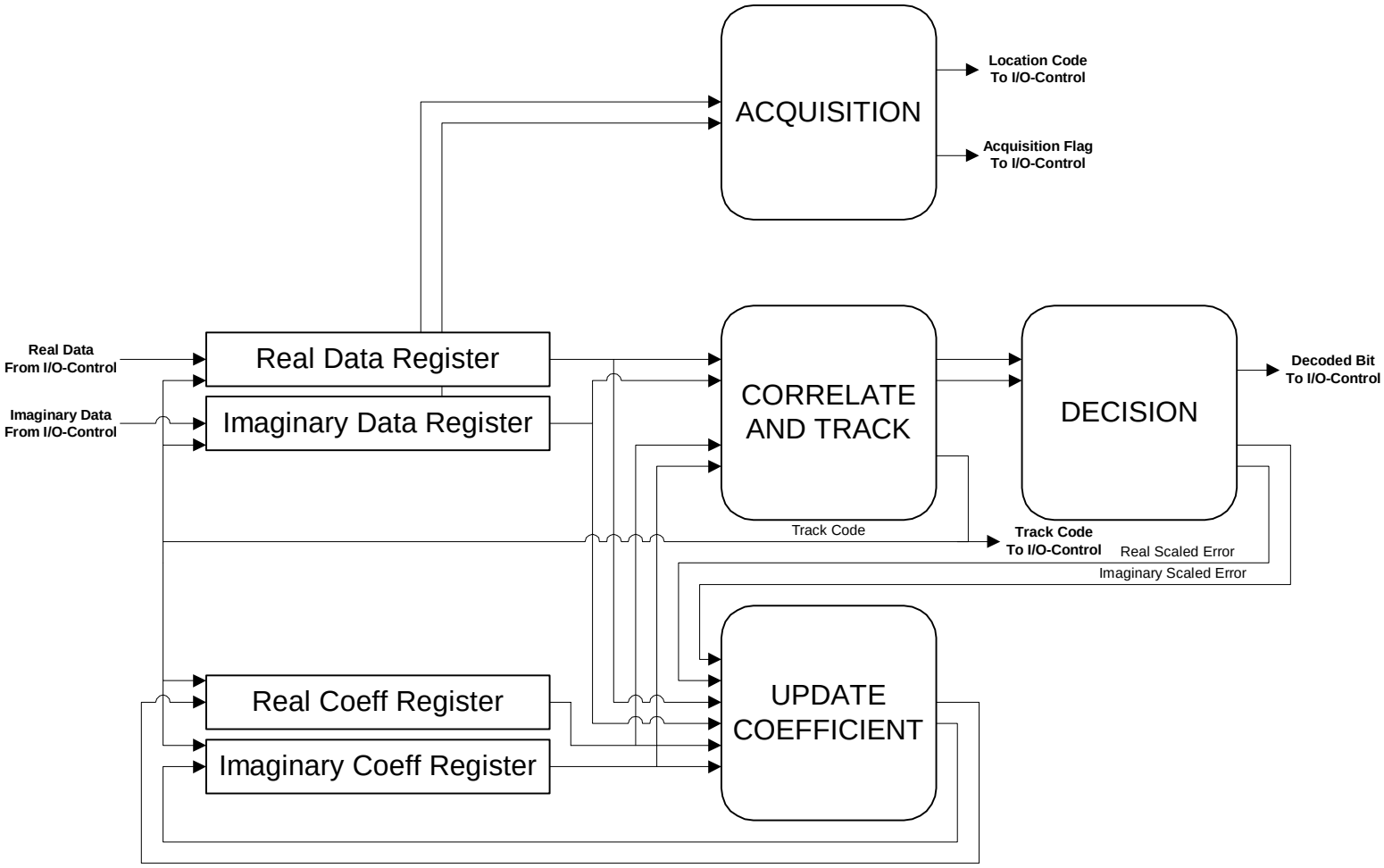

Figure 4.2. *Processing Unit Architecture*

Table 4.1. *Processing Unit Modes.*

Mode	Code	Enabled Modules	Function
Loading Coefficients	0000	Coefficient Registers	Loads initial coefficients
Acquiring	0001	Acquisition, Data Registers	Initial symbol synchronization
Advancing Data	0010	Data Registers	Aligns data following acquisition
Correlating	0011	Correlate and Track, Data and Coefficient Registers	Correlates signal with coefficients while maintaining symbol synchronization
Finish Correlating	0100	Correlate and Track	Finishes correlating while preserving data alignment
Deciding	0101	Decision	Performs hard bit decision and computes scaled error
Updating Coefficients	0110	Update Coefficients, Data and Coefficient Registers	Updates coefficients
Finish Updating Coefficients	0111	Update Coefficients, Coefficient Registers	Finishes updating coefficients while preserving data alignment
Shift Register Setup	1000	Data and Coefficient Registers	Changes register modes
Idling	1001	None	Stalls current operation
Configuring	1010	Configuration	Parameter configuration

Configuration Layer. Table 4.1 summarizes the Processing Unit's modes—included are each mode's function and activated modules.

The four register modules handle the support functions: they store and order the data and coefficients to insure proper processing. The architecture requires these registers because the proto-Stallion modules, like the Stallion device, cannot store large data sequences. The signal processing algorithms implemented on the proto-Stallion modules need the Processing Unit to successively perform two operations on the same data set. In order to support swappable modules on a single device, the Processing Unit must preserve this data set for use in subsequent operations, yet the Stallion device has no means to store data for use by future configurations. The four register modules provide this data preservation; they maintain both the data and coefficients through mode changes. These registers insure proper data alignment for the proto-Stallion module and simplify the I/O-Control module's responsibilities—it sends each data sample only once although that sample serves as an operand for several calculations. These registers also incorporate a crucial function of the adaptive tracking algorithm.

4.4.1 Acquisition Module

The Acquisition module is responsible for user detection; additionally, this module provides the initial alignment between the spreading code and the incoming data. This module operates on the I and Q data from the Data Registers; the spreading code, the required peak persistence, and allowable peak errors (termed caprice) are additional configuration inputs. An acquisition flag and a peak location code are outputs. This module implements the acquisition algorithm described in Section 3.5.1. It correlates the incoming data with the configured spreading code, then performs a maximum magnitude search over a symbol window's correlated outputs to recover the user's symbol timing.

Figure 4.3 depicts the structure of the Acquisition module. Two correlators (one each for the I and Q channels) perform matched filtering on the incoming data, producing a complex result each sample. A sub-module evaluates the magnitude of this complex result, followed by a maximum magnitude search over one symbol window. Subsequently, a persistence checker compares the located symbol maximum with the maximum from previous symbols. Should a peak persist for the appropriate number of symbols, the module has acquired a user; it sets the Acquisition Flag and Peak Location registers appropriately.

