Design and Implementation of a Soft Radio Architecture for Reconfigurable Platforms

Srikathyayani Srikanteswara

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

Doctor of Philosophy

in

Electrical and Computer Engineering

Dr. Jeffrey H. Reed, Chair

Dr. Peter M. Athanas

Dr. Brian D. Woerner

Dr. William H. Tranter

Dr. Richard E. Nance

July 2001

Blacksburg, Virginia

Keywords: Soft Radio, Software Radio, Software Radio Architecture, Reconfigurable Computing, 3G Systems, SDR

Design and Implementation of a Soft Radio Architecture for Reconfigurable Platforms

Srikathyayani Srikanteswara

(ABSTRACT)

Software radios have evolved as multimode, programmable digital radios that perform radio functions using digital signal processing algorithms. They have been designed as software programmable radios using a combination of various hardware elements and structures. In this dissertation a *soft radio* refers to a completely configurable radio that can be programmed through software, to change the radio behavior including the hardware functionality. Conventional software radios achieve flexibility through software with the use of static hardware. While these radios have the flexibility to operate in multiple modes, the hardware is not used efficiently. This inefficient utilization of hardware frequently limits the flexibility of software radios and the number of modes the radio can support. Soft radios however, attempt to gain flexibility through the use of reconfigurable hardware. The same piece of hardware can be configured to perform different functions based on the mode the radio is operating in.

While many soft/software radio architectures have been suggested and implemented, there remains a lack of a formal design methodology that can be used to design and implement reconfigurable soft radios. Most designs are based on ad hoc approaches which are appropriate only for the problem at hand.

After examining the design issues of a soft radio an architecture, called the *Layered Radio Architecture*, is developed with the use of stream based processing and run-time reconfigurable hardware. These choices aid in maximizing performance with minimum hardware while keeping the architecture robust, simple, and scalable. The reconfigurable platform enables *hardware paging* through reusability hardware. The stream-based ap-

proach gives a uniform modular structure to the processing modules and defines the protocol for interaction between various modules. The architecture describes a formal yet open design methodology and makes it possible to incorporate all of the features of a software radio while minimizing complexity issues. The layered architecture also defines the methodology for incorporating changes and updates into the system.

The layered radio architecture assumes run-time reconfigurability of the hardware. This feature is not supported by existing commercial reconfigurable hardware, like FPGAs. A Custom Computing Machine (CCM), called Stallion that supports fast run time reconfiguration, has been developed at Virginia Tech. This dissertation describes the deficiencies of existing commercial reconfigurable hardware and shows how the Stallion is capable of supporting the layered radio architecture.

The dissertation presents algorithms and procedures that can be used to implement the layered radio architecture using existing hardware. The architecture is validated with the implementation of two receivers: A single user CDMA receiver based on complex adaptive filtering and a W-CDMA downlink rake receiver with channel estimation. Performance analysis of these receivers show that it is important to keep the paging ratio high while maximizing utilization of the processing elements. The layered radio architecture with the use of Stallion can support existing high data rate systems.

Acknowledgments

I would like to thank my advisor, Dr. Jeffrey H. Reed for his support and encouragement throughout my research, without which this work would not have been possible. I was able to learn a lot through his guidance and would like to express my deep appreciation. I would like to thank Dr. Athanas who helped me understand the intricacies of hardware implantation and for being on my committee. I would like to express my gratitude to all my committee members for their valuable input throughout my research, as well as for their suggestions on improving my dissertation.

I would like to thank Jody Neel, Bruce Pucket, Nitin Mangalvedhe, Michael Hosemann, and Santiago Leon, who worked with me on the GloMo project and Dr. Bob Boyle for coordinating the research. In particular I would like to appreciate Jody's assistance with the Stallion simulator. I also benefited from conversations with Max Robert and other students of MPRG. I would also like to thank all the students and staff of MPRG for their cooperation and for making my stay pleasant.

I would like to acknowledge the financial support provided by the Defense Advanced Research Projects Administration's (DARPA) towards the GloMo project.

I would like to acknowledge all my friends for their support. Finally, I would like to express my deepest gratitude to my parents for their love, patience, and support throughout my research even though I have been away for many years with infrequent visits. Their motivating me to always do the best has been a constant source of encouragement and I dedicate this work to them.

List of Acronyms

ALU	Arithmetic and Logic Unit
CCM	Custom Computing Machines
CLB	Configurable Logic Blocks
COCOT	Completely COnfigurable Transmitter
COTS	Commercial Off The Shelf
CRC	Cyclic Redundancy Check
DARPA	Defense Advanced Research Projects Administration's
DPDCH	Dedicated Physical Data Channel
EDGE	Enhanced Data rates for GSM Evolution
EGC	Equal Gain Combining
EV	Embedded Variables
FPGA	Field Programmable Gate Array
FU	Functional Unit
GPRS	General Packet Radio Service
GUI	Graphical User Interface
ICNIA	Integrated Communications, Navigation, Identification, and Avionics system
IF	Intermediate Frequency
IMT-2000	International Mobile Telecommunications - 2000
ITU	International Telecommunications Union
JTRS	Joint Tactical Radio System
LMS	Least Mean Squares
LUT	Lookup Table
MAC	Multiply-ACcumulate
MSB	Most Significant Bit

NEV	Non-Embedded Variables
OSD	Office of the Secretary of Defense
OSI	Open Systems Interconnect
OVSF	Orthogonal Variable Spreading Factor
PMCS	Programmable Modular Communication System
QoS	Quality of Service
RKRL	Radio Knowledge Representation Language
RTR	Run-Time Reconfiguration
SDR	Software Defined Radio
SRAM	Static Random Access Memory
SRI layer	Soft Radio Interface layer
TAJPSP	Tactical Anti-Jam Programmable Signal Processor
W-CDMA	Wideband-Code Division Multiple Access

Contents

1 Introduction **1**

1.1 Introduction to Soft Radios 1

1.2 Motivation 3

1.3 Contributions 4

2 Background **7**

2.1 Software Radios in Existing and Future Wireless Systems 7

2.2 Soft Radio Architectures 11

2.3 The Evolution of Soft Radios 13

2.3.1 Other Software Radio Implementations 15

2.3.2 Limitations of Existing Soft Radio Architectures 17

2.4 Design Issues in a Soft Radio 18

2.5 Reconfigurable Computing 20

2.5.1 Operation of an SRAM based FPGA cell 21

2.5.2 FPGA Architectures 22

2.6 Applications of Reconfigurable Computing to Software Radios 27

2.7 Design Methodologies Using Reconfigurable Computing 28

2.7.1 Stream Based Processing 29

2.8 Conclusions 30

3 The Layered Radio Architecture 32

3.1 Use of Custom Computing Machines (CCM)in Soft Radios 32

3.1.1 Stallion Architecture 33

3.1.2 Software Tools for Design with Stallion 35

3.2 Layered Radio Architecture 38

3.2.1 Use of Stream Based Processing in the Layered Architecture . . . 40

3.3 Description of the Layered Radio Architecture 40

3.3.1 Soft Radio Interface Layer 42

3.3.2 Configuration Layer 44

3.3.3 Processing Layer 44

3.4 Advantages and Tradeoffs in the Layered Architecture 49

3.5 Example: Baseband Implementation of an IS-136 Receiver 51

3.6 Conclusions 52

4 Implementation of the Layered Radio Architecture 54

4.1 Implementation of the SRI Layer 57

4.1.1 Memory Organization 57

4.1.2 Configuring the Radio 58

4.2 Implementation of the Configuration Layer 60
4.2.1 Controller Packet Description 62
4.2.2 Configuration Module Packet Description 62
4.2.3 Data Module Packet Description 64
4.2.4 Configuration Layer Memory Organization 66
4.2.5 Configuration Memory 67
4.3 Implementation of the Processing Layer 68
4.4 Conclusions 69

5 Hardware Implementation of Receiver Architectures 70

5.1 Implementation of a Single-User Adaptive Receiver Using Stallion 71
5.1.1 Receiver Structure 72
5.2 Implementation of a W-CDMA Rake Receiver with Channel Estimation . 77
5.2.1 Performance of the Receiver 79
5.3 Hardware Requirements for Implementing the Layered Radio Architecture 86
5.3.1 Suggested Improvements for Stallion 88
5.4 Conclusions 90

6 Conclusions 91

6.1 Scope for Future Research 93

A Stallion Configurations for the Single User CDMA Receiver 103

B Stallion Configurations for the W-CDMA Receiver 110

List of Figures

2.1 Canonical software radio 11

2.2 Phase space for software radios 12

2.3 Programming bit for SRAM based FPGA 22

2.4 Three input lookup table 23

2.5 Xilinx 4000 series routing structure 26

2.6 Xilinx 4000 series FPGA structure 26

2.7 Design process for reconfigurable computing 29

2.8 Stream-based processing 30

3.1 Stallion organization 35

3.2 Structure of a Functional Unit (FU) 36

3.3 The Stallion chip 37

3.4 Layered radio architecture 39

3.5 Structure of a stream packet 40

3.6 Soft Radio Interface Layer: Forward direction 43

3.7 Soft Radio Interface Layer: Reverse direction 44

3.8 Processing elements in the processing layer 45

3.9 Structure of a processing element 47

3.10 Structure of a processing element 48

3.11 IS136 transmitter 51

3.12 Processing layer modules (baseband) for IS136 receiver 52

4.1 Implementation of a variable length FIR filter on Stallion 56

4.2 Soft radio interface layer state table 58

4.3 SRI layer memory organization 59

4.4 Structure of the configuration layer 60

4.5 Controller packet 63

4.6 Configuration module packet structure 64

4.7 Data module packet structure 65

4.8 Data memory of the configuration layer 66

4.9 Configuration memory of the configuration layer 67

5.1 Block diagram of the single user adaptive receiver 73

5.2 Acquisition algorithm for the receiver 74

5.3 GloMo soft radio 75

5.4 Downlink rake receiver with channel estimation 80

5.5 Percentage utilization of Stallion for different spreading factors 82

5.6 Power consumption for the downlink rake receiver 84

5.7 Tradeoffs in Stallion for W-CDMA rake receiver implementation 87

5.8 Possible approach for adding memory to Stallion 89

1.1 Acquisition 104

1.2 FIR filtering 105

1.3 Complex demodulation 106

1.4 Alter μ 107

1.5 Error computation 108

1.6 Computation of new weights 109

2.1 Maximal ratio combining 111

2.2 Matched filtering 112

2.3 Generation of the spreading code 113

2.4 Channel estimation 114

2.5 Generation of the pilot sequence 115

Chapter 1

Introduction

1.1 Introduction to Soft Radios

Software radios are evolving as flexible all-purpose radios that can implement new and different standards or protocols through reprogramming [35], [6], [38]. Software radios can reduce the cost of manufacturing and testing, while providing a quick and easy way to upgrade the product to take advantage of newer signal processing techniques and new wireless phone applications. The ideal software radio has a set of features that are not yet realizable in commercial systems due to limitations in current technology and cost considerations. Specifically, DSP microprocessors are not fast enough to implement all radio functions and thus some dedicated Application Specific Integrated Circuits (ASIC) are required. This is a consequence of processing speed being traded for flexibility and efficiency. All research efforts are thus geared towards realizing the ideal software radio. The main features of a software radio are:

• Placement of the A/D converter: Ideally the A/D converter should be placed adjacent to the antenna in the transmitter and receiver, so that almost all the functions are performed digitally for maximum flexibility. However, due to practical limitations in

cost and speed of digital signal processing, most current software radio architectures digitize the signal at the Intermediate Frequency (IF) stage.

• Highest level of reconfigurability: It should be possible to modify the entire configuration of the radio entirely in software. The system should accommodate various protocols and formats. For example, it should be possible to configure the radio to any modulation format, frequency, coding formats, network protocols, and standard. Further, the radio should also have the flexibility to incorporate changes or additional functionality as the need arises.

• Run time reconfigurability: It should be possible to change some of the parameters of the system during run-time without powering the system down. This allows for a change in the standard being supported during transmission. This capability also allows for a sub-system to be "paged" into the hardware as needed.

In this report the term *soft radio* refers to a completely configurable radio that can be programmed in software to reconfigure physical hardware. In other words, the same piece of hardware can be modified to perform different functions at different times. This allows the hardware to be specifically tailored to the application at hand, resulting in greatly increased speed and silicon efficiency, while maintaining a high degree of flexibility.

Soft radios give rise to the possibility of having multi-mode terminals without the "Velcro approach" of including separate silicon for each possible standard, which considerably increases the size and cost of the radio. The number of discrete components are reduced in a soft radio, since many of the traditional radio functions like synchronization, modulation, and coding can be integrated into one chip.

Soft radios also reduce the cost associated with manufacturing and testing the radios. It is possible to precisely predict the performance of digital hardware, unlike analog hardware, which frequently show a drift over time in characteristics.

Once the soft radio is fabricated, the design cycle for upgrading the radio is reduced,

since most of the existing hardware can be used. Furthermore, a soft radio is scalable, that is, additional hardware modules can be easily added to expand the capabilities of the radio, similar to adding memory to a PC. Total software reconfigurability also makes it possible to transmit upgrades to the mobile receiver over the air with minimal hardware compatibility issues. Soft radios also have the capability to perform self-diagnosis, thus reducing the need for human intervention, thus increasing reliability.

A soft radio might autonomously select the best transmission mode, send probing signals to establish a link, explore communications protocols with the remote end and adapt to the remote signal format [34]. It could also select the mode for lowest cost, service availability or best signal quality.

1.2 Motivation

Research in the area of soft/software radios has led to the development of many theoretical architectures as well as specific soft radio implementations. While most researchers agree on the usefulness of the soft/software radio and most of its desirable features [36] [35] [32] [61], there remains a lack of a general design methodology that can be applied to reconfigurable systems. Most designs are based on ad hoc approaches which are appropriate only for the problem at hand. The theoretical architectures lack a formal design methodology needed for the systematic realization of soft radios.

Third generation (3G) systems are now starting to be deployed. However, 1G and 2G standards will continue to be widely used for some time to come. To achieve "communications anywhere, anytime," it is imperative to support these varied standards, necessitating the use of software radios. Further, even with global convergence of standards, 3G and future systems will have to support services beyond voice. There will eventually be systems where the user can select the Quality of Service (QoS) for each call that have different physical layer characteristics based on the specific services the user

selects. Such systems could use various adaptive algorithms like adaptive modulation schemes which can be implemented efficiently with the use of software radios. However, for these systems to be commercialized and deployed on a large scale, a systematic design methodology for software radios is needed. Standardization of the methodology would help vendors to design products that can be used by independent software radio manufacturers. However, the design methodology should be open and allow room for development of new algorithms and should not limit the scope of the communication or signal processing algorithms currently used in wireless systems.

Research has been continuing at Virginia Polytechnic Institute and State University in developing algorithms and architectures for soft radios under the sponsorship of Defense Advanced Research Projects Administration's (DARPA) GloMo I & II projects. The research efforts of GloMo I culminated in the implementation of a multiuser receiver based on reconfigurable computing [2]. As a follow-on to GloMo I, a generic soft radio architecture was developed that uses reconfigurable hardware and stream-based processing [48].

1.3 Contributions

A survey of existing soft radio architectures was performed to identify the areas that needed further research. The survey identified the following key areas: first, the latest innovations in reconfigurable computing are not being exploited in soft radio architectures. Second, the existing architectures are based on ad-hoc designs and lack a systematic design procedure which stifle their widespread use across platforms and standards. Third, the existing soft radio architectures support only a subset of the existing systems and features and are not truly soft. In other words, if a radio is designed to handle IS-95 and GSM, the same radio cannot handle third generation systems without modifications to the hardware. Also, features like over-the-air updates and software validation are

commonly ignored in current software radio architectures. A detailed study of the design issues led to the development of an architecture that overcomes the limitations of existing software radio systems.

In this report a formal design philosophy and methodology is developed that can be used to design any soft radio. The architecture ties in the latest developments in custom computing machines with software radio architectures for designing radios that can support all the desirable software radio features. A soft radio architecture is developed that can handle complex data processing with efficient resource allocation, while maintaining hardware reusability, flexibility and scalability. Stream-based processing was found most suitable for soft radio architectures to keep them simple and scalable in hardware.

The concept of *layered radio architecture* is introduced, that enables maximum flexibility in radio capability. The architecture describes a formal yet open design methodology. Many of the ideas behind the layered architecture were drawn from the Open Systems Interconnect (OSI) layered network model. The architecture assumes stream-based computing applied to a run-time reconfigurable hardware platform. The reconfigurable platform enables reusability of hardware during run-time thereby optimizing the use of silicon in a multimode radio. The stream-based approach gives a uniform modular structure to the processing modules and defines the protocol for interaction between various modules. The layered architecture provides a formalized structure for the implementing communications signal processing functions in reconfigurable hardware modules. The layered architecture also describes inter-module interface and communication that is not dependent on a particular standard or system. The design of the architecture also includes a detailed description of the actual procedure needed to configure the radio to different systems, load in a new configuration and other programming issues.

This architecture is capable of supporting a soft radio and all its desirable features. Current research at Virginia Tech focuses on the development and testing of a soft radio based on the layered architecture and building a library of soft radio modules. The

report also shows that the architecture is capable of supporting some of the widely used standards like IS-136, IS-95 and other 3G systems. A single user CDMA receiver using complex adaptive filtering was implemented to validate the layered radio architecture on worm-hole reconfigurable hardware platform.

A theoretical analysis of the architecture is used to demonstrate that the architecture is capable of supporting all the desired software radio features. In addition, it is shown that the architecture can support existing and future high data rate systems. The implementation of a downlink rake receiver with channel estimation for W-CDMA shows that the architecture with the use of a CCM called Stallion can handle complex high data rate signal processing while retaining flexibility. Speed/flexibility tradeoff has been well accepted in software radios. The layered radio architecture with the use of CCMs can support existing 3G data rates as well as higher data rates while retaining complete flexibility and using acceptable levels of power consumption.

Finally the report outlines some of the requirements on the development of the reconfigurable hardware. The design methodology and concepts used in the architecture can be easily extended for building reconfigurable nodes in the higher network layers.

Chapter 2

Background

2.1 Software Radios in Existing and Future Wireless Systems

With the emergence of new technologies and services, the demands placed on the mobile handset have continually increased. There is an increasing need for integrated services that offer seamless global coverage, future proof radios and user controlled Quality of Service (QoS). Software radios are now poised to handle these different issues in an efficient manner.

There has been an emphasis on providing integrated services for voice, data and video through communication across heterogeneous devices to provide seamless service to the user. The concept of integrated seamless global coverage requires that the radio support two distinct features. First, it should offer global roaming or seamless coverage across geographical regions. Second, it should interface with different systems and standards to provide seamless services at a fixed location. Multimode phones that can switch between different cellular standards like IS-95 and GSM fall in the first category while the ability to interface with other services like Bluetooth, or IEEE 802 networks fall in the second

category.

A brief look at the development of different wireless standards throughout the world indicates the presence of diverse systems even with a strong effort to unify global standards, as well as a shift towards higher data rates as mobile handsets provide increased services. Studies on globalization of personal communications began in 1986 by the International Telecommunications Union (ITU), to identify the long term spectrum needs for the future 3G mobile telecommunications systems [16]. In 1992, ITU identified 230 MHz of spectrum in the 2GHz band to implement the International Mobile Telecommunications - 2000 (IMT-2000) system for worldwide basis for satellite and terrestrial components. The aim of IMT-2000 is to provide universal coverage enabling terminals to have seamless roaming across multiple networks. Standard bodies in various countries have since been trying to achieve harmonization on issues like radio interfaces, system evolution, backward compatibility, user migration, global roaming, and introduction of mobile services and capabilities to support terminals.

2G standards enabled data capabilities from 9.6kb/s to 14.4kb/s [29]. As the demand for capabilities requiring higher data rates percolated in the mid 90's, 2.5G standards began to evolve with higher data rates enabled by multi-slot data. GSM was enhanced through the introduction of Enhanced Data rates for GSM Evolution (EDGE), and General Packet Radio Service (GPRS) that allows data rates up to 170kbits/sec [16]. EDGE allows GSM operators to use the existing GSM bands to offer wireless multimedia Internet Protocol (IP) based services and applications. EDGE provides bit rates of 384 kbps for pedestrian and low speed vehicular movement, 144 kbps for high speed vehicular movement, 2 Mbps using indoor wideband. Among the third generation systems, CDMA2000 uses a chip rate of 3.6864 Mchips/s for 5 MHz bandwidth with a direct spread downlink, and a 1.2288 Mchips/s chip rate for a multicarrier downlink, where the multicarrier approach partitions the downlink spectrum into multiple 1.25 MHz carriers. CDMA2000 provides data rates from 144kbps to over 2Mbps in the forward link. W-CDMA specifies a constant chip rate of 3.84Mchips/s with data rates from 8kbps to

2Mbps.

As 3G systems are launched, they will have to co-exist with the second and even the first generation wireless systems [54], [32], [11]. The 2G and 3G systems differ in their modulation formats, channel coding, multiple access, etc., which leads to the need for multiband, multimode terminals and base stations. 3G terminals offer a wide range of challenges since they have to deliver additional services beyond voice. This results in increased processing requirements and a more complex user interface [29]. Ko et al. estimate the 3G DSP MIPS requirements to be an order of magnitude higher than that of 2G's, as a result of the higher data rates and additional functionality of 3G systems [29]. Thus 3G systems are likely to dissipate an order of magnitude more power in the DSP assuming the same silicon process technology. Hence, aggressive low power design strategies, optimization of process technology and design techniques must all be used ensure same or better 3G handset solutions. Kourtis et. al. [28] state that while the existing silicon technology is adequate for the implementation of 3G standards, there must be improvements in the performance and power consumption of A/D converters and DSPs to efficiently implement future systems. People are thus increasingly turning towards software radios to provide solutions to 3G systems.

Existing technologies for voice, video and data use different packet structures, data types as well as different signal processing techniques. Integrated services can be obtained either with a single device capable of delivering various services or with a radio that can communicate with devices providing complementary services. The supporting technologies and networks that the radio might have to use can vary with the physical location of the user. To successfully communicate with different systems, the radio has to communicate and decode the signals of devices using different air interfaces. Such radios can be implemented efficiently using soft radio architectures where the radio reconfigures itself based on the system it has to interface with and the functionalities that it has to support.

Even if there is a global convergence of cellular standards, 3G systems require multimode operation and automatic mode selection. It is likely with 4G and possibly 3G systems, the user's application will have the ability to control the quality of service and obtain a higher QoS for a more expensive call [53]. Higher QoS can be achieved through the use of better error protection coding, minimizing the amount of traffic in that band, better channel equalization techniques and so on. The mobile subscriber unit should also have the ability to select the network provider as well the services it needs. Software radios are needed to efficiently implement such systems [42]. A software based approach that can accommodate 2G and 3G standards is suggested in [54], while an FPGA based solution is proposed in [11].

Accompanying the rapid development of new and improved technologies and changes within the standards itself, is the widespread recognition of the need to design "future proof" radios that do not become obsolete soon after the handsets are delivered to the customers. Two possible situations that could make an existing software radio obsolete. First, a change in one of the standards that the multimode radio has been programmed for, causing a discontinuity in the geographical coverage area, or development of improved algorithms that make existing systems perform better. In both these situations, the user should be able to upgrade the radio through over-the-air updates with software validation and simple hardware upgrades that can be performed by the user if necessary.

All these service requirements indicate the need for compact, power efficient, fully programmable radios that perform multiple diverse functions that are scalable in hardware as well as software. Satisfying these diverse needs makes research in the development of software radios extremely important.

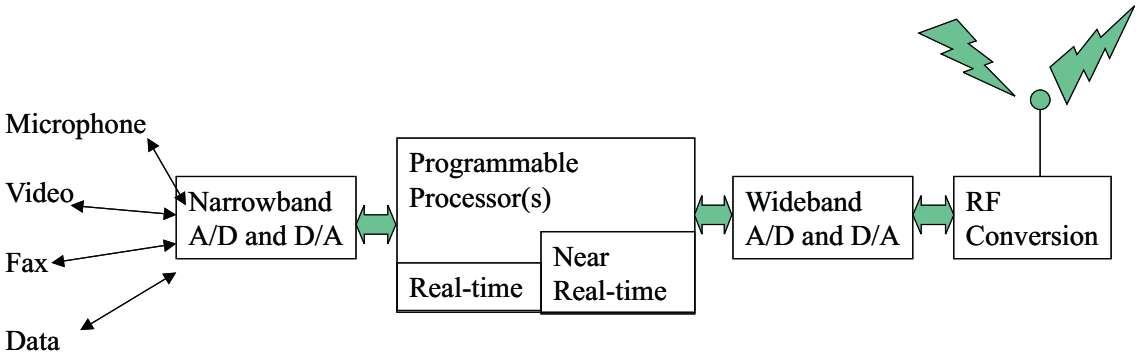


Figure 2.1: Canonical software radio

2.2 Soft Radio Architectures

One of the first descriptions of a software radio was based on a canonical architecture by Mitola [35]. The canonical software radio consisted of a power supply, an antenna, a multiband RF converter, and a single chip containing A/D and D/A converters with on-chip general purpose processor and memory to perform the radio functions and required interfaces between them as shown in Figure 2.1.

Mitola describes software radios in terms of a *phase space* [38]. The phase space is a graph with the type of hardware on the X-axis and the frequency of digitization on the Y-axis. Each radio is given a place on the graph based on the flexibility of the hardware and the frequency at which the signal is digitized. Existing digital radios can be placed in various locations on the phase space based on their degree of flexibility and re-programmability. The ideal soft radio needs to have complete reprogrammability in hardware and software, as well as the highest digitization frequency as shown in Figure 2.2.

Mitola [37] also uses a *vector space* definition for a software radio. He describes four key technical and economic criteria that define to what degree a radio is a software radio. The criteria are: the number of air interface channels simultaneously supported (N),

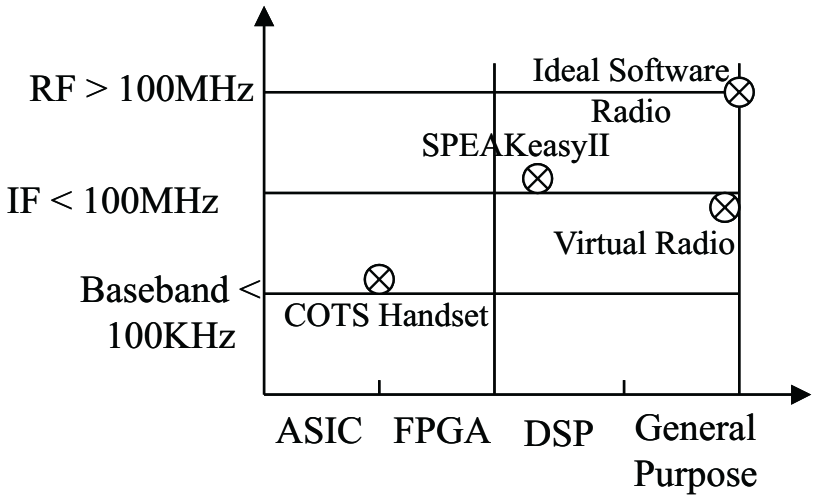


Figure 2.2: Phase space for software radios

programmable digital access (PDA), hardware modularity (HM), and software flexibility and affordability (SFA). The level of programmable digital access is the stage at which the A/D converter is placed and the radio is functionally programmable in the software radio phase. The hardware modularity indicator can be described by coarse grain modules such as receivers and exciters and finer grain modules such as FPGA, A/D converters and DSP chips. Software modularity characterizes the service provider's ability to buy plug and play software modules from multiple independent vendors. Software that runs on many platforms and that is widely available is desirable.

The above four features can be thought of as different dimensions in a feature space. Each radio can be assigned a value (between 0 and 3) along each coordinate axis, where (0,0,0,0) is a completely fixed radio and (3,3,3,3) is the ultimate soft radio.

There has also been some effort to give a mathematical perspective to the functioning of software radios [38]. Greater understanding of the mathematical properties of the software radio architecture could accelerate progress toward cost-effective plug and play services. Mitola describes three architectural features:

• Bounded recursive modules: All the software radio modules should be built using bounded recursive modules. That is, these modules are defined for all inputs and consume predictable system resources. Although the software may contain errors, those errors will be much less likely to cause system crashes than unconstrained modules. Thus, some specific system behavior will be defined at all times.

• Explicit extensible interface topologies: Extensible languages such as UML, SDL and ASN.1 should be used until a more domain specific language such as a radio knowledge representation language (RKRL) emerges. These languages should be capable of defining the radio itself as well as its external interfaces. [1]

• Distributed layered virtual machine: The plug and play modules should be positioned in the distributed layered virtual machine hierarchy in a way that maximizes the use of existing (lower and higher level) components. Modules partitioned according to the distributed virtual machine reference model will insulate lower layer hardware-dependent modules from upper layer service defining modules.

Studies in software radio issues in cellular base stations have led to proposals of complete software solution [56], similar to MIT's SpectrumWare project [5], as well as specific hardware implementations [63]. Gunn et. al. [20] describe a hardware architecture based on a DSP core that can accommodate homodyne as well as heterodyne receivers.

2.3 The Evolution of Soft Radios

The earliest software radios were initially multimode radios and flexibility was present only in software. In the late 1970's, the U.S. Air Force began work on a system known as the Integrated Communications, Navigation, Identification, and Avionics system

[1] RKRL is used to capture knowledge of radio etiquette, devices, propagation networks, user needs, and application scenarios enables software radios to learn about the personal use patterns of the owner and the context of communications events.

(ICNIA)[10]. This was one of the first systems to use DSP-based programmable modem and control to obtain fully integrated capability for airborne platforms. The heart of ICNIA was a 25MHz 6800-series low latency DSP used as a processing element, a general purpose processing element, and an enhanced security unit, which interfaced with an external federated Communications Security (COMSEC) device (KOV-5). ICNIA's technology laid the foundation for other systems, and the Advanced Tactical Fighter's (F-22) Communications, Navigation, and Identification (CNI) system that evolved over the next two decades. Building on the ICNIA technology, in 1989, the Air Force initiated the Tactical Anti-Jam Programmable Signal Processor (TAJPSP) with the objective of developing a modular, reprogrammable modem.