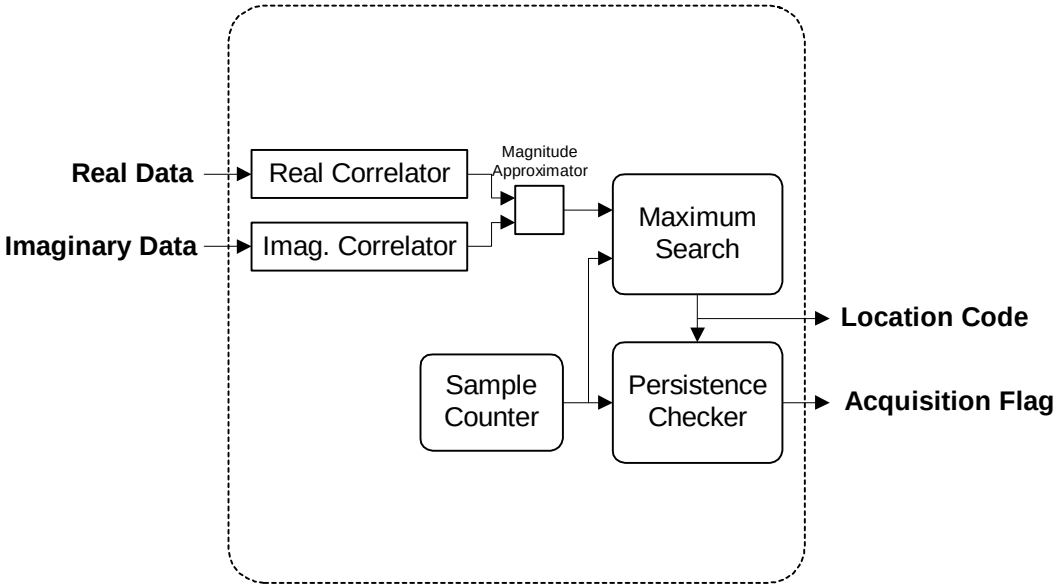

Figure 4.3. *Acquisition Module.*

A wide variety of structures will implement a correlation operation, but the computational and data flow characteristics of both the Stallion and FPGAs engendered a novel approach. The traditional FIR structure (Figure 4.4) employs a tapped delay line, with a coefficient multiplication at each tap. An adder tree sums the results of these multiplications to produce the filter output. This adder tree is problematic: it is difficult to scale for configurable-length filters and, when pipelined, can add considerably to the filter latency. The reverse FIR structure (Figure 4.5) corrects these shortcomings. In this structure the data sample is supplied simultaneously to all coefficient multipliers; delays in the adder chain restore the effect of the delay line. This implementation scales easily and its latency is no greater than the filter length [17]. Broadcasting the data sample to all multipliers in the filter does not, however, map efficiently to the data flow capabilities of either the Stallion or an FPGA. This research presents a new FIR filter structure that eliminates the adder tree while controlling the fan-out of the data sample: the registered reverse FIR structure.

The registered reverse FIR structure (Figure 4.6) combines the tapped delay line of the traditional filter structure with the delayed adder chain of the reverse filter structure. This

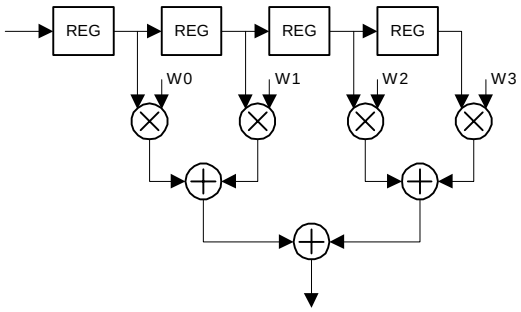

Figure 4.4. *Traditional FIR Filter Structure.*

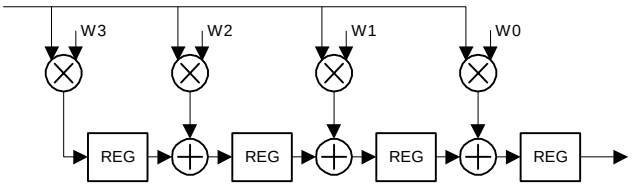

Figure 4.5. *Reverse FIR Filter Structure.*

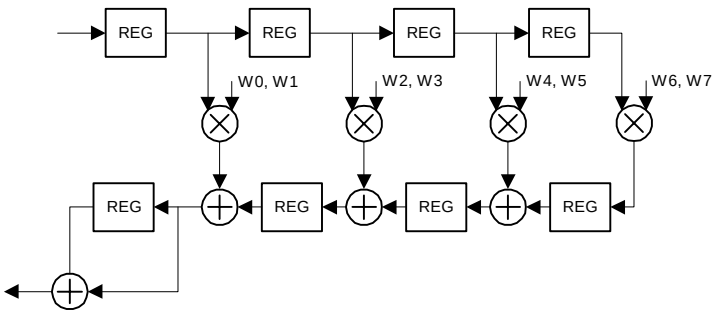

Figure 4.6. *Registered Reverse FIR Filter Structure.*

structure takes advantage of the oversampled signal; the structure functions only for an even number of samples per chip, but as a result both the tapped delay line and the registered adder chain are each only half as long as the symbol length. To maintain proper alignment, these two register structures pass their contents in opposite directions; the result is equivalent to the results of the other two structures. This structure is easily scalable, has latency equal to the filter length, and does not require the broadcasting of the input data. The resource requirements of this structure shrink further when we consider the values of the coefficients. Since the initial matched filter coefficients are limited to ±1, the multiplications reduce to selective negation operations. As adder/subtractors, the adders incorporate this negation, eliminating the need for separate multiplication operations.

The complex output of the correlators feeds a magnitude-approximator; since only relative magnitudes are significant in this module, a magnitude approximation is sufficient and avoids the use of a multiplier.

Using this magnitude approximation, the Acquisition module performs a maximum search over each symbol. Figure 4.7 depicts the structure of the maximum search sub-module. The sub-module, which resets once per symbol, first compares the stored peak value with the magnitude of the current sample with a subtractor and a sign bit test. If the current value exceeds the stored peak value, then a pair of registers stores the current value and its position as the new symbol peak. Once this sub-module analyzes a complete symbol window, the result corresponds to the symbol peak.

The persistence checker compares this symbol peak with the peak from previous symbols. Figure 4.8 depicts the structure of the persistence checker sub-module. This process only considers the peak location—the peak magnitude is no longer significant. The sub-module

logically compares the symbol peak with the stored symbol peak; if they are equal, the persistence counter increments, recording the number of times a peak remains at a given location. If they are not equal, the caprice counter increments, indicating the number of times a