The TAJSP effort developed into a tri-service programmable radio program called SPEAKeasy [10]. SPEAKeasy used the ICNIA technology, but quickly migrated to a robust tactical security architecture and an improved DSP base. The primary objective of the SPEAKeasy was to develop a modular, reprogrammable, modem with an open architecture and demonstrate this software-defined radio core using a flexible RF front-end. [31]. A secondary objective was to develop a generic software architecture to facilitate the addition of new waveforms. SPEAKeasy phase 2 was initiated in 1995 to expand beyond the modem to encompass an open, modular, reprogrammable architecture for the entire radio system and focus on many of the link layer and communications layer details. The system was developed and tested using the FLIP-WAVE spread spectrum modem invented at the U.S. Air Force Research Laboratory [44]. These designs then led to various other commercial architectures for radios and spawned research in the area of soft radios.

The Office of the Secretary of Defense (OSD) initiated the Programmable Modular Communication System (PMCS) effort in 1997 as a continuation of the SPEAKeasy effort. The PMCS guidance document encapsulated the efforts of the program, providing a notional software radio functional architecture. This notional architecture served as the precursor to the Joint Tactical Radio System (JTRS) architecture as well as the

baseline for the architecture promulgated by the Software Defined Radio (SDR) Forum.

The recent architecture proposed by the JTRS consists of six layers. The bottom layer is the hardware platform, which includes antennas, RF, IF and software programmable components such as ASICs, FPGAs, DSPs and general purpose processors. The next layer, the infrastructure layer, provides common real time distributed processing services which can include setting up and tearing down high performance pathways in heterogenous signal processing hardware for the flow of isochronous signals such as digitized speech. The radio aspects layer consists of software objects that employ infrastructure and platform capabilities to realize the programmable capabilities of a software defined radio on the physical and link layers of the ISO protocol stack. The next layer, the networking aspect layer includes link management, host adaptation, and conventional network-layer to presentation layer protocol stacks. The communications applications layer is responsible for interfacing with diverse networks while the top layer, the communications applications layer is responsible for providing the user interface. While the JTRS recommends a layered architecture, the actual implementation of these layers is not defined nor are the hardware components that can support all the required services.

2.3.1 Other Software Radio Implementations

Research in the area of software radios has been rapidly gaining speed and a number of industries have developed their own versions of soft radios and the hardware components for implementing them. The Software Defined Radio (SDR) Forum is also actively involved in examining and defining software radio architectures.

The software architecture and design issues for tactical radios have been examined [19]. A tactical software radio is a computer-based radio with analog RF hardware specific to radio communications. Once the signal is digitized the radio is defined entirely in software. Configurable computing options have also been investigated to develop tactical radios that provide individualized communication security capabilities without re-design

[27].

The Defense Advanced Research Projects Agency (DARPA) is funding various projects to study the development of various reconfigurable platforms that have runtime reconfigurability and are cost effective. The Adaptive Computing Systems project funded by DARPA is aimed at developing the Commercial Off The Shelf (COTS) hardware components, design, programming, and runtime environments to enable an application to reach through to the hardware layer and directly manipulate the datapath-level architecture at runtime to optimize the application-level performance. This program explores new techniques that will result in rapid realization of algorithm specific hardware architectures on a low-cost COTS technology base [13]. The Pleiades, a project funded by DARPA under the Adaptive Computing Systems project at UC Berkeley seeks to achieve ultra-low power high-performance multimedia computing through the reconfiguration of heterogeneous system modules. Pleiades attempts to achieve high power efficiency by providing programmability at the right granularity and exploiting other energy reducing techniques, such as parallelism, pipelining and dynamic voltage scaling [45]. MIT's SpectrumWare project implements the entire radio in software while relying on a standard workstation's hardware platform [5].

The research in the area of soft radios has led to the development of a number of commercial products by various companies like Chameleon Systems Inc., Morphics, and Quicksilver. A reconfigurable processor platform architecture that reduces the tradeoff between performance and flexibility needed in wireless base stations is being designed by Chameleon Systems Inc. [8]. On the other hand, Morphic's Dynamically Reconfigurable Logic (DRL) is supposed to enable replacement of the multiple hardwired logic cores in today's wireless baseband processors with a single-programmable, hardware reconfigurable core [40]. The Sirius CDMAx test chip can be software reconfigured to support mobile station and base station configurations. It is also reconfigurable as a GPS navigation baseband receiver and as a Satellite UMTS transceiver core. The CDMAx supports multiple standards including 3G UMTS/FDD standard (Europe), 3G ARIB standard

(Japan) and Satellite UMTS [47].

2.3.2 Limitations of Existing Soft Radio Architectures

Software radios were initially multimode radios whose operation could be controlled through software. Such radios assumed a static hardware and attained flexibility only through software. Extending the flexibility to hardware through the use of reconfigurable computing can optimize a soft radio architecture and its implementation to make the radio truly "soft". The reconfigurable hardware and changeable software constitute the two vital parts of a soft radio. Reconfigurability in hardware is especially challenging to attain in soft radios. As described in the preceding sections, researchers have examined various theoretical architectures as well as specific soft radio implementations. General design principles for building soft radios have been suggested based on the challenges facing today's designers [32], [39]. Simultaneously, there has been substantial progress in the development of new reconfigurable hardware technologies that incorporate flexibility at various levels. However, these innovations have not been integrated into a common soft radio architecture and design philosophy for reconfigurable platforms.

While most researchers agree on the usefulness of the soft/software radio and most of its desirable features [36] [35] [32] [61], there is a lack of general design methodology applicable to reconfigurable systems. Most designs are based on ad hoc approaches which are only appropriate for the problem at hand. There is a lack of a formal design methodology needed for the systematic realization of soft radios. Though much work is being done in the software and network layers of the soft radio, there has been little progress in the development of a generic hardware architecture for a soft radio.

This dissertation addresses these issues in an attempt to develop a generic architecture based on reconfigurable computing. A formal design philosophy and methodology is developed that lays the foundation for soft radio architectures. An architecture is developed that can handle complex data processing with efficient resource allocation, while

maintaining hardware reusability, flexibility, and scalability.

2.4 Design Issues in a Soft Radio

Developing a soft radio architecture requires a thorough understanding of communication algorithms as well as hardware design principles. It is important to identify the design issues in order to develop an efficient architecture. The properties and features of soft radios bring forth design issues that are different from those of other digital radios. A number of papers describe various software radio architectures and technical challenges in great detail [1], [35], [6], [38]. The key design issues include the design of fast and efficient A/D converters, flexibility at the RF front end, effective data management procedures, resource allocation and smooth reconfigurability of the hardware.

A/D conversion and flexible RF front end are extremely challenging to design, particularly for a handset, and set the limitations of dynamic range and bandwidth. The design issues involving the A/D converter and the RF design are beyond the scope of this work but are covered in [46], [55], [7]. A/D converter performance set the limit on how much of the IF signal processing can be moved into the digital domain [28]. Great advances in power dissipation and resolution are needed to make commercial software radio terminals possible, where channel selectivity is achieved completely in the digital domain. Performance increases in A/D converters lag the performance increases in digital components, since analog components need redesign with each reduction in process geometry, due to their sensitivity to noise. Digital components migrate simply from one lithography shrink to the next but analog circuits need a process where the parasitics are minimized and precision capacitors are available. Hence, a careful process scaling is needed rather than an optical shrink [28].

A soft radio is a complex entity that needs to handle very high data rates efficiently. Processing data efficiently entails good computing resource allocation. There are some

very important differences in the resource allocation of a system implemented with dedicated hardware and the same system implemented as part of a soft radio. While both approaches strive for compactness and power efficiency, a soft radio design also requires flexibility and reusability, which impacts the partitioning of a process into its hardware components. This resource allocation is closely related to the data flow properties of the system since data has to be processed with minimal delays and overhead.

A soft radio needs to be scalable at the system level, the software level, and the hardware element level. Hardware scalability supports easy addition of computing hardware, ideally as easily as adding additional memory to a conventional processor. Software scalability refers to modifying/adding features to the radio through software upgrades. At the system level, the radio has to be conducive to *replication*, i.e., it should be easy to add channels to support more users.

Soft radios require complex signal processing at very high speeds which place high demands on the hardware platform. The traditional choices for implementation of soft radios have been ASICs, FPGAs, and DSPs. Current single chip Digital Signal Processors (DSP) are incapable of supporting many of the computationally demanding algorithms such as multiuser interference cancelation and large and sophisticated adaptive arrays. While DSPs provide maximum flexibility and a short design cycle, they are not efficient in terms of power consumption and silicon area. Multi-DSP computing platforms are quite common, but achieving inter-processor communication is often complicated, which reduces the scalability of the system when dealing with already limited I/O bandwidth and high sample rates associated with wideband signals. Use of ASICs, on the other hand, result in the most efficient implementation of a given circuit. However, they have little flexibility, high initial cost, and a long design cycle.

Another important design issue is the ability to have smooth reconfiguration of the radio, which include supporting run-time reconfiguration, over-the-air updates with software validation. Smooth reconfiguration is necessary to minimize delays and make the system

robust and easy to handle. The reconfigurable nature of a soft radio gives rise to some secondary design issues like reusability of hardware, scalability, and processing power. For mobile communication systems, power consumption is an important issue. Reusability of hardware when supporting multiple communication standards is what makes a soft radio efficient in terms of silicon utilization when compared to a "velcro radio" that uses hardware optimized for each system. In a "velcro radio," while each individual system might demonstrate better silicon efficiency, as the number of systems increase, a soft radio implementation will have better silicon efficiency since additional systems can be implemented without the addition of new hardware.

The choice of the hardware platform and the different hardware components used, is crucial to the design of a soft radio. The subsequent sections give an overview of some of the existing reconfigurable technologies and their relevance to the design of soft radios. The traditional hardware options available for implementing reconfigurable radios, their advantages and deficiencies will be examined next.

2.5 Reconfigurable Computing

The use of reconfigurable hardware is central to the design of soft radios. The Field Programmable Gate Array (FPGA) was introduced in the mid-1980's as a device to rapidly create digital logic, and this marked one of the major developments in reconfigurable computing. FPGAs were designed for multilevel circuits, which means they could handle complex circuits on a single chip. Since FPGAs were prefabricated, they were quicker to use and also cost less. However, customers considered the Static Random Access Memory, (SRAM) arrays in the FPGAs too volatile [21]. Antifuse FPGAs were then developed to get around the *drawback* of SRAMs. Antifuses are one time programmable devices which when *blown* create a connection. When they are *unblown*, no current can flow between their terminals. Thus it is an antifuse since its operation

is opposite to that of a standard fuse. In the late 1980's and early 1990's, the industry began to realize that rather than being a drawback, the volatility of SRAM is what makes the FPGA attractive for digital systems. Since they are re-programmable, their configurations can be easily changed to upgrade systems or correct system bugs, making them ideal for prototyping. FPGAs are now used in different configurations in multi-mode and reconfigurable systems, and are very useful in meeting the needs of a soft radio system.

SRAM cells are larger than antifuses and EEPROM/EPROM, meaning that SRAM-based FPGAs will have fewer configuration points than FPGAs using other programming technologies. However, because of their flexibility, SRAM based FPGAs are now widely used. SRAM based FPGAs are configured in the target system by means of a bit stream that completely specifies the logical functions and connectivity to be implemented. The configuration program is either loaded automatically from external memory on power up, or is downloaded by a processor during system initialization. SRAM based FPGAs can be re-programmed an unlimited number of times, and possibly *on the fly*, during the operation of the system.

The flexibility of FPGAs comes at the cost of efficiency and power consumption relative to ASICs. FPGAs cannot achieve the power, clock rate, or die size that could be realized in a full custom chip optimized for a particular task though they do work out cheaper for lower production runs when compared to an ASIC [58].

2.5.1 Operation of an SRAM based FPGA cell

In SRAM based FPGAs, memory cells are scattered throughout the FPGA. As shown in Figure 2.3, a pair of cross-coupled inverters sustain whatever value is programmed into them. A single n-transistor gate is provided for either writing a value or reading a value. The ratio of sizes between the transistor and the upper inverter is set to allow values sent through the n-transistor to overpower the inverter. The read back feature is

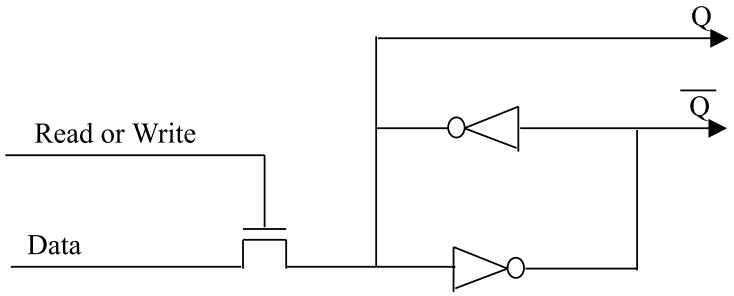


Figure 2.3: Programming bit for SRAM based FPGA

used during debugging to determine the current state of the system. The actual control of the FPGA is handled by the Q and Q^- outputs. One simple application of an SRAM bit is to have the Q terminal connected to the gate of an n-transistor. If a 1 is assigned to the programming bit, the transistor is closed, and the values can pass between the source and the drain. If a 0 is assigned, the transistor is opened and the values cannot pass. Thus, this construct operates similar to an antifuse, though it requires much more area. One of the most useful SRAM based structures is the Lookup Table (LUT). By connecting 2^N programming bits (P1... P7), to a multiplexer as shown in Figure 2.4, any N-input combinational boolean function can be implemented. Although it can require a large number of programming bits for large N, LUT's of up to five inputs can provide a flexible, powerful function implementation medium.

2.5.2 FPGA Architectures

Although there is a wide range of architectural approaches that are used in FPGAs, all FPGAs consist of an array of logical units distributed across a grid of programmable interconnect, or routing. For a given total die size, the basic design issues are much the same as those involved in the design of any other computing device. These issues include the amount of resources to devote to routing as opposed to computation; the amount,

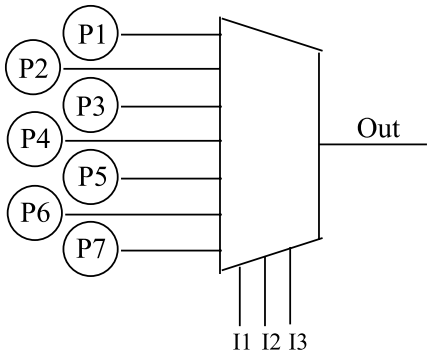


Figure 2.4: Three input lookup table

granularity, and distribution of on-chip memory; and the quantity and accessibility of I/O [58].

FPGAs have been classified based on their architecture [43], amount of reconfigurable logic, and the availability of dedicated local memory [18]. FPGAs can also be broadly classified as either *coarse grained*, meaning that they possess a smaller number of relatively powerful logical units, or *fine grained*, which refers to a larger number of very elementary logic units. Most FPGAs are coarse grained and have logical units based on LUTs [58]. For example, the Xilinx 4000 series family of FPGAs has logical units (referred to by Xilinx as configurable logic blocks (CLBs) that each contain two 4-input, 1-output LUTs, one 3-inout, 1-output LUT, two filp flops, and several multiplexors to select from among the LUT and flip flop inputs and outputs.

FPGAs contain a matrix of logic elements (cells) that can be interconnected in any desired configuration to implement a given algorithm. There are various configuration in which FPGA cells can be arranged. This section will describe two popular architectures used: the Xilinx 4000 Logic Cell Array and the cellular array style CLi FPGA. In the Xilinx 4000 FPGA, the logic cells are embedded grid like on the chip. This arrangement is ideal for long distance communication as long as most connections between input and output are relatively short. The cellular array is configured specifically for fast

local communications. This configuration features a small amount of routing resources running horizontally and vertically between the cells, limiting communication to short distances on the chip.

The difference between the Xilinx and cellular style lies in the routing structure. The Xilinx structure is designed for complex, irregular random logic and has fewer but denser cells containing large amounts of programmable gates. It features a powerful routing structure optimized for arbitrary global routing and large logic cells capable of providing arbitrary four and five input functions. This provides a very flexible architecture though one that requires many programming bits per cell, which means that the cells take up a large portion of the chip area.

The cellular style employs simpler logic cells but in greater numbers per chip. The architecture is optimized for highly local, pipelined circuits such as systolic arrays and bit serial arithmetic. It emphasizes local communication at the expense of global routing. Due to the presence of simple logic cells, there will be many more CLi cells on a chip than will be found in the Xilinx FPGA, yielding a greater logic capacity for those circuits that match the FPGA's structure. Due to the restricted routing in CLi FPGA, it is much harder to map than the Xilinx 4000 series FPGAs, though the simplicity of the CLi architecture makes it easier for the human designer to hand-map to the CLi's structure. Thus in general cellular architectures tend to appeal to designers with appropriate circuit structures who are willing to spend the effort to hand-map their circuits to the FPGA, while the Xilinx 4000 series is more appropriate for handling random logic tasks and automatically mapped circuits. The logic capacity of the two systems is thus equivalent but of different kinds.

Xilinx 4000 Logic Cell Array

The Xilinx FPGA is an *IslandStyle* FPGA with logic cells embedded in a general routing structure that permits arbitrary point to point communication. The logic blocks are

surrounded by horizontal and vertical routing channels as shown in Figure 2.5. The only requirement for good routing structure is that the source and destination be relatively close together. Details of the routing structure are shown in Figure 2.6. Each of the inputs of the cell (F1-F4, G1-G4, C1-C4, K) comes from one of a set of tracks adjacent to that cell. The outputs are similar (X, XQ, Y, YQ), except that they have the choice of both horizontal and vertical tracks. The routing structure is made up of three different length lines: single length, double length, and long lines. Single length lines travel the height of a single cell, where they then enter a switch matrix. The switch matrix allows this signal to travel out vertically and/or horizontally from the switch matrix. Thus single length lines can be cascaded together to travel longer distances. Double length lines are similar to single length lines except that they travel the height of two cells before entering a switch matrix. Finally, long lines go half the chip height without entering the switch matrix. This way, routes of very long distances can be accommodated efficiently. Thus the Xilinx 4000 series FPGAs can handle fairly arbitrary routing demands, though mappings emphasizing local communications can be handled more efficiently.

The Xilinx 4000 series logic cell is made up of 3 LUTs, two programmable flip-flops, and multiple programmable multiplexers. The LUTs allow arbitrary combinational functions of its inputs to be created. Thus the structure shown can perform any function of 5 inputs (using all three LUTs with F and G inputs identical), any 2 functions of 4 inputs, or some functions of up to 9 inputs (using all 3 LUTs, with F and G inputs different). SRAM controlled multiplexers then route these signals out of the X and Y outputs (Figure 2.6), as well as to the 2 flip-flops. The inputs at the top (C1-C4) provide enable, set or reset signals to the flip-flops, a direct connection to the flip-flop inputs, and the third input to the three input LUT. This structure yields a powerful method of implementing arbitrary, complex digital logic. Note that there are several additional features of the Xilinx FPGA not shown in these figures, including support for embedded memories and carry chains [21].

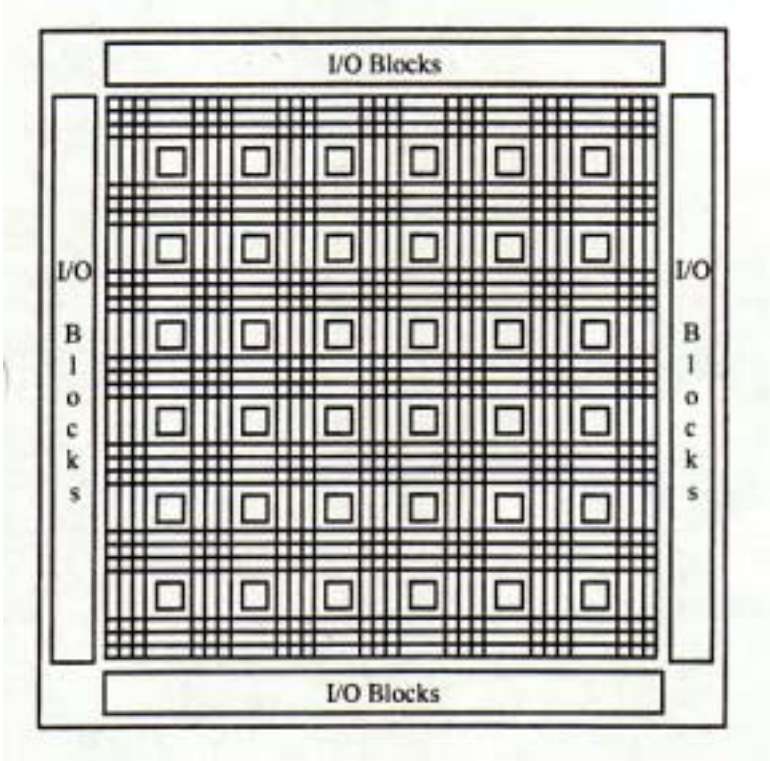


Figure 2.5: Xilinx 4000 series routing structure

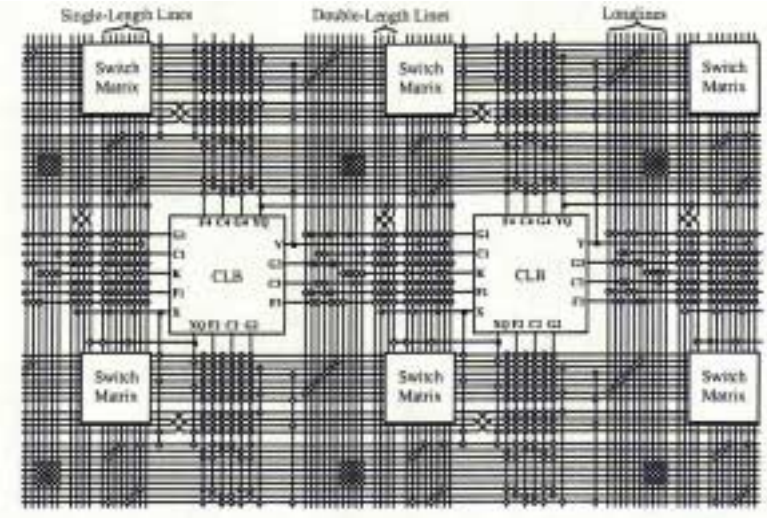


Figure 2.6: Xilinx 4000 series FPGA structure

2.6 Applications of Reconfigurable Computing to Software Radios

FPGAs are commonly used in logic emulation and prototyping systems. They are ideal for prototyping since the chips are reusable after the bugs have been fixed or the system has been upgraded. They are also widely used in logic emulation and testing the design of a custom circuit. Though software simulations can verify the accuracy of a circuit, it is done at a much slower rate compared to logic emulation circuits using FPGAs. Though the mapping (of the circuit to the FPGA) may take a long time to complete, once it is done, the system can operate at a very high speed.

FPGAs have not been used traditionally as computing devices, mostly because of their higher power requirements, low clock speed, and relatively long programming time [58]. However, in the recent years, with the development of advanced FPGAs and successful demonstrations of FPGA based computing systems, FPGAs are being re-evaluated as efficient computing devices [11] and are being used to enhance the performance of traditional digital signal processors [15].

In the case of software radios, FPGAs have the additional advantage of adding adaptability and flexibility in the final product. FPGAs at times can also help conserve silicon area, since one chip can be configured to perform more than one function and the configurations can be changed during run-time. Situations where the use of FPGAs in digital signal processing applications is most beneficial include systems with high sample rates and short word length. FPGA based DSP designs run faster as the word width decreases since the word length on the FPGA can be set exactly to the required length. In very high order FIR filters, the algorithm can be implemented in parallel decreasing the run-time. The lookup table architecture of FPGAs provide a fast and efficient way to build fast correlators.

Though hardware flexibility is desirable, the flexibility of FPGAs is not tailored for

wireless communications or signal processing applications, which sometimes lead to inefficient implementations of certain widely used functions. Further, most commercial FPGAs lack run-time reconfigurability and partial reconfigurability, which limit their use in soft radios.

One of the limitations of contemporary FPGAs is the reconfiguration mechanism. There are two primary approaches used: serial configuration and random access configuration [4]. In devices using serial configuration, the configuration storage elements are connected as a large scan chain around the entire chip. During reconfiguration, the programming information is loaded into the device and shifted throughout in a bitwise manner. While parallel loading of the configuration data is possible with some devices, close examination of the timing employed by these devices reveal an internally serial architecture. In most cases, the entire chip has to be reconfigured before use. The ability to configure one region of a chip while simultaneously executing within another region would aid not only in pipelining the configuration/execution cycle for a single process, but also in the multitasking of several processes onto the same chip. Speed of reconfiguration is another deficiency of the serial configuration method.

2.7 Design Methodologies Using Reconfigurable Computing

The first step towards implementing a soft radio on the layered architecture is the partitioning of the algorithms into processes that can be expressed as a data flow graph to fit into the stream-based modules of the processing layer. Once the basic structure is defined, the functionality of the modules can be verified in simulation using traditional software tools. A general overview of the design process is given in Figure 2.7. High level languages like Matlab and C can be used to verify the accuracy of the algorithms though they do not directly aid in the hardware implementation. VHDL and SPW

^{TM}(Signal Processing Worksystem) are good combinations for simulating the soft radio. SPW can be used to provide the environment for their simulation and the algorithms can be broken down into processes that are coded in VHDL, and the VHDL modules can then be ported on to actual hardware using various synthesis tools. The synthesized binary code can be stored in the library of the soft radio.

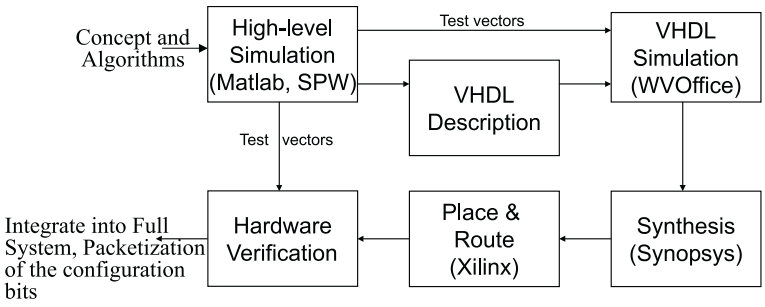


Figure 2.7: Design process for reconfigurable computing

2.7.1 Stream Based Processing

Traditional reconfigurable designs typically use separate busses or data paths for programming information and data. The programming bits use separate clocks and synchronization circuits, and once the programming is completed, data is buffered through a separate bus. This scheme, however, is not ideally suited for soft radio implementations where it is necessary to keep the design scalable as well as simple.

The soft radio architecture being developed at Virginia Tech is based on a concept called *stream-based* processing. A stream packet is a packet of known length containing either programming (configuration) information or the data to be processed. Stream-based processing provides a means to exploit the processing power attainable through deep pipelining, while still maintaining some degree of flexibility [9]. The algorithm to be implemented is first represented as a *data flow graph*. The data flow graph is

then decomposed into smaller computational primitives called *processing modules*. Each processing module performs a unique subset of the overall processing on the data and then passes the data and control information to the next module. An analogy can be drawn to an assembly line process where each module performs a specific task as the component moves forward in the assembly line, as shown in Figure 2.8.

Figure 2.8: Stream-based processing

2.8 Conclusions

An overview of existing soft/software radio architectures, trends in the industry with 3G systems and the relevance of soft radio architectures for existing and future systems is presented in this chapter. The key design issues of a soft radio include placement of the A/D converter, flexibility at the RF front end, effective data management procedures, resource allocation and smooth reconfigurability of the hardware. Reconfigurability in hardware is critical for designing efficient soft radios. While existing FPGAs provide reconfigurability and are useful in the design of soft radios, the lack of run-time reconfigurability and partial reconfigurability, and long reconfiguration times limit their use in soft radios. The next chapter introduces a new class of hardware called custom computing machines, that overcome the limitations of FPGAs.

Although a number of architectures as well as specific designs have been implemented, these radios lack a common design philosophy and a methodology that can be used with reconfigurable hardware. Further, the use of stream based processing allows for the design of soft radios that are simple and scalable in hardware. The next chapter

describes the layered radio architecture that develops a common design methodology using run-time reconfigurable hardware and stream based processing, and retains all the features of a soft radio.

Chapter 3

The Layered Radio Architecture

One of main challenges in designing a soft radio is supporting smooth reconfigurability while keeping the design simple, scalable and efficient. This indicates the need to examine the existing options for reconfigurable hardware and select an optimum mix of available technologies. Some of the traditional approaches to reconfigurable designs were discussed in the previous chapter. While these techniques are useful, they do not meet all the requirements of soft radios for supporting run-time reconfigurability, simple reconfiguration procedures, and stream based processing. A new class of hardware called custom computing machines, overcome the limitations of traditional approaches to reconfigurable computing, and is discussed next.

3.1 Use of Custom Computing Machines (CCM)in Soft Radios

The development of Custom Computing Machines (CCM) allows designers to achieve flexibility in hardware without sacrificing power or silicon efficiency. CCMs try to retain the desirable characteristics of FPGAs and ASICs for a particular application or a suite

of applications. CCMs have static hardware for frequently used cores (like multiplication for instance), which result in efficient radio designs. Other features that are typically used to enhance the CCMs are strategically placed shift registers, circular buffers, large number of I/O pins for good data throughput, etc. CCMs attempt to customize FPGAs such that the flexibility of FPGAs is retained only where necessary.

3.1.1 Stallion Architecture

Virginia Tech has developed a CCM called the Stallion that is specifically suited for flexible, high-throughput, low-power computations and is based on wormhole-reconfigurable computing [4]. Stallion supports fast run-time reconfigurability, which makes it attractive to soft radio applications. The configuration time for Stallion is in the order of microseconds while most commercial FPGAs have a configuration time in the order of milli-seconds [62].