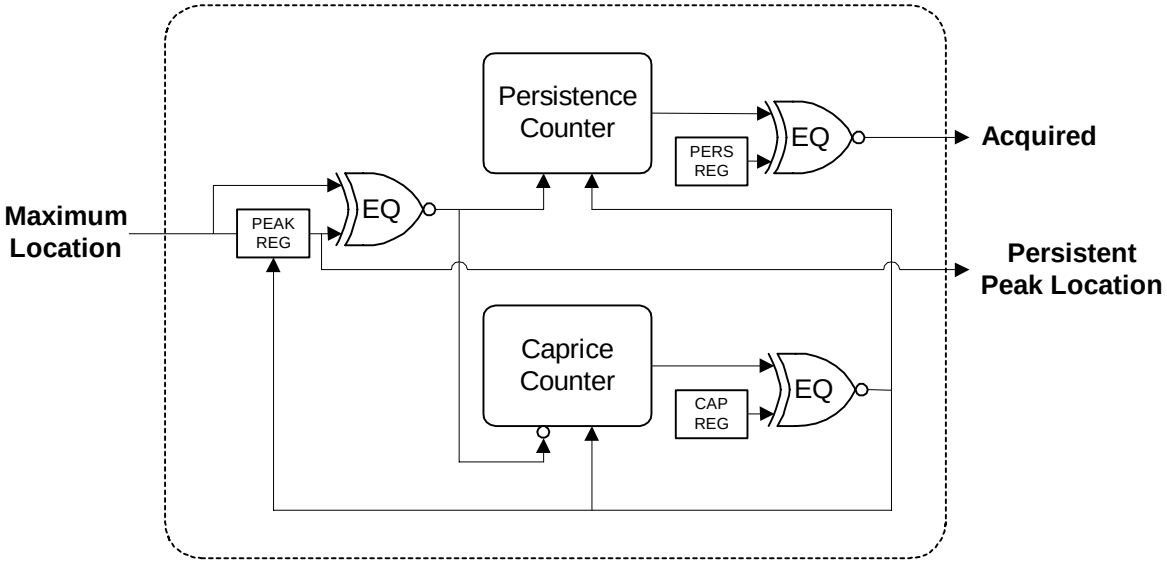

Figure 4.7. *Maximum Search Sub-Module.*

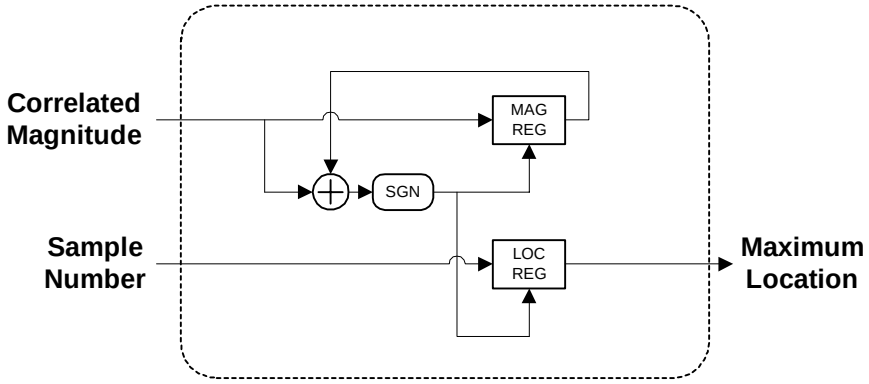

Figure 4.8. *Persistence Checker Sub-Module.*

peak has appeared at a different location. The sub-module compares both of these counter values to parameters set by the Configuration module; if the persistence count meets its threshold, the sub-module sets the acquisition flag and the receiver begins extracting data at the registered peak location. However, if the caprice count exceeds its threshold first, the register abandons its peak and replaces it with the current symbol peak.

4.4.2 Correlate and Track Module

The Correlate and Track module correlates the receiver data with the filter coefficients—initially corresponding to the user's spreading code—to despread the signal. As the Update Coefficients module changes these filter weights, this filter also provides channel equalization. The coefficients may drift to track a moving signal; to compensate, this module monitors the coefficients and signals the Data and Coefficient Registers to realign, if necessary. It monitors the coefficients by performing three additional correlation operations: the module correlates the coefficients with the user's original spreading code at three alignment points. This module takes the user data and coefficients as its inputs; once per symbol, the module outputs the symbol's correlated result, as well as a tracking code that signals the movement of the coefficients.

Figure 4.9 depicts the structure of the Correlate and Track module. A primary correlator filters the incoming complex data by the corresponding coefficients. Three auxiliary correlators operate in parallel to the main correlator; these track the filter coefficients as they drift through the update process.

The complex correlation operation is the most important function of the Correlate and Track module. The Stallion's processing resources dictate a key implementation issue of this

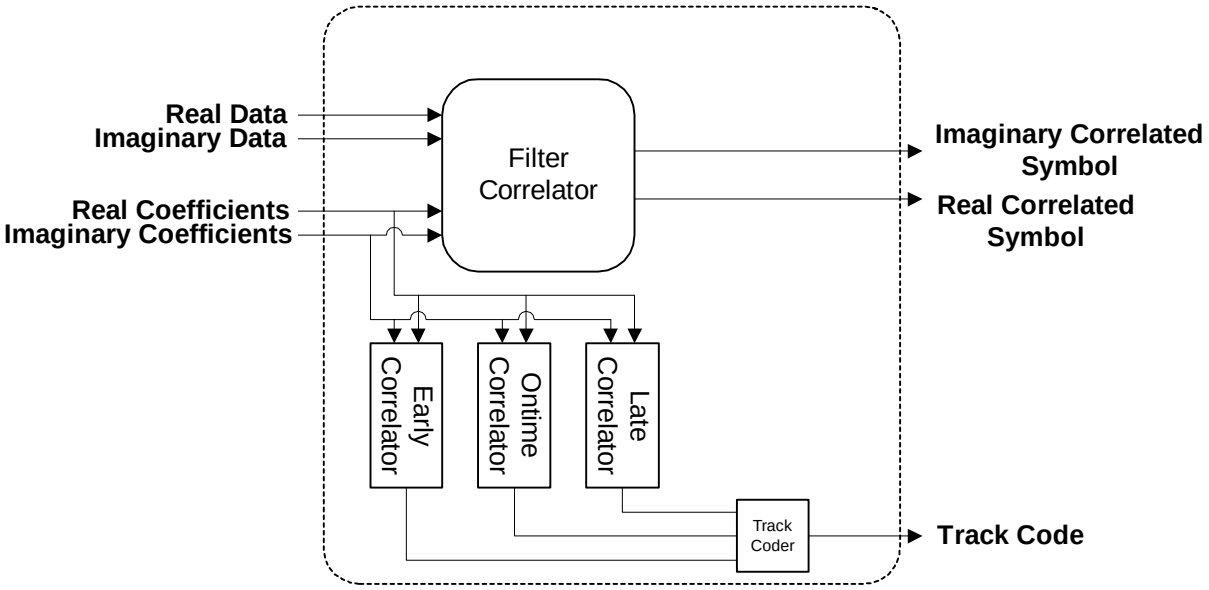

Figure 4.9. *Correlate and Track Module.*

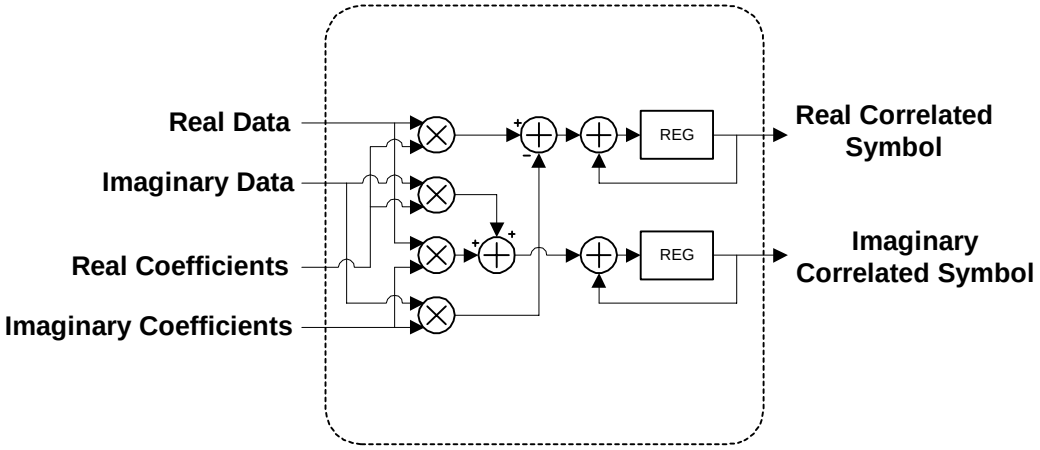

Figure 4.10. *Complex Correlator Sub-module.*

module: since the Stallion has four multipliers and each complex multiplication requires four real multiplications, the module can only execute one complex multiplication per clock cycle. This limits options for the correlator's structure; with only one complex multiplier the module must sequentially compute each product, employing no parallelism or pipelining. Figure 4.10 depicts the structure of this correlator. A complex accumulator follows a complex multiplier; once per symbol, the correlator outputs and resets this accumulator, producing the correlated symbol value.

Figure 4.11 depicts the structure of the tracking correlators. Each of the three tracking correlators filters the coefficients with the original spreading code. As with the acquisition correlator, the tracking correlators exploit the spreading code's simple values; since the code values are limited to ±1, the Correlate and Track module performs these filter operations with only additions and subtractions. An adder/subtractor sums the current coefficient with the registered total; the register updates with the resulting value. The corresponding spreading code sample selects the function of the adder/subtractor. The sub-modules subsequently compute the magnitudes of their complex correlated outputs.