Wormhole Run-Time Reconfiguration (RTR) attempts to address the weaknesses of FPGAs when used in a computational environment, and provide a framework for implementing large-scale rapid run-time reconfigurable CCM platforms. The essence of wormhole RTR concept is formed from independent self steering streams of programming information and operand data that interact within the architecture to perform the computational problem at hand. The stream is self steering and, as it propagates through the system, configuration information is stripped from the front of the header and is used to program the unit at the head of the stream. Thus, the size of the header diminishes as the stream propagates through the system [4]. Since the streams independently guide themselves through the system using the information contained in the stream header, the configuration process is inherently distributed. Multiple independent streams can wind their way through the chip simultaneously. Part of the system can be used to process data operands while any set of neighboring units are accepting configuration data from the header of one or more streams, thus allowing overlap of the

configuration and processing operations, as well as partial reconfigurability.

Stream based processing uses a common bus for data as well as programming information, with a header in the stream packet, that indicates the nature of the stream packet. When the stream makes use of wormhole-reconfigurable computing, the stream packets will also indicate the next module where subsequent programming packets should be forwarded. Instead of requiring separate paths for distributing programming information which would necessitate a dramatic increase in the interconnection complexity and area as multi-module systems are scaled-up, stream-based processing based upon wormhole-reconfigurable computing makes use of the data paths which greatly simplifies the implementation and programming of multiple module systems.

Figure 3.1 shows the architecture of Stallion. It consists of two meshes of 8x4 functional units and multipliers, and a network of flags and busses connected by a programmable crossbar. The layered radio architecture stream controllers allocate and then program a path through the chip originating at one of the data ports. From the dataports, the stream proceeds through the crossbar and then to either mesh of functional units. After being processed in the functional unit, the stream can continue through the mesh, go back through the crossbar to any other part of the chip, or off the chip via a data port.

The functional unit is the basic computational unit in the system and is shown in Figure 3.2. It can accept two 16-bit operands through two registers called the right and left input register. The left operand can be conditionally passed through a barrel shifter. The (optionally) shifted left operand and right operand are then sent through an Arithmetic and Logic Unit (ALU). The ALU is capable of performing various arithmetic and logic operations. The output of the ALU then passes through a conditional unit that can select, based on a condition flag, between the ALU output or the unprocessed input from the right register. The output of the conditional unit can be optionally delayed before leaving the functional unit. The output of the functional unit can be routed to other functional units, multipliers or dataports through the crossbar or the skip bus.

The skip bus is a special kind of bus that sends data to any other functional unit within one clock cycle. Finally, in addition to the condition flag, each functional unit is capable of generating and processing an array of flags, which can be used to control and monitor the functional unit's operation.

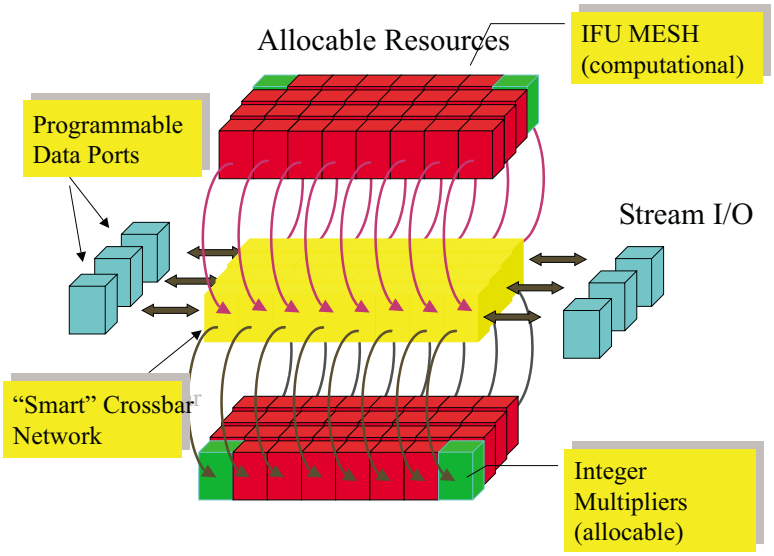


Figure 3.1: Stallion organization

The Stallion chip is manufactured using .15 micron technology and is shown in Figure 3.3. The chip has a dye size of 7.95 x 7.95 square mm, and operates at 3.3 Volts.

3.1.2 Software Tools for Design with Stallion

A compiler and simulator have been developed to assist designing with Stallion. The compiler has a MATLAB Graphical User Interface (GUI) that can be used to program the path of streams through the stallion. The simulator can be used to program the functional units, crossbar, data ports and multipliers. The simulator has a graphical display of each of these elements which can be used to program the interconnections between two elements and within an element. For example, the programming interface

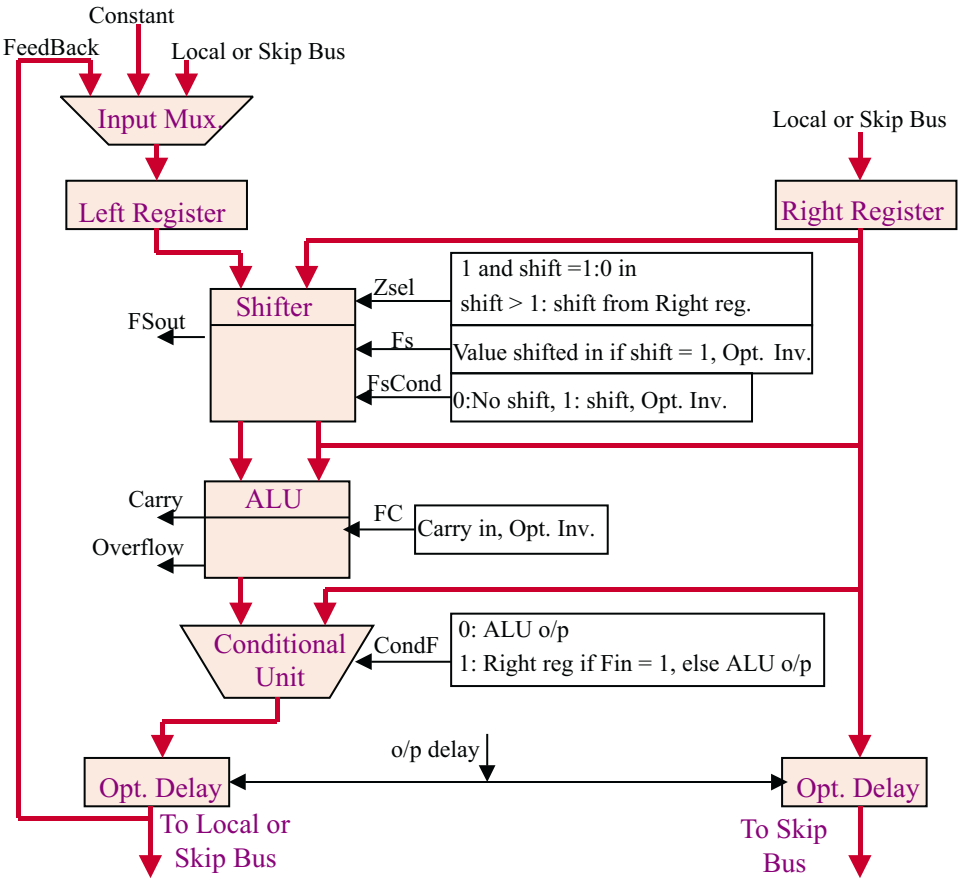


Figure 3.2: Structure of a Functional Unit (FU)

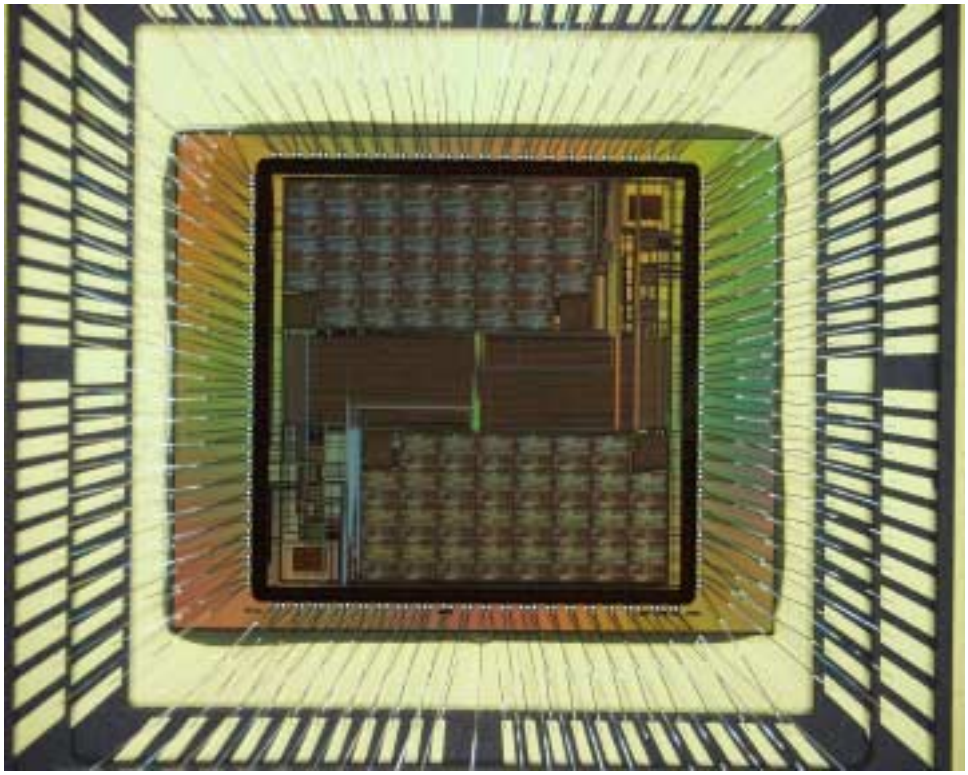


Figure 3.3: The Stallion chip

for a functional unit allows the user to set the functionality of the ALU, the shifter, the conditional unit etc., and also set the connections of the flag, skip and local busses. The compiler then generates the binary streams necessary to program the Stallion.

To verify the designs developed for the Stallion a simulator was developed using an object oriented C^{++} platform. The simulator can be used to verify logic as well as timing within the designs.

3.2 Layered Radio Architecture

The approach to designing a reconfigurable radio using *hardware paging* is formalized in the layered radio architecture [49]. Hardware paging refers to hardware modules being paged in and out of the system in a manner similar to software paging performed with the use of virtual memory. The architecture defines the necessary procedure for configuring the modules and setting up libraries. The layered architecture leverages on stream-based processing to develop a design methodology suitable for reconfigurable platforms. The use of the stream-based processing simplifies the interface between modules, making it easy to replace one module with another or add additional modules. The use of a reconfigurable hardware platform enables hardware paging.

The functionality of the radio is divided into layers, where each layer attaches/modifies the header and passes the information to the next layer. Once the processing is complete, the information is sent back through the layers the same way. The layered radio architecture defines three layers (Figure 3.4), the Soft Radio Interface (SRI) layer, the configuration layer, and the processing layer. The application software residing above all these layers consists of high-level graphics and software that interface the radio with the user. The three layers of the soft radio are implemented using stream-based processing. The SRI layer is responsible for interfacing the radio hardware with the external world, coordinating the various sources of information coming in and going out of the

radio, and prioritizing the requests made to the radio. The SRI layer also packetizes the information to form stream packets, and sends the programming and data information to the configuration layer. The configuration layer forms the stream packets that can be interpreted by the processing layer and also sends the processed data back to the SRI layer. The processing layer consists of a series of reconfigurable modules called processing modules that perform the actual operations on the data to implement the functionality of the radio. The functions of each of these layers are described in detail in the next section. Once the processing is completed, data is sent back through the configuration layer to the SRI layer, which in turn outputs it to the host PC.

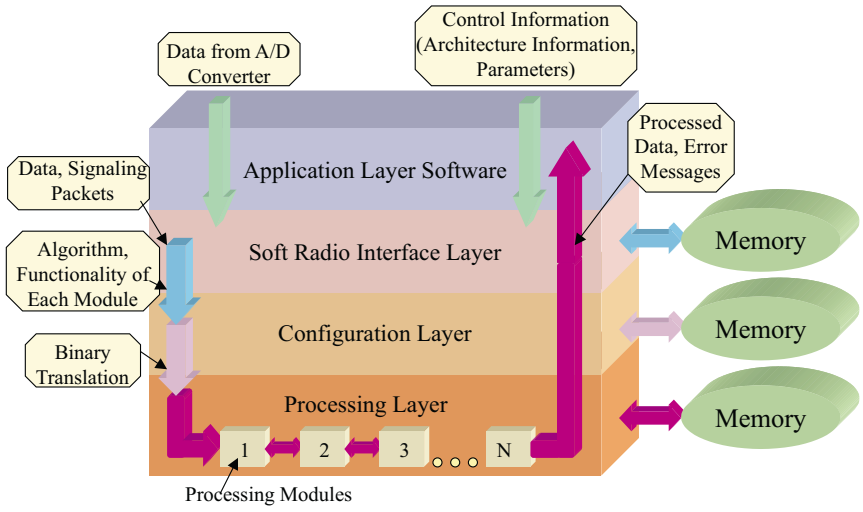


Figure 3.4: Layered radio architecture

The stream packets coming out of the processing layer contain the processed data as well as error and status messages. When only data is being processed by the radio, the configuration layer acts as a bypass layer and sends the information to the SRI layer. However, if an error occurs in the processing layer while executing control packets, the configuration layer has two options. It can either re-transmit the control information or send the error information to the SRI layer which in turn can be programmed to re-transmit the code or let the higher layers handle the error.

3.2.1 Use of Stream Based Processing in the Layered Architecture

A typical stream packet has a header and a main body of information as shown in Figure 3.5. The header indicates the nature of the packet, assisting in the interpretation of the main body of information. There are also bits in the header to indicate whether the packet is valid or not. There are three main types of packets: programming or control packets, data packets, and signaling packets. A packet that has programming information has headers indicating the nature of the desired programming. There can be two types of programming information: packets that change the parameters of the existing functionality of the module or packets that change the functionality of the module itself. The signaling packet indicates certain events that can affect the functioning of the module, while not physically changing the functionality or the parameters of the module. A data packet contains the actual data to be processed. When a processing module encounters a stream packet, it takes the necessary action based on the nature of the stream packet.

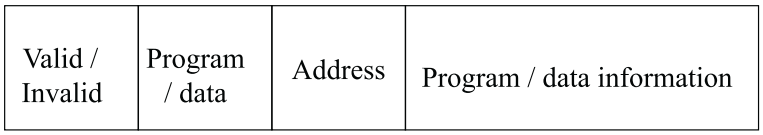


Figure 3.5: Structure of a stream packet

3.3 Description of the Layered Radio Architecture

The layered radio architecture divides the functionality of the radio into layers, where each layer attaches/modifies the header and passes the information to the next layer. Once the processing is complete, the information is sent back through the layers in the

same way. Each layer's functionality is isolated from the other layers, but designed in such a manner that information is passed between the layers utilizing stream-based processing. The layered radio architecture defines three hardware layers, the Soft Radio Interface (SRI) layer (top layer), the configuration layer, and the processing layer (lowest layer). The SRI layer and the configuration layer together aid in configuring the processing layer to the desired functionality. The actual data processing is performed in the processing layer. The processing layer requires run-time reconfigurable hardware while the higher layers do not have this constraint.

The information coming into the radio, whether from the A/D converter or from a larger network of nodes, can be either data to be processed or programming information requesting a new setup of the radio or a modification to the existing setup. When handling packets, whether generated within the layer or passed into the layer, each layer decides if it is the appropriate layer to handle this packet or if the packet should be handed off to a higher or lower layer.

Should information arrive (potentially over-the-air) requesting the change to a different system (such as W-CDMA instead of GSM), the layered radio architecture handles the change in the following manner. After the SRI layer makes the determination that a new system is required, the SRI layer sends instructions in the form of programming packets to the configuration layer which will reconfigure its operations accordingly. It is important to note that the SRI layer does not contain the bits needed to configure the hardware. Rather, it contains the list of algorithms that are to be used and their sequence of interconnection. The configuration layer has in its memory the actual bits needed to configure the processing layer hardware. Based on the programming packets sent by the SRI layer, the configuration layer uses the configuration bits extracted from its memory to reconfigure itself to organize the packet streams it will be sending to the processing layer. The processing layer consists of a series of reconfigurable modules called processing modules that perform the actual operations on the data to implement the functionality of the radio. The processing layer will reconfigure itself based on the packets it receives

from the configuration layer. Once the radio is configured, packetized data will be sent through the layers using the same bus that was used for the programming packets.

The following sections give a functional description of the three layers. The actual implementation of these layers will be discussed in the next chapter.

3.3.1 Soft Radio Interface Layer

The Soft Radio Interface (SRI) layer receives the incoming digitized RF data and control information from the host PC that contains the desired configuration and settings of the radio. The SRI layer transmits processed data to the host PC, performs initialization of the system on power-up and performs the first step in configuring the radio. This layer has the system level description of the desired radio configuration in its local memory, which contains the algorithm codes for each processing module needed to implement the system. This system level description is sent to the configuration layer, which in turn configures the processing layer modules. For instance, if the user desires to use the IS-95 standard, the SRI layer sends the codes of all the algorithms needed like demodulation, spreading codes etc. in the appropriate sequence. The configuration layer then uses the information in it's local memory to implement each of these modules on the processing layer hardware. It is important to note that the SRI layer does not contain the bits needed to configure the hardware. Rather, it contains the codes for the algorithms that are to be used and their sequence of interconnection.

The external packets coming in to the SRI layer can be either data or control packets. The control packets can be of two types; they can be configuration packets that set up the radio configuration or request status information. The control packets can also be updates to the radio, i.e., they can program a new configuration into the radio or modify the nature of an existing configuration. This information can either be input by the user or handled as over-the-air updates by the system. This type of control information correspond to memory updates performed by the layer as shown in Figure

3.6. Once the user or the network decides on the system and algorithms to be used, the SRI layer inserts the corresponding codes and forwards it to the configuration layer. Data that follows will be processed using these preset algorithms until the user decides to change the configuration of the system. Control packets containing over-the-air updates are used to change the configuration of the library. The requests from the PC (control information), as well as data from the A/D converter, is buffered and prioritized by the SRI layer and sent to the configuration layer. The structure of the SRI module for processing data coming in from external sources is shown in Figure 3.6.

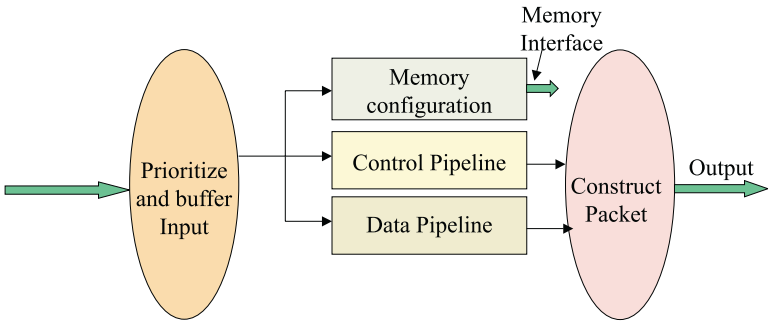


Figure 3.6: Soft Radio Interface Layer: Forward direction

The stream packets received by the SRI layer from the configuration layer have more information in them. The configuration layer sends the processed data back to the SRI layer along with other status and error messages. First, the SRI layer checks to see if the data contained in the packet is valid. If it is valid and the packet contains data, it is sent to the host PC, along with error messages if any. When the SRI layer sends program or control information that was requested by the host PC to the configuration layer, the packets from the configuration layer will contain acknowledgments that indicate if the operation was successful. These messages can either be sent back to the host PC or the SRI layer can re-transmit the control packets. The structure of the SRI layer for processing data coming from the configuration layer is shown in Figure 3.7.

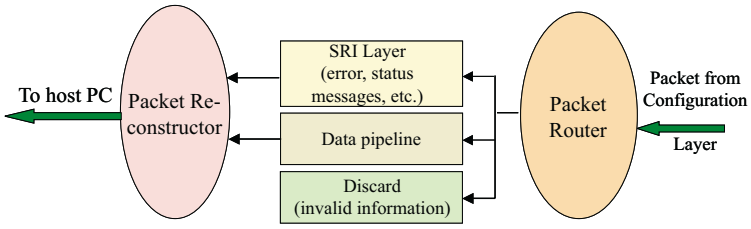


Figure 3.7: Soft Radio Interface Layer: Reverse direction

3.3.2 Configuration Layer

The functional layout of the configuration layer is similar to that of the SRI layer. The information from the SRI layer indicates whether the packet is a control or data packet. If it is the former, the packet contains the algorithm codes of the desired configuration, or a modification/update to the existing modules present in memory. For example, the control packet may request a DQPSK demodulator followed by a Viterbi decoder. The appropriate configuration code is extracted from the local memory and sent to the processing layer with the address/addresses of the processing layer module/modules. The local memory contains a table that has the addresses of the stored configuration code for each operation. The configuration layer also maintains a status list of the processing layer modules. When the configuration layer receives a data packet, the configuration layer headers are attached to the data and sent to the processing layer. Data coming in from the processing layer is sent back to the SRI layer for appropriate action.

3.3.3 Processing Layer

The processing layer forms the core of the soft radio that performs the actual processing of data. This layer is responsible for processing data sent from the configuration layer and sending the processed data back to the configuration layer. It consists of a set

of linearly connected processing modules, as shown in Figure 3.8. Each processing module has the capability of reconfiguring itself to perform the desired function without disrupting synchronization in the main pipeline. The main flow of data is in the forward direction from the first to last module, as shown in Figure 3.8. Each processing element maintains the continuity of this stream pipeline at all times. Each processing element has both, a static and a reconfigurable section. The static sections have the capability of configuring the reconfigurable section using the control packets sent by the configuration layer. Data can thus either bypass each processing element entirely or be routed through the reconfigurable section for processing. The structure and operation of the processing element is described in the next section.

Many communication circuits have feedback loops. To accommodate such systems, the stream pipeline is designed to be bi-directional in the processing layer. The input/output ports of the processing element can either be implemented as bi-directional ports or can have a separate bus in the reverse direction. The reverse slots/reverse buses are used to implement feedback loops. If the ports are bi-directional, the main clock cycle is divided into forward and reverse slots, and the directions of the interconnections are reversed in the reverse slots. If a separate reverse bus is used, then with every clock cycle, a module both accepts and outputs a forward and reverse packet to the neighboring module. Reverse packets are not needed in the absence of feedback loops.

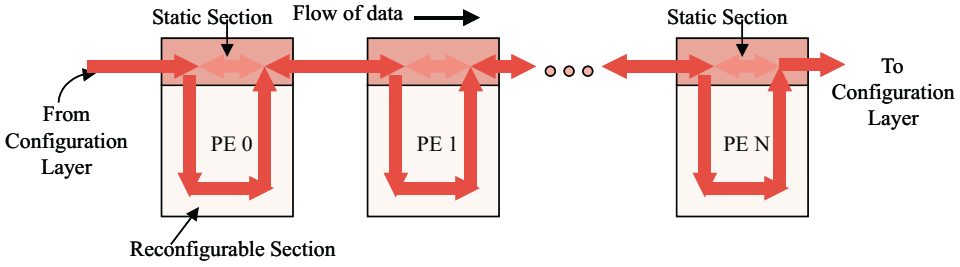


Figure 3.8: Processing elements in the processing layer

When valid data enters an appropriate module, it is processed and sent to the next

module. The address field is updated to indicate the next module that needs to act on the data. Similarly, if there is control information in the packet for the module, the corresponding code is executed to change the configuration of the reconfigurable section. If there is any latency while the module is being reconfigured, the static portion of the module buffers the subsequent incoming data and sends out idle packets. However, if it gets control packets intended for modules further down the chain, then those packets are bypassed and sent to the next module. Thus the continuity of the stream is maintained at all times.

Data in the stream packet is sent to the processing pipeline. At the end of the processing pipeline, the packet is reconstructed, and error or status messages if any are added to the header. When the processing is completed, data is sent from the output module back to the configuration layer and finally to the SRI layer that delivers the output data. If the packet contains program information, the module checks to see if the address on the header matches the module address. If so, the code present in the body of the packet is executed in the configuration pipeline, thereby changing the configuration of the processing pipeline. It is also possible to have special error packets to handle more serious errors. If an error occurs in the execution of the code, the module sends a signaling packet indicating the location and nature of the error to the configuration layer. The address field is set to a special value so that none of the succeeding processing modules intercept the packet containing the error message.

Structure of the Processing Element

The functional description of each processing module is shown in Figure 3.9. Running on each processing element are three sets of pipelines and a state machine that interprets how the packet is channeled through the pipelines. The processing pipeline processes the data, i.e., performs an operation on the data as part of the demodulation process. The configuration pipeline controls the hardware configuration of the processing pipeline.

The ability of the configuration pipeline to execute programming information makes it possible to modify the low-level parameters and functionality through high-level software. The bypass pipeline allows the packet to bypass the module if the packet is not intended for the module. The bypass pipeline is present to ensure that the module does not corrupt data it is not supposed to act on. It is important that each of these pipelines have the same amount of delay, so that the packets are synchronized with the main clock cycle. At the end of the pipeline, the stream packet is reconstructed with the updated header and routed to the next module.

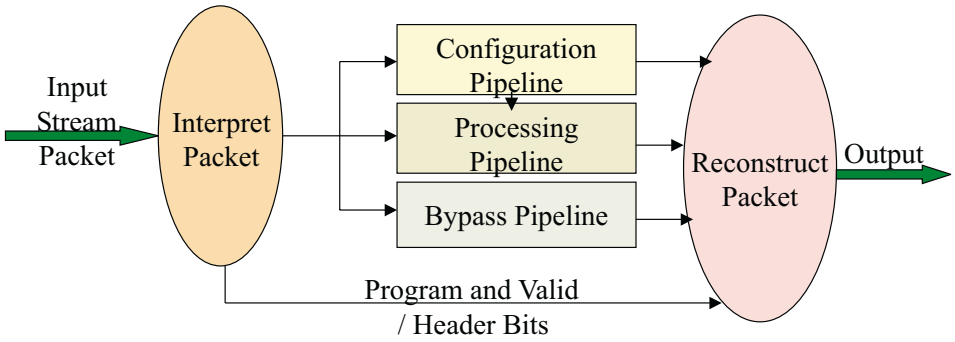


Figure 3.9: Structure of a processing element

Figure 3.9 gives a functional description of the processing element. However, when this structure is implemented in hardware the structure is shown in Figure 3.10, which also shows the distribution of the module into the static and reconfigurable sections. The static sections are identical in all the processing modules. An *empty* processing module can be thought of as a module with no special programming in the data interpretor and reconstructor or processing pipeline. In this situation all data bits (whether or not they are intended for the module) are bypassed. In other words, the processing pipeline is also a bypass pipeline and the data interpretor and reconstructor act as a wire connecting the input and output. This is the default setting on power up. Before the radio is configured, if data is sent, the processing layer essentially sends the data back to the configuration layer without any processing.

The static section of the processing module consists of a packet interpreter, configuration pipeline, and packet reconstructor. The packet interpreter is essentially a switch. By looking at the header bits, all control packets are sent to the configuration pipeline and all data and signaling packets are sent to the processing pipeline.

The function of each of these packets is described in Section 3.2.1. The configuration pipeline accepts all the control packets and has the capability to configure the processing pipeline accordingly. The modules maintain the configuration to which they are set and act accordingly on valid data until the configuration is changed. The packet reconstructor accepts data from all the pipelines, buffers them if needed, and sends them out to the next module.

All non-control packets are routed to the data interpreter. The data interpreter takes care of the byte ordering and the address field and decides whether the packet has to go through the processing pipeline or the bypass pipeline. The data reconstructor reconstructs the processed packets and sends them to the next module. For instance in a deinterleaver, the data reconstructor gets the bits in the correct order and constructs new packets.

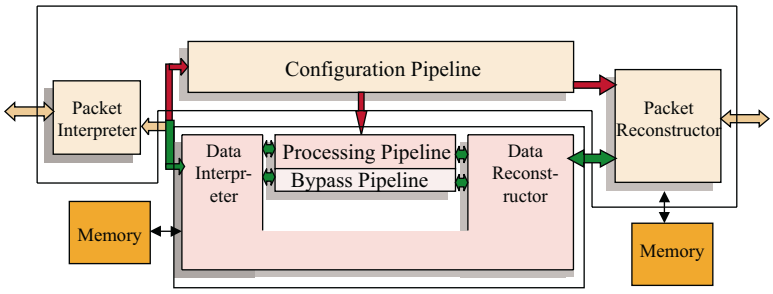


Figure 3.10: Structure of a processing element

When a stream packet enters a processing module, the module interprets the packet and performs the necessary action. The processing element examines the contents of a packet only if the valid bit is set. The valid bit is introduced to maintain synchronization

between the modules. Every clock cycle, each processing element accepts a packet and sends out a packet. However, the module acts on the packet only if the valid bit is set. Similarly, a module sends out an invalid packet instead of inserting wait states.

3.4 Advantages and Tradeoffs in the Layered Architecture

The layered architecture provides the framework for building a flexible soft radio at the expense of the overhead for packetizing data. It is possible to implement radically different radios on the same hardware platform through the use of this architecture. Additionally, the architecture is conducive to the implementation of adaptive modulation schemes, achieved by reconfiguring the hardware in real-time.

One of the main advantages of the architecture is its excellent hardware reusability. For example, core units like BPSK and QPSK demodulators can be used in a wide variety of standards when built in hardware with the right parameters. Thus it is possible to build libraries of hardware functions much like building libraries of software functions.