The three tracking correlators simultaneously operate on the same coefficients; their spreading code bits, however, are offset. One correlator's code alignment corresponds to centered coefficients (on-time), one corresponds to coefficients advanced by one sample (early), and one corresponds to coefficients delayed by one sample (late). A track encoder compares the outputs of the three track correlators. If the on-time magnitude is largest, the coefficients require no adjustment. If either the late or the early magnitude is largest, the encoder signals that the

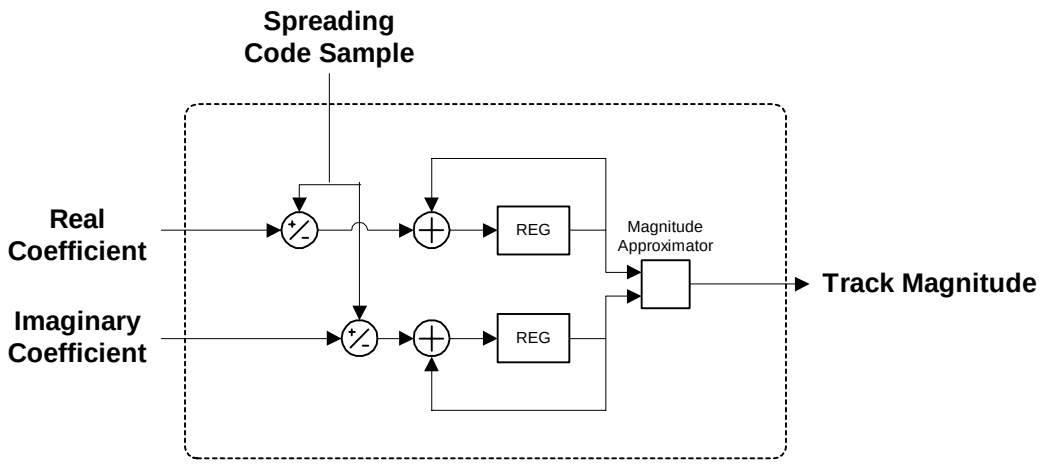

Figure 4.11. *Tracking Correlator Sub-module.*

coeffficients need to be repositioned within their register. Table 4.2 summarizes the encoder's track codes.

Table 4.2. *Track Codes.*

Track Code	Largest Magnitude
0x	On-time correlator
10	Early correlator
11	Late correlator

4.4.3 Decision Module

The Decision module demodulates the transmitted data bits from the Correlate and Track module's correlated symbol value. This module also uses correlated symbol value to calculate the scaled error used by the Update Coefficients module.

The Decision module implements the bulk of the LMS update equation described in Chapter 3. These equations are repeated here for convenience. Since this radio uses a differential-encoding scheme, the module must consider both the current and the previous correlated result. The module multiplies these two correlated values,

$$z(n) = y(n) \cdot y^*(n-1) \tag{4.1}$$

then performs a hard decision on the result *z*,

$$d(n) = sign\big(y(n)\big) \tag{4.2}$$

demodulating the DBPSK signal to produce the estimated data bit *d*, valued at ±1.

The Decision module also uses its correlated input to calculate the LMS error. In order to streamline the Update Coefficients module, the Decision module consolidates most of the multiplicands of the update equation into a single term: the scaled error value, ε. The unscaled error value is equal to the difference between the estimated data bit and its actual value, or

$$e(n) = d(n) - z(n) \tag{4.3}$$

This becomes the scaled error value as it is multiplied by the previous correlated input and a scaling factor μ, or

$$\varepsilon(n) = \mu \cdot e(n) \cdot y(n-1) \tag{4.4}$$

The Update Coefficients module adapts the filter weights using this value.

The other proto-Stallion modules perform the same operation many times over a large data set; the Decision module, however, performs its operation sequence only once per symbol. Three of these operations involve complex multiplication, and the Stallion can perform only one complex multiply per clock cycle. Since the Decision module operates only once per symbol, using multiple configurations has prohibitive overhead. Instead, the Decision module time-multiplexes its complex multiplier in a single configuration, greatly reducing configuration overhead; the price paid for this gain is that the Decision module has the most complicated data flow of any of the modules.

Figure 4.12 depicts the structure of the Decision module. Eight multiplexors feed the complex multiplier—one for each of the inputs to the four real multipliers. A set of registers

Table 4.3. *Decision Module Operation Sequence.*

Process #	Complex Multiplier Operands	Result
00	Current correlated symbol, Previous correlated symbol	Differential correlated symbol
01	Previous correlated symbol, Step size	Scaled correlated symbol
10	Scaled correlated symbol, Error	Scaled error

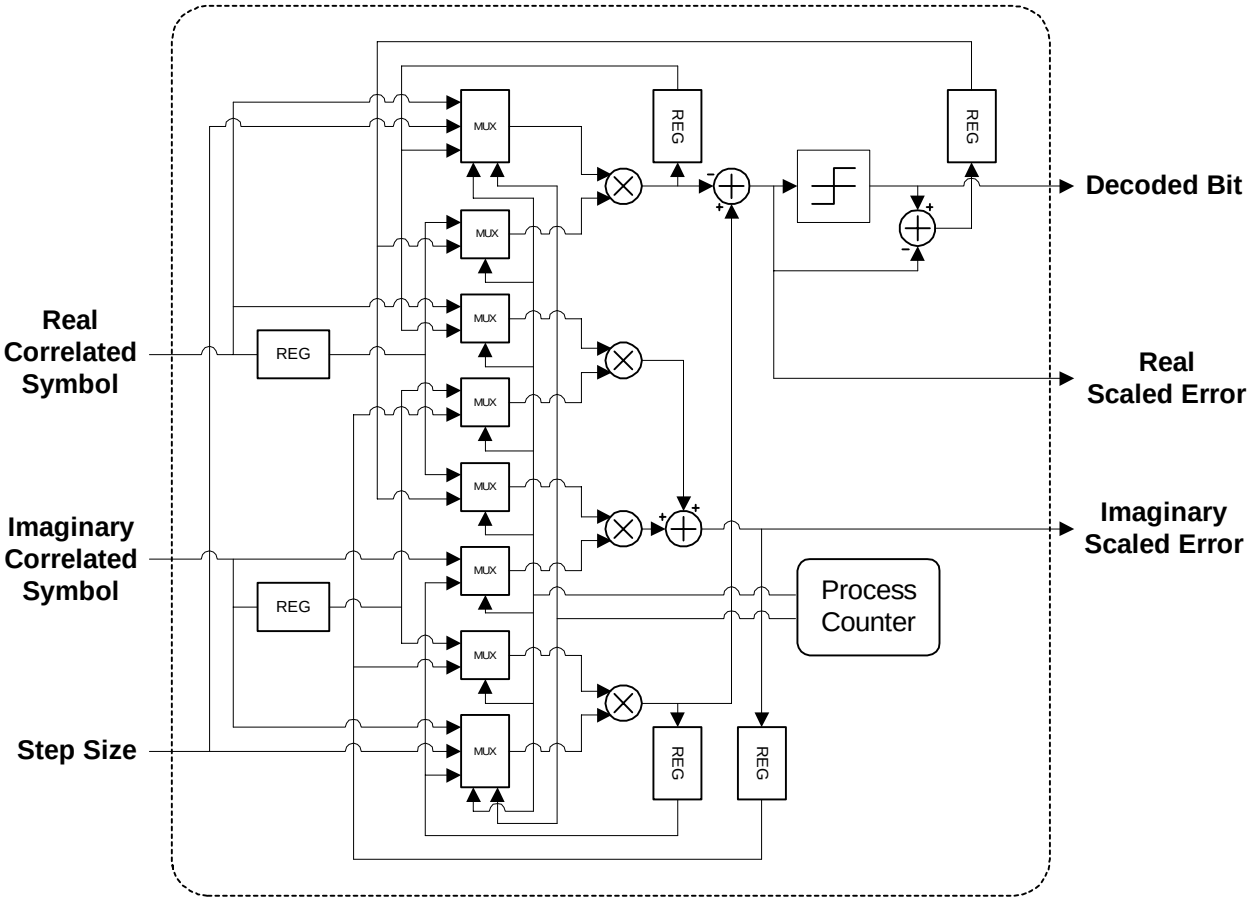

Figure 4.12. *Decision Module*

captures the appropriate inputs for these multiplexors and a process counter selects the proper operands for the multipliers. Table 4.3 summarizes the order of operation for the Decision module.