The layered architecture has good data flow properties and a simple interface between the processing layer modules, which make it scalable at the system level as well as the hardware element level. The use of stream-based processing ensures that each processing module has one input and output bus that is common for control information and data bits. This simplifies the interface between two processing modules and makes adding additional hardware modules easy. Furthermore, with the use of a reconfigurable chip where every pin is a bi-directional data/program port, the aggregated I/O rate for the chip can be made to be very high. The core computing units have the basic algorithm, while the data interpretors and reconstructors *customize* the core computing units to each standard. In most cases, to add a new standard, no new core computing units

need to be added once the libraries have been constructed. For instance, existing core computing units used to perform demodulation, equalization, and decoding can be used to implement a new standard. Only the data interpretors and data reconstructors need to be stored for the new standard.

The simple interface between two processing modules also makes it easy to integrate designs from different engineers. There are no constraints on the design of the processing modules as long as they are capable of accepting a stream packet and sending out a stream packet each clock cycle.

Over-the-air updates can be performed by changing the contents of the SRI configuration layer libraries, which can be accomplished by two different approaches. In the first approach, the actual bit files stored in the libraries are transmitted to the radio and each layer modifies its library based on the programming information accompanying it. This approach however, requires the transmitter to have prior information about the hardware specifics of the receiver, which could be vendor dependent. Thus, it may not be the commercially viable solution. Alternatively, some standardized high level programming language can be used to transmit the code to the radio. The SRI layer then compiles the code to generate the bit files needed for the radio. The SRI layer modifies its own library contents using the newly generated bit files. The SRI layer also packetizes the bit files corresponding to the configuration layer and assembles them into appropriate programming packets. The second approach assumes a compiler within the SRI layer.

3.5 Example: Baseband Implementation of an IS-136 Receiver

In this section the methodology for building the baseband system for an existing cellular standard the IS-136, is described. The structure of the IS-136 transmitter is shown in Figure 3.11. The output of the VSELP speech coder is separated into class 1 and class 2 bits, based on their importance in the speech decoding process, with class 1 bits being given a higher priority. The 12 most perceptually significant class 1 bits are encoded using a Cyclic Redundancy Check (CRC) computation. If the CRC check fails at the receiver, the entire frame is discarded as having too many errors. The encoding process generates 260 bits from 159 speech bits for each 20ms frame. Before transmission, data is interleaved over two time slots with speech data from adjacent speech frames [26].

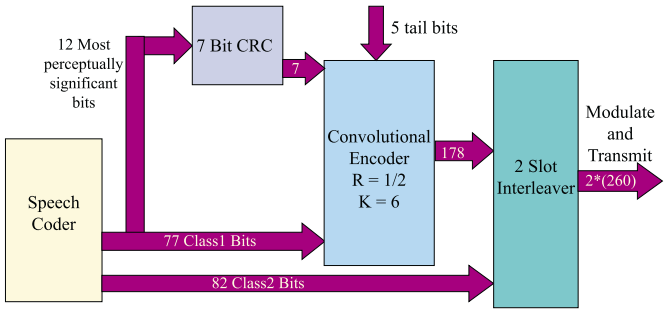


Figure 3.11: IS136 transmitter

In a conventional receiver, the frames are first de-interleaved. The class 1 and class 2 bits are then separated, processed, and fed into the speech decoder. This procedure inherently requires interconnections between modules that are not merely sequential. This section shows the high level mapping of the baseband algorithms into the layered architecture.

Figure 3.12 shows the layout of the processing layer and the interconnection of the modules. The first module is a standard DQPSK demodulator. The output of the

demodulator is sent to the de-interleaver which complies with the IS-136 air interface. The de-interleaver separates the class 1 and class 2 bits using it's stored knowledge of the location of the bits within a frame. The packets containing the class 2 bits are addressed directly to the speech decoder, hence bypassing the CRC and Viterbi decoder. The class 1 bits are addressed to the Viterbi decoder. The packet constructor at the output of the Viterbi decoder identifies the bits that need to go into the CRC check and assembles them in a separate packet. The other decoded class 1 bits are sent to the speech decoder. At the speech decoder, the packet interpreter, aligns the bits of one frame so that the speech decoder can generate the speech waveform.

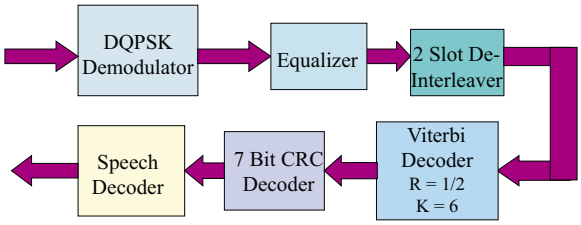


Figure 3.12: Processing layer modules (baseband) for IS136 receiver

3.6 Conclusions

A new architecture called the layered radio architecture was introduced that provides a formal yet open design methodology for soft radios. The architecture was developed after a detailed study of the needs and design issues in a soft radio. The use of CCMs and stream-based processing was found most suitable for designing soft radios that used hardware paging. Hardware paging allows only the system in use to be implemented in hardware. The layered approach lends itself easily to standardization of soft radio systems and also allows code reuse. The processing modules and header information can also be easily standardized, allowing support from a variety of vendors. One of the

main advantages of the structure of the processing modules is that the interface between two modules is very simple. Each module is able to accept a stream packet and send out a stream packet in the forward and reverse directions in each cycle. The layered architecture is scalable and completely flexible. The next chapter discusses the details of the implementation of the three layers and describes a methodology to implement the functionalities of these layers in hardware.

Chapter 4

Implementation of the Layered Radio Architecture

The layered radio architecture defines a three layered hardware architecture based on stream based processing using run-time reconfigurable hardware. The layered radio architecture is based on dividing a given transceiver system into two levels of abstraction between the SRI layer and the configuration layer, which together configure the processing layer for any desired functionality either once or multiple times during the implementation of the system [50]. The preceding chapter discussed the functional description of each of the layers, interaction between the layers, configuration of the radio, loading new configurations and the flow of data and control messages within the radio. This chapter presents algorithms for implementing the three layers that describe how a system can be split into different levels of abstraction, stored in memory and downloaded without interrupting the stream pipeline.

Each radio algorithm is partitioned using a modular structure and is represented as a parameterized function. The SRI layer stores the system level information along with the parameters used for the desired configuration, while the configuration layer stores the configuration bit sequences needed to program the processing layer elements. Each

algorithm can use two types of parameters called Embedded Variables (EV) and Non-Embedded Variables (NEV). Embedded variables (EV) are those variables that can be changed by altering one or more packets of the Stallion configuration bit stream. Non-embedded variables (NEV), on the other hand, enter the system through one of the data ports.

For example, an implementation of a generic variable length FIR filter uses filter length and filter coefficients as two of its parameters. The FIR filter can be implemented as a multiply-accumulate (MAC) operation with a counter that is controlled by the length of the filter. The following example of a variable length FIR filter of length $(m+1)$, demonstrates the use of EVs, NEVs and in parameterized hardware design. Figure 4.1 shows a mapping of a variable length FIR filter on Stallion. The mapping uses four functional units labeled B - E, and one multiplier labeled A. The output of the FIR filter $y(n)$, is calculated as

$$y(n) = \sum_{i=0}^{m+1} x_{(i)} w_i \tag{4.1}$$

where, x_i is the input data and w_is are the FIR filter coefficient. x_i and w_i are each 16-bit values and the output of the multiplier unit A, is 32 bits. The lower 16-bits are fed to the right register of FU D. The FU D, performs the accumulate operation, adding the new product in the right register to the previously computed sum in the left register. The conditional unit can select either the output of the ALU or the right register and is controlled by $Flag1$. If $Flag1$ set to '0' selects the output of the ALU while $Flag1$ set to '1' selects the output of the right register. Once $y(n)$ is computed $Flag1$ changes to '1' for one clock cycle and the accumulator is reset with the contents of the right register. The FUs B,C, and E together form a counter and control the value of $Flag1$. $Flag1$ is set to the MSB of the ALU output of FU E. This value is initially '0'. The variable A (initially set to '0') is incremented in FU C, which also acts as an accumulator. The incremented value of A is subtracted from a constant m stored in FU E. As long as the

result $(m - R)$ is not negative, the MSB of the ALU output of FU E is always zero. When R becomes greater than m, then $Flag1$ (MSB of the ALU output) is set to '1', which triggers a set of reset operations. In FU C, the variable A is set to '0' since $Flag1$ controls the conditional unit. In FU D, the conditional unit selects the contents of the right register which is then stored back in the left register, thus starting a new set of MAC operations. Thus the MAC operations are performed $(m + 1)$ times and the is an FIR filter of length $(m + 1)$.

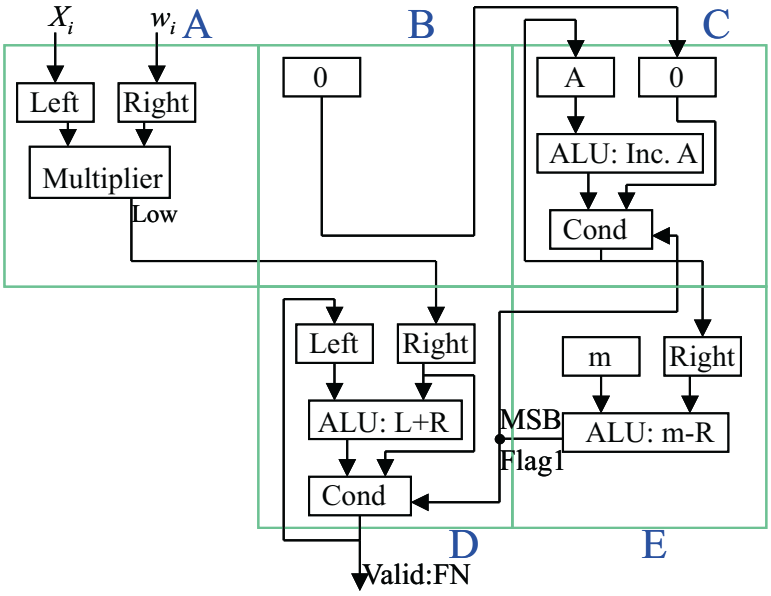


Figure 4.1: Implementation of a variable length FIR filter on Stallion

The length of the filter controlled by the constant m, stored as a constant in the register of a functional unit is an embedded variable, while the filter coefficients fed in through the data ports are non-embedded variables. It is important to note that the differentiation between EVs and NEVs depends on the implementation of the algorithm in the processing layer and not just the algorithm. However, the architecture allows both these types of variables and accounts for them in the design.

4.1 Implementation of the SRI Layer

The SRI layer contains the Stallion configuration codes along with the EVs and NEVs of each configuration in its memory. For each system, the SRI layer stores a configuration table for each of the processing elements used in the processing layer. The configuration table indicates the sequence of the SRI configurations, the number of cycles, and the next configuration as shown in Table 4.2. Each SRI configuration consists of a Stallion configuration code along with the EVs, the NEVs, and a stopping criterion. The SRI layer is unlike the configuration layer where the actual configuration bit streams are stored and the embedded variables are part of the bit stream. Each configuration may have to be run for a certain number of cycles that could depend on the input to the radio, or the output of the radio. If the number of cycles is controlled by the input to the radio, then the information is sent in the form of a control packet to modify the state table in the configuration layer. If however, the number of cycles is not known initially, and cannot determined by external sources, but is dependent on the processed data, then a stopping criterion is stored in the SRI layer, that decides the number of cycles that each configuration needs to run. For example, during acquisition, the algorithm need to run until the signal is acquired, which is determined by the output of the radio. The stopping criterion checks for a certain value, or range of values, at a particular output of the processing layer. The SRI layer programs the Stallion configuration state table stored in the configuration layer using this information.

4.1.1 Memory Organization

The SRI layer memory shown in Figure 4.3 contains the algorithm codes and parameters that are downloaded to the configuration layer. The SRI layer memory has a main header that contains information about the total number of processing elements present in the processing layer and the size of each of these processing elements. This information is

	0	1	2	3	4		31
SRI Config. Code	1	2	7	5	23		0
Cycles	3	1	0	4	1	. . .	0
Next Index Ptr.	1	100	100	4	1		0

Figure 4.2: Soft radio interface layer state table

used to validate new configurations before they are stored in the radio. The next section in the memory contains the system descriptions. The header contains the number of valid systems, followed by the details of the processing elements used for each system. The system description contains the processing elements used and the SRI configuration table for each processing element. The next section in the memory contains information about the EVs, NEVs, stopping criterion and the next configuration. The stopping criterion and the next configuration information is used, only if they are not known a priori in the SRI configuration table. The stopping criterion tells the radio when to stop executing the current configuration and the next configuration value indicates the next configuration to be loaded based on the stopping criterion.

4.1.2 Configuring the Radio

The SRI layer performs the following functions for configuring the radio with an existing configuration that is stored in memory.

• Raise the hold line. This operation stalls the execution of the Stallion configuration table stored in the configuration layer. The SRI layer can now tell the radio about the new configurations that need to be loaded.

•Load the Stallion configuration tables. The Stallion configuration tables are sent to

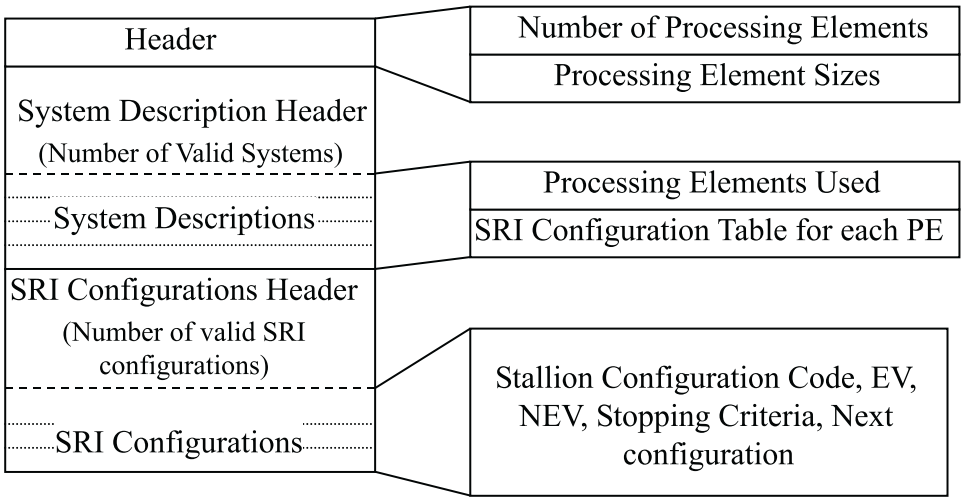


Figure 4.3: SRI layer memory organization

the configuration layer through a series of control packets.

•Load the EV and NEVs for each processing element. The values of the EVs and NEVs are then sent to the configuration layer. The configuration layer now has all the information needed to download and execute the desired configuration. The configuration layer inserts the EVs and NEVs in the appropriate packets before downloading the information to the processing elements (Stallion).

• Reset the state table in the configuration layer. This ensures that the configuration layer starts to execute the Stallion configurations starting at the correct configuration.

•Release the hold line. Once the hold line is released by the SRI layer, the configuration layer takes over configuring the processing elements based on the new information provided.