4.4.4 Update Coefficients Module

The Update Coefficients module adapts the filter coefficients using the scaled error value produced by the Decision module. For each coefficient, the module multiplies the scaled error value with the corresponding data value and then sums this product with the previous coefficient. This module thus completes the coefficient update equation described in Chapter 3, repeated here:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \varepsilon \cdot \mathbf{r}(n) \tag{4.5}$$

Figure 4.13 depicts the structure of the Update Coefficients module. The simplest of the proto-Stallion modules, the Update Coefficients module performs a full complex multiplication on the scaled error value and the data. The module then adds the resulting product to the corresponding previous coefficient; a delay insures the proper alignment of this addition. This complex sum is the new coefficient. The Update Coefficients module outputs this value for storage in the Coefficient Registers.

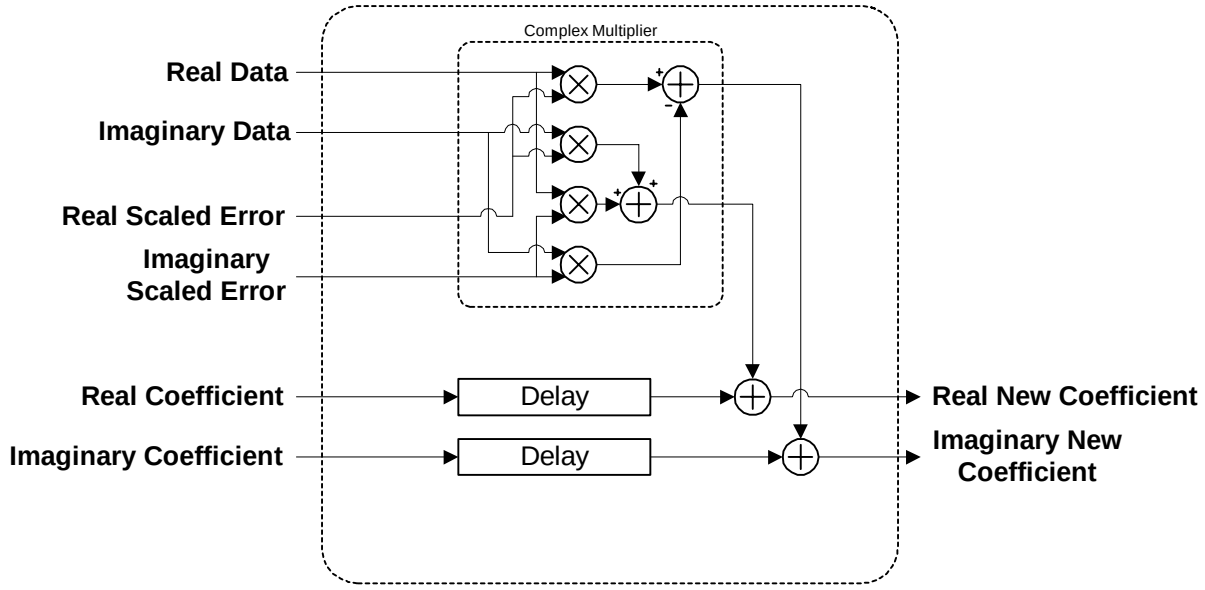

Figure 4.13. *Update Coefficients Module*

4.4.5 Data and Coefficient Register Modules

The Processing Unit contains four shift register modules: the real and imaginary Coefficient Registers and the real and imaginary Data Registers. In addition to storage, these register modules order the data and coefficients according to the specific processing needs of the proto-Stallion modules.

The Data and Coefficient Registers serve several distinct purposes in the proto-Stallion architecture. The Data Registers buffer the interface between the I/O-Control and Processing units, properly reusing the incoming data. All four registers provide a communication medium between the Correlate and Track module and the Update Coefficients module. These two proto-

Stallion modules successively operate on the same operands; the register modules preserve the data and coefficients for the second operation. Finally, the Register modules implement the centering of the coefficients required by the adaptive tracking algorithms.

The architecture of the Coefficient Registers is straightforward. They have two modes of operation, termed fresh and stale. While operating in fresh mode, the registers shift their contents while accepting new coefficients at their tails; in stale mode, the registers rotate their coefficients, returning the values outputted at the head of the register to the tail's input. Stale mode feeds the Correlate and Track module, preserving the coefficients for reuse. Fresh mode provides the Update Coefficients module with coefficients; at this time the Coefficient Registers also replace their contents with the new filter weights that the Update Coefficients module calculates. The Coefficient Registers also use fresh mode at reset to initialize to the user's spreading code. As the module switches from fresh to stale mode, the filter adjusts according to the results of the previous iteration of the adaptive tracking algorithm. The filter weights can be advanced or delayed by one coefficient, as appropriate. In either case, the shift displaces an edge coefficient; the register rotates, inserting the edge value on the opposite end.

The *Data Registers* share many features with the Coefficient Registers, but they also possess additional capabilities for data reuse. The Data Registers have fresh and stale modes; the stale mode supports the Correlate and Track module, while fresh supports the Update Coefficients module. Fresh mode also supplies the Acquisition module with its data through a dedicated output. As with the Coefficient Registers, the Data Registers perform adaptive tracking adjustments upon the change from fresh to stale mode. Unlike the Coefficient Registers that insert a rotational value following this shift, the Data Registers must insert the data value of the corresponding time sample to maintain proper data/coefficient alignment.

The unusual requirements of the adaptive tracking filter force the Data Registers to preserve a portion of their contents between symbols. For adaptive tracking to behave properly, the adaptive filter must consider samples from the previous and next symbols in addition to the current symbol. Therefore the filter must retain some of the samples of the current symbol as the next symbol enters the register. Additionally, the I/O-Control Unit has already advanced the processing module a portion of this next symbol; the register must retain this portion as well.

4.4.6 Parameter Configuration Module

In an architecture based on the Stallion device, the configuration module would have the responsibility of storing and loading the Stallion programs for each functional module. In the proto-Stallion architecture, its responsibilities are far less significant. The Parameter Configuration module allows basic computational parameters to be changed in runtime. While the feature provides only a glimmer of the capabilities of a true RTR device, it is an important capability that facilitates testing of the receiver and lays the foundation for greater things to come.

The Configuration module is based on a state machine; once the module is enabled, it loads new values for all of the parameters it controls. While this is not the most time efficient means to reconfigure, the number of parameters is small and reconfiguration should occur infrequently. Table 4.4 describes the parameters that this module controls. Additionally,

Table 4.4. *Configuration Parameters*

Parameter	Default Value	Corresponding Module
Step size	0100000000 (0.5)	Decision
Persistence	0000000100 (4)	Acquisition
Caprice	0000000010 (2)	Acquisition
Spreading code	011110101100100	Acquisition, Correlate and Track

Configuration manages several static parameters that may be configurable in future implementations of this radio. These include spreading code length (15 chips), oversampling factor (4), and filter extensions for tracking (4 samples per side). The settings result in a length 60 filter for acquisition and a length 68 filter for the adaptation.

4.5 Fixed Point Scaling Issues

Signal processing algorithms are often developed using infinite precision calculations; their implementations, however, must be limited precision. The proto-Stallion architecture, like the Stallion device, uses fixed-point 16-bit words. The implementation of the receiver's algorithms must consider fixed-point effects in order optimize its results. In particular, the use of fixed

point numbers necessitates accommodation in three areas: coefficient storage, coefficient adaptation, and Decision module scaling.

The Coefficient Registers store the coefficients; their scaled values must permit growth without severely limiting precision. The I/O-Control unit initializes all coefficients to $\pm 1/Np$ so that with a perfect, noiseless transmission, the correlated output will be one. Since this receiver employs an adaptive filter, these coefficients are not chained to their initial values; they can grow or shrink by significant amounts. In order to allow for growth while maintaining high precision, the Coefficient Registers provide a sign bit, four growth bits, and 11 precision bits for each 16-bit coefficient. Fixed-point Matlab simulations revealed that this configuration provided adequate dynamic range for the expected operation environment.

A peculiar problem arises when using fixed-point two's-compliment numbers for adaptive filtering. Digital systems typically employ truncation when reducing the precision of a result. The result of a two's-compliment truncation always reduces the value of the number, pushing it toward $-\infty$. Unfortunately, this produces two undesirable effects in adaptive filters. First, the coefficients are asymmetrically updated—coefficients that become more negative do so by a slightly larger amount than those that become more positive. Second, when the coefficients converge, the coefficients do not oscillate around their ideal values. Numbers smaller than the smallest representable scaled error value are represented as zero if they are positive, but as a small nonzero value when negative. The coefficients therefore adapt in only one direction, eventually losing convergence. The solution is to round rather than truncate; this restores the symmetry to the adaptation process.

The Decision module computes a series of multiplications; each of these multiplications produces a radix change in the result. The module must track these changes such that the resulting scaled error value is appropriately positioned for the coefficient update. The module maintains appropriate alignment by operating on a shifted product. Each 16-bit multiplication produces a 32-bit result, only 16 of which are needed for the next operation. Rather than always selecting the topmost bits, selecting a 16-bit window of less significant bits performs a left shift on the result, restoring the desired radix point position.

4.6 Implementation Performance

The proto-Stallion architecture, including both the complete Processing and I/O-Control units, occupied a single XCV1000 device on the SLAAC-1V. Table 4.5 summarizes the utilization of the X1 processing element. This receiver processes one symbol in approximately 165 clock cycles, corresponding to a data throughput of 300 kbits per second.

Table 4.5. *Utilization of X1 (XCV1000)*

Number of Slices:	10,250 out of 12,288 83%
Slice Flip Flops:	12,441
4 input LUTs:	7,877 (10 used as a route-through)
Number of bonded IOBs:	123 out of 512 24%
Number of GCLKs:	1 out of 4 25%
Number of GCLKIOBs:	2 out of 4 50%
Number of RPM macros:	12
Total equivalent gate count for design:	186,117
Maximum clock speed	54.9 MHz

Table 4.6 summarizes the computational capabilities of the proto-Stallion receiver. It describes the number of operations (multiplications and additions) per clock cycle each module performs. These modules operate sequentially in the current radio, but this implementation supports simultaneous operation. Therefore, the receiver can perform up to 104 calculations per cycle; at 55 MHz, this corresponds to 5.7 billion operations per second. In sequential mode, however, this rate drops considerably: the radio averages 3.5 billion operation per second when acquiring and 750 million operations per second following acquisition. Note that this implementation pipelines much of its overhead; a DSP design would waste instructions on non-computational functions, producing lower data throughput at an equivalent operational rate.

Table 4.6. *Proto-Stallion Receiver Computation Capabilities*

Module	Multiplications	Additions
Acquisition	0	65
Correlate and Track	4	20
Decision	4	3
Update Coefficients	4	4
TOTAL	**12**	**92**

Chapter 5

Results

5.1 Overview

This chapter assesses the performance of the implemented proto-Stallion receiver. In particular, this chapter concentrates on this receiver's adaptive tracking algorithm. While other works have characterized the performance of the receiver's other algorithms, the proto-Stallion receiver represents the first implementation of this novel tracking technique.

5.2 Testbed

The focus of this research is the baseband processing of a software radio; however, a complete soft-radio receiver also requires a multi-band RF front end and a wideband Analog to Digital converter. To realize this system, the proto-Stallion receiver interfaces with the Rockwell Collins Miniature Radio Codec [19]. This interface, however, is incomplete. Rather, the radio accepts host-generated test data to verify its performance.

The proto-Stallion receiver resides exclusively in the SLAAC-1V's X1 processing element. Since X1 does not have a FIFO connection, two PE SRAMs must transfer data between the host and the SLAAC-1V. One holds Matlab-generated input data while the other holds both the radio's demodulated output bits and several operational statistics. While Memory 0 is large, it can hold only 4000 symbols at 60 samples each. Many characterizations of the receiver

require much larger data sets, so the radio is configured to process 4000 symbol blocks indefinitely. The receiver processes each block, then waits for the host program to refresh the memory before continuing processing.

5.3 The Effect of AWGN on Adaptive Tracking

Channel noise plays a critical role in the behavior of the proto-Stallion receiver's adaptive tracking. The LMS algorithm applies its error signal uniformly across all coefficients (refer to Equation 3.4); only variations in the received samples diversify the new filter weights. Consider the case of a perfect, noiseless transmission (SNR=∞ dB)—disregarding their signs, the received samples are uniform. Any sampling errors will therefore result in equal adaptation across all filter weights. Yet, for a single sampling error with a spreading code oversampled by four, only one quarter of the coefficients require adaptation. The other coefficients needlessly adapt away from the ideal solution; i.e. a shifted version of the initial coefficients. Figure 5.1 illustrates the coefficients of a MATLAB adaptive filter after it has tracked 75 missing samples in a noiseless

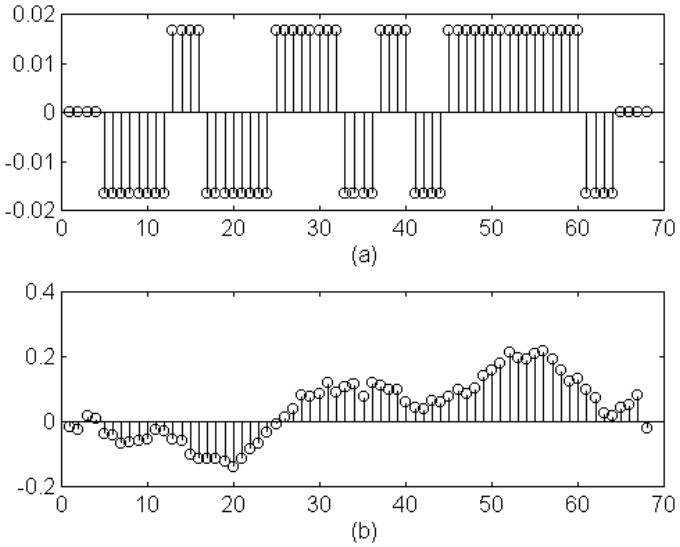

Figure 5.1. *Adaptive Tracking Coefficients: (a) initial; (b) after 75 missamplings at SNR=∞ dB, μ=0.5.*

channel. Note that the filter retains only a very coarse representation of the original coefficients. This filter has lost the spreading code's good auto- and cross-correlation properties. As a result, the filter cannot continue to track the drifting signal or isolate its user in MAI.

The adaptive filter's behavior changes dramatically as it tracks a drifting signal in an AWGN channel. Unlike the noiseless case, the received samples are not uniform. These variations in the received samples produce variations in updated coefficients; as a result, the adaptation algorithm does not regularly force coefficients to converge away from the ideal. The data sample deviations, although random, allow the LMS algorithm to ultimately converge to a shifted approximation of the original coefficients. Figure 5.2 illustrates the coefficients of a MATLAB adaptive filter that has tracked 75 missing samples at 0 dB SNR. While the high noise has introduced randomness to the coefficients, the weights retain the character of the spreading code. Under the tested conditions, this filter will track a drifting signal indefinitely. Although it will fail eventually, the number of symbols required to observe this failure currently exceeds the testbed's capabilities.

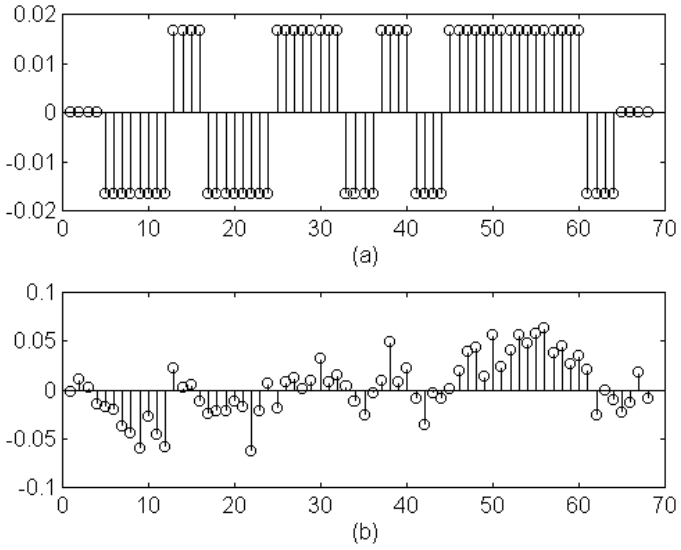

Figure 5.2. Adaptive Tracking Coefficients: *(a) initial; (b) after 75 missamplings at SNR=0 dB, μ=0.5.*

5.4 Step-size and Adaptive Tracking

The step-size μ scales the LMS algorithm's error value. As discussed in the previous section, the LMS algorithm exploits the received signal's Gaussian noise to maintain synchronization. The step-size plays a critical role in this synchronization: a large step-size amplifies low noise values when SNR is high, producing the coefficient variations necessary for tracking. When using a small step-size, these variations are not significant and tracking fails.

When SNR is low, however, a large step-size is a liability. The random coefficient variations overwhelm the spreading code, forcing incorrect bit decisions and the collapse of the matched filter. A smaller step-size dampens these high error values, allowing the filter to continue to track even when the noise power exceeds the signal power.

Table 6.1 provides the SNR operational range for several step-sizes. The minimum step-size represents the point below which the filter is overwhelmed by bit errors. The failure occurs after only a few symbols, and it was tested in 1 dB increments. The maximum step-size represents the point above which the filter cannot track a signal drifting at a rate of one missample per 1000 samples. Unlike the minimum SNR test, the output of the filter must be observed over several thousand symbols to determine if it can maintain synchronization. Since this is a more involved test, it was only performed at the following SNRs: 0, +3, +5, +8, and +10 dB. No step-size worked below –8 dB and none worked above +8 dB. At the low end, the

Table 5.1. *Operational SNR Ranges*

Step size	Minimum SNR	Maximum SNR
0.75	0	8
0.5	-3	5
0.25	-5	5
0.125	-5	3
0.0625	-6	3
0.03125	-8	3
0.015625	-8	0

precision of the calculation limits performance. Since the receiver is fixed point, as the step-size shrinks, so does the number of significant bits in the scaled error. At the high end, increasing the step size to amplify the low noise values interferes with the receiver's convergence.

5.5 Sampling Error Rate and Adaptive Tracking

To assess the overall performance of adaptive tracking, the receiver test bed processed a data set in three mode: as an adaptive receiver with adaptive tracking, as an adaptive receiver with perfect tracking, and as a matched filter with perfect tracking. For each of these modes, the receiver considered data with an SNR ranging from –9 dB to 5 dB and a timing error period of 1000 samples. These noise levels produce many bit errors; the receiver does not need to process an extreme number of symbols to produce a statistically significant number of bit errors. The receiver test bed generated each of the three result types. This allows evaluation of the adaptive tracking algorithm with any quantization effects applied uniformly across the tested modes. Figure 5.3 summarizes the result of this experiment. The adaptive tracking receiver approaches the performance of the ideally tracking adaptive receiver, lagging by at best 4 dB.

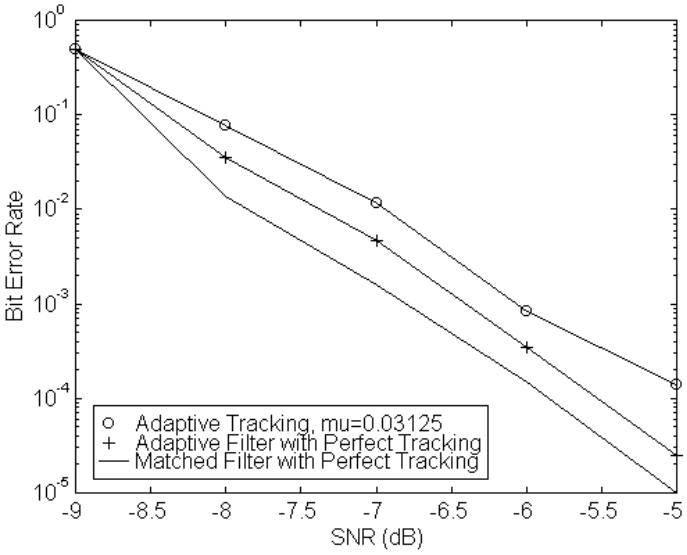

Figure 5.3. *BER for Proto-Stallion Receiver*

A receiver exhibiting perfect tracking is equivalent to a receiver operating on a signal with no drift. The performance of the adaptive receiver with perfect tracking thus predicts the performance of the adaptive tracking receiver operating on data with an infinite timing error period. Therefore, the adaptive tracking algorithm's performance will improve as the timing error period increases. Figure 5.4 depicts the behavior of proto-Stallion receiver at several timing error periods. The graph considers the ratio of the BER of the proto-Stallion receiver incorporating adaptive tracking to the BER of this receiver with perfect tracking.

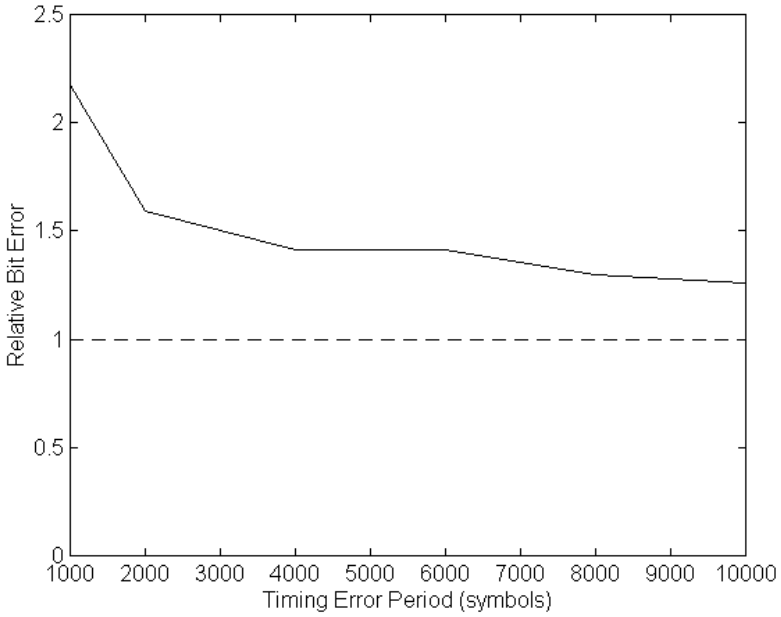

Figure 5.4. Relative *BER vs. Timing Error Period at μ=0.03125, SNR=-8dB.*

Figure 5.4 suggests that the operational SNR ranges summarized in Table 5.1 are dependent on the timing error period. While the lower bound is indepent of the timing error period, the upper bound will vary as the period increases. A filter adapting over a long period of time with low Gaussian noise receives more high-power noise pulses than an equivalent filter adapting over a shorter time interval. These larger noise pulses are critical to adaptive tracking. A receiver analyzing a signal with a longer timing error period has a better opportunity to receiver such pulses, and can therefore track in a higher SNR environment.

Chapter 6

Conclusions

6.1 Summary

This thesis introduced the proto-Stallion receiver, an FPGA-based soft-radio. Through the proto-Stallion receiver, this research both investigated Stallion-based radio design and pioneered a Virtex-based soft-radio baseband processor. Following the introduction in Chapter 1, Chapter 2 overviewed configurable computing for soft-radio. The chapter included a discussion of the Stallion—the CCM that inspired the proto-Stallion architecture. This device provides rapid run-time reconfiguration through a highly connective stream-based mesh of coarsely grained functional units. The foundation of the proto-Stallion receiver's implementation is the SLAAC1-V FPGA platform; its Virtex devices provide the speed and computational capabilities vital for software radio processing. Also, this chapter introduces other contemporary CCM-based software radios and briefly compares them to the proto-Stallion receiver.

Chapter 3 provides a background in spread spectrum systems and introduces the proto-Stallion receiver's algorithms. The chapter discusses both adaptive filtering and synchronization techniques. The proto-Stallion receiver's use of adaptive tracking is particularly significant. This tracking scheme offers reduced computational complexity over loop tracking schemes while providing significant gains in fading channels.

Chapter 4 details the architecture and implementation of the proto-Stallion receiver. The receiver contains two units: Processing, which performs the radio's signal processing, and I/O-Control, which handles the radio's interface, as well as controlling and configuring the Processing unit. The Processing unit is further divided into several computational entities termed proto-Stallion modules. Each implements a basic function of receiver's baseband algorithms.

While this prototype organizes these modules to implement a specific receiver, the flexibility of the architecture lends itself to a more generalized radio.

Chapter 5 details the performance of the proto-Stallion receiver. This chapter highlights the performance of the adaptive tracking algorithm. This algorithm's dependence on noise is particularly significant. While the tracking algorithm fails to adequately track a perfect signal, it performs well in channels with significant noise levels.

6.2 Future Work

This research creates several opportunities for future investigations. These fall into two broad categories: architectural improvements to the proto-Stallion receiver and further exploration of adaptive tracking using this receiver.

6.2.1 Proto-Stallion Architectural Improvements

While verification of the proto-Stallion receiver's baseband architecture was the primary goal of this research, a wireless interface is critical to the utility of this receiver. Therefore, the finalization of the Rockwell Collins MRC interface should be a priority of future research. The current receiver's memory-based interface should map easily to MRC data. This data requires buffering to prevent data interruptions; a separate process could manage the memory as a FIFO for the MRC. This would allow the current I/O-Control module to continue its memory-based operation with little modification.

The proto-Stallion receiver supports several run-time reconfigurable parameters. However, all parameters relating to the filter length are fixed at the time of the receiver's implementation. These parameters include the length of the spreading code, the samples per chip, and the tracking filter extensions. Manipulating these parameters requires replacing the Data and Coefficient Registers with reconfigurable equivalents. While this is a nontrivial modification, it would add considerably both to the receiver's flexibility and to its ability to characterize the behavior of the adaptive tracking algorithm.

The addition of broad reconfigurability would elevate this radio architecture to its full potential. Generalizing the I/O-Control unit with a reconfigurable state machine and extending the library of proto-Stallion modules would permit the implementation of a wide variety of receiver structures on a single platform. The utilization of the Virtex device's partial reconfiguration would permit sweeping functionality changes at run time, further enhancing the radio's abilities.

6.2.2 Further Adaptive tracking Investigations

There are a number of for further characterization of the adaptive tracking algorithm using the proto-Stallion receiver. Enhanced performance in a fading channel in a key motivation for the use of adaptive filter techniques; while [3] demonstrated a performance gain in simulation, the behavior of this implementation of the adaptive tracking algorithm in a fading channel remains unexplored. Since this investigation does not require any hardware modifications, it is the most straightforward means to expand the knowledge of adaptive tracking.

Support for a reconfigurable filter length enables several adaptive tracking investigations. The use of four samples per chip in the proto-Stallion receiver is generous; in other systems, the computational savings with two samples per chip often outweighs the performance gains that accompany higher sampling rates. The adaptive tracking algorithm particularly benefits from high oversampling rates. At four samples per chip, a single sample offset still produces a matched filter output at 75 percent of the aligned level. At two samples per chip, however, this output drops to 50 percent; this smaller peak is more difficult to reliably track. Characterizing the performance of adaptive tracking at several oversampling levels would allow designers to streamline the implementation for either minimized complexity or maximized throughput.

Similarly, a reconfigurable filter length permits an investigation of the effect of the tracking filter extension length. While MATLAB simulations indicated that increasing the length of these filter extensions did not benefit—and often hurt—the performance of adaptive tracking, it would be beneficial to characterize the effect of extension length in order to optimize the allocation of computational resources.

6.3 Conclusions

Inspired by the Stallion configurable computing machine, this research implemented a DS-CDMA single-user LMS adaptive filter on a Virtex FPGA platform: the proto-Stallion receiver. This architecture is modular and expandable. A state machine selectively enables a series of computational modules to implement its processing algorithms; with a reconfigurable state machine and a library of processing modules, this architecture could support numerous radio structures. The proto-Stallion receiver employs adaptive tracking, a novel technique that uses the adaptation process to maintain symbol synchronization with a drifting signal. The results of the proto-Stallion receiver characterized the behavior of the adaptive tracking implementation. Most significantly, adaptive tracking requires a substantial noise level to properly track; the more slowly a signal drifts, the less noise power required to maintain synchronization. The proto-Stallion architecture and the adaptive synchronization technique that it hosts each produced impressive results; both warrant continued investigation to refine and enhance their capabilities.

Bibliography

[1] S. F. Swanchara, “Design and Implementation of an FPGA-Based Multi-user Receiver,” Master’s Thesis, Virginia Polytechnic Institute and State University, July 1998.

[2] P. Atiniramit, “Design and Implementation of an FPGA-based Adaptive filter Single-User Receiver,” Master’s Thesis, Virginia Polytechnic Institute and State University, September 1999.

[3] M. Hosemann, “Investigation of Synchronization Techniques for Direct-Sequence Spread-Spectrum Signals and Implementation on the Layered Software Radio Architecture Using Reconfigurable Hardware,” Master’s Thesis, Virginia Polytechnic Institute and State University, February 2000.

[4] “JBits,” Xilinx Online, http://www.xilinx.com/xilinxonline/jbits.htm, July 2000.

[5] S. M. Scalera, J. R. Vazquez, “The Design and Implementation of a Context Switching FPGA,” *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 1998.

[6] R. A. Bittner. “Wormhole Run-Time Reconfiguration: Conceptualization and VLSI Design of a High Performance Computing System,” Ph.D. Dissertation, Virginia Polytechnic Institute and State University, January 1997.

[7] P. Athanas, “Stallion: a wormhole run-time re-configurable computing machine,” Technical report, Virginia Tech, 1999.

[8] B. Schott, “SLAAC,” Presentation at the Loki Team Retreat, University of Southern California—Information Sciences Institute, February 2000.

[9] B. Schott, “SLAAC-1V VHDL Reference Guide,” University of Southern California—Information Sciences Institute, May 2000.

[10] B. Schott, “SLAAC-1V SDK Reference Guide,” University of Southern California—Information Sciences Institute, May 2000.

[11] C. Dick, F. Harris, M. Rice, "Synchronization in Software Radios—Carrier and Timing Recovery Using FPGAs," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2000.

[12] A. Alsolaim, J. Becker, M. Glesner, J. Starzyk, "Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communications Systems," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2000.

[13] A. J. Viterbi, *CDMA: Principles of Spread Spectrum Systems*, Addison-Wesley, New York, 1995.

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

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

[16] S. Haykin, *Adaptive Filter Theory*, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[17] K. Chapman, P. Hardy, A. Miller, and M. George, "CDMA Matched Filter Implementation in Virtex Devices," Xilinx Application Note 212 v1.0, March 2000.

[18] S. Srikanteswara, J.H. Reed, P. Athanas, and R. Boyle, "A Soft Radio Architecture for Reconfigurable Platforms," *IEEE Communications Magazine*, vol. 38, no. 2, pp. 140-147, February 2000.

[19] M. C. Horn, "Equipment Specification for the Miniature Radio Codec (MRC)", Rockwell Collins, 1999.

Vita

John Davies was born in Fairfax, Virginia in 1976. He entered Virginia Tech in 1994, graduating with a B.S.E.E. in May of 1998. While an undergraduate, he interned with Megadyne Corporation in 1995 and 1996, where he assisted the development of advanced switching power supplies. Also, in 1997 he interned in the telephone industry with Pulsecom, Inc. Following graduation, John continued at Virginia Tech, pursuing a master's degree. In May of 1998, he joined the Configurable Computing Laboratory, where he researched the implementation of wireless communications systems on FPGAs. John began employment with Integrity Broadband Networks (now Spike Broadband Networks) as a digital hardware engineer in August 2000.