For storing a new configuration, the corresponding values are written into the configuration layer memory as defined by the packet structure described in the following sections.

4.2 Implementation of the Configuration Layer

The bit streams needed to configure Stallion are stored in the configuration layer memory. The memory is split into two sections for data and stored configurations. Figure 4.4 shows the structure of the configuration layer. The data module buffers data and sends it to the processing layer when requested by the controller. The configuration module is responsible for maintaining the stored bit streams and sending the configuration packets to the processing layer when requested by the controller. The controller coordinates the data module and the configuration module. When a new configuration is to be loaded into memory the controller enables the configuration module to output the necessary configuration packets to the stream router. Once all the packets are sent, the configuration module sends the "all done" flag to the controller which in turn enables the data module to output the data packets to the stream router. The controller also tells the stream router whether to get the next packet from the data or the configuration module.

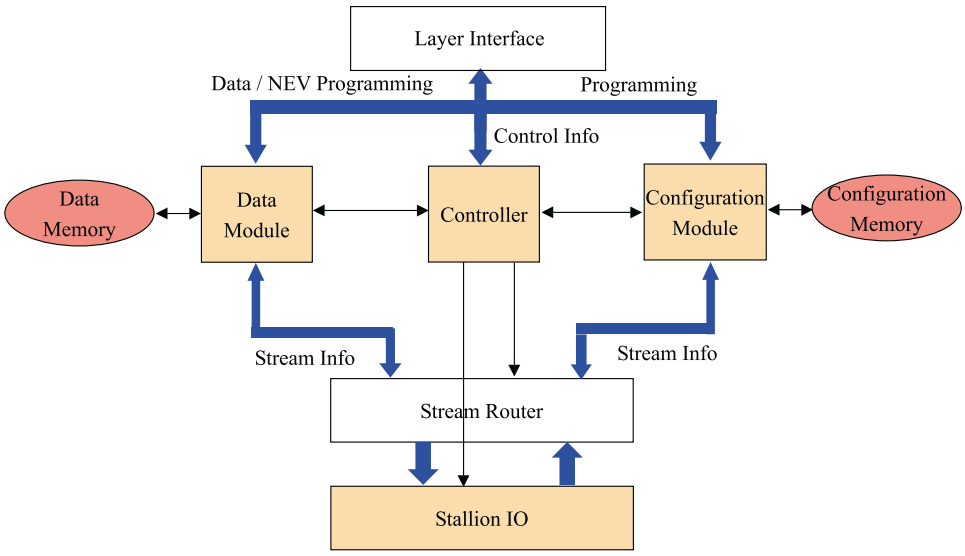


Figure 4.4: Structure of the configuration layer

The SRI layer sends the controller the information about the configurations to be loaded

in the form of a table shown in Table 4.1. The configuration layer has to be supplied with such a table for each Stallion (processing element)in the system. The table indicates the configuration to be run, the number of cycles, and the subsequent configuration to be run once the current run is completed. The SRI layer also has the ability to suspend the operation of the configuration layer and modify any of the entries in the system state table.

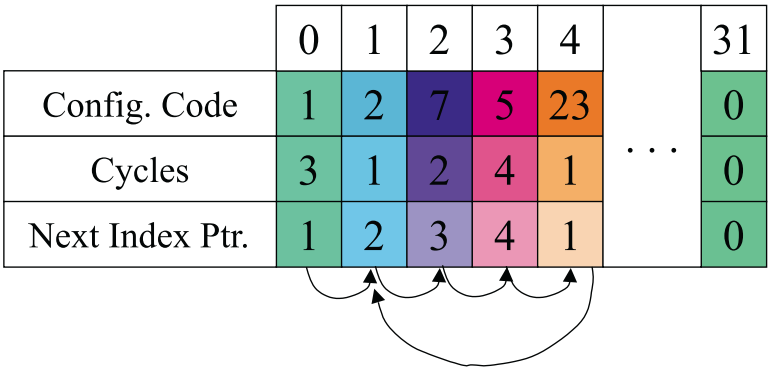


Table 4.1: Configuration layer state table

The following sections describe the packet structures for the configuration layer. The packets can be of variable length based on the header bits that indicate the nature of the packet. All the packets contain the program and the valid bit. Having these bits in each packet aids in synchronization and maintains scalability of the system. The program bit allows for the same bus to be used for data and programming information thus simplifying interface issues in hardware. The use of the valid bit has a few advantages if the receiver clock speeds are not exactly matched to the bit rates. In a soft radio that has different systems with different data rates, it eliminates the need to have a dedicated clock for each system. The overall clock rate of the radio has to be higher than the processing rates since there has to be some allowance for the programming overhead.

4.2.1 Controller Packet Description

Figure 4.5 shows the structure of the packet sent to the controller from the SRI layer. Each set of bits have a specific functionality as described below.

Hold: Halts the Stallion Configuration State Machine (SCSM) once the next set of output data is read back from Stallion.

Reset: Resets the SCSM to configuration code A and cycle 0 in the Stallion Configuration State Table.

Clear Stallion Configuration State Table: Clears all the contents of the Stallion Configuration State Table

Change Stallion Configuration State Table: The SCST contains the list of Stallion configurations to be implemented, their order, and the number of cycles for each configuration. The packet will contain a revised entry for the state table.

4.2.2 Configuration Module Packet Description

The structure of the configuration module packets is shown in Figure 4.6

Add/Delete: Indicates whether the configuration is being added or deleted. The valid configuration list is updated based on this information.

Configuration Code: Indicates the SRI layer code for the configuration information contained in the packet.

Type: These two bits indicate whether the packet contains information for the configuration header, an embedded variable or a programming stream.

Configuration Header: Packet contains the configuration header bits

Embedded Variable: The subsequent bits contain the variable number, the destination stream number, the packet number and the indirection flag. If the flag indicates a value,

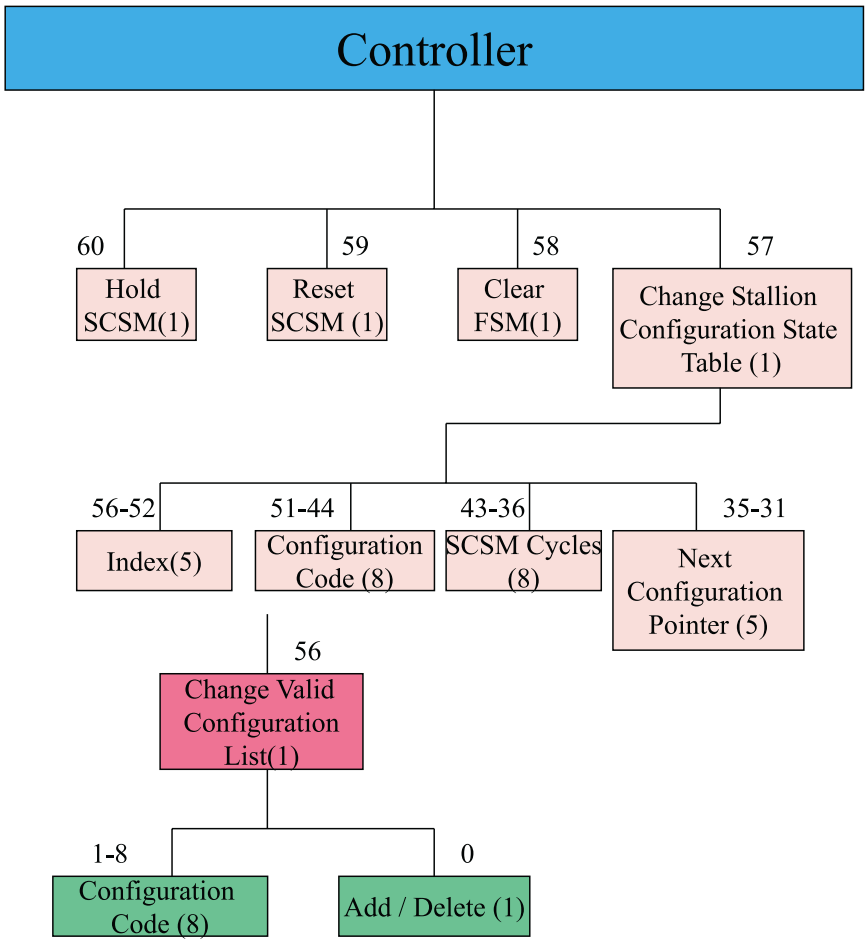


Figure 4.5: Controller packet

then the information section of the packet contains the actual value of the embedded variable. Instead, if the packet contains an address then another flag bit is used to indicate whether it is an embedded or a non-embedded variable.

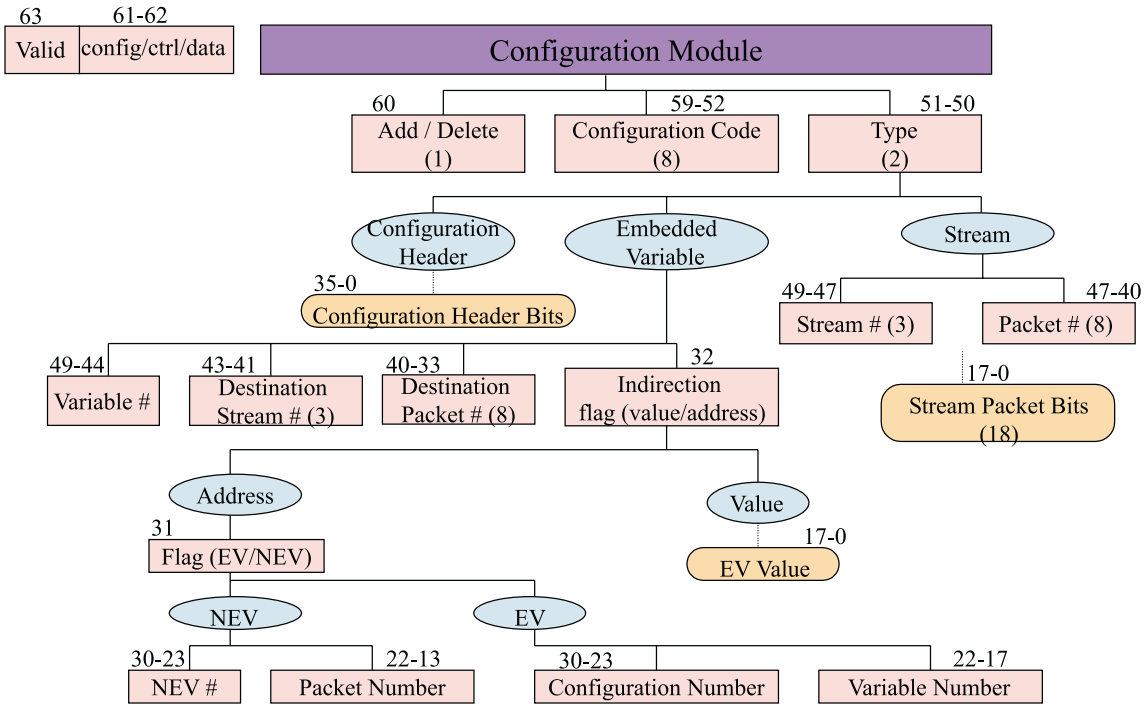


Figure 4.6: Configuration module packet structure

4.2.3 Data Module Packet Description

Figure 4.7 shows the structure of the data module packet.

I data: Indicates that the packet contains a single sample of I data in bits 0 - 15

Q data: Indicates that the packet contains a single sample of I data in bits 16 - 31. A packet can contain I data as well as Q data.

NEV: If this bit is set, it then indicates that the packet contains a single value of an NEV array. The NEV bit cannot be set if either the I or the Q data bit is set.

NEV/Clear: Clears the contents of all the 1024 locations of the NEV array, setting them to 0. The NEV number is contained in NEV/NEV var #.

NEV/Packet #: If clear is not set, this indicates the particular NEV packet number (0 - 1023).

NEV/Flag: Indicates whether the NEV packet that is being updated is a new value or a pointer to another NEV.

NEV/Addr/NEV#: Indicates the where the NEV points to.

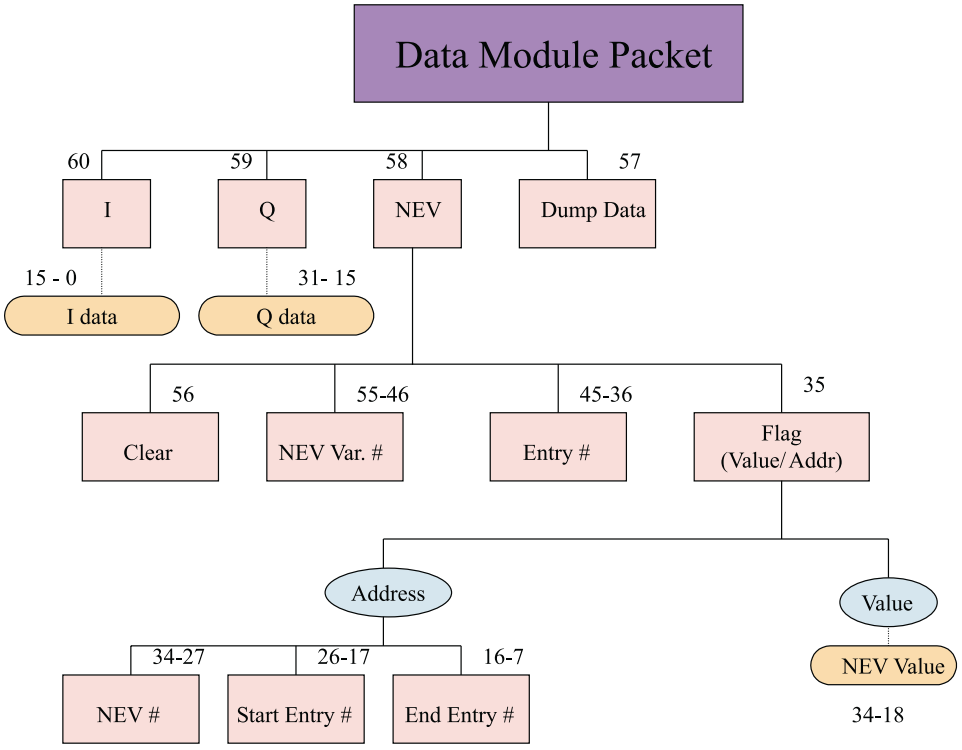


Figure 4.7: Data module packet structure

4.2.4 Configuration Layer Memory Organization

The configuration layer memory is split into two independent sections, one for storing data and one for storing the configuration bit streams. The data module buffers incoming data into the data memory and outputs it to the stream router when requested by the controller. The configuration module controls the configuration memory.

Data Memory

The data memory is divided into three parts. The first two parts contain the I and Q data while the third part contains the Non-Embedded Variables (NEV). Pointers, I_{start}, I_{end}, Q_{start}, and Q_{end} are used to update the location of the start and end of I and Q data. There can be a maximum of 127 NEVs, each having a length of 1022 and a header of length 2. The header of the NEV memory contains the length of the NEV. If the header is set to zero, then the NEV has not been defined. The next memory location contains the offset value for the NEV. Offset denotes the start of the NEV in the 1022 long array. The variables wrap around once the end of the array is reached.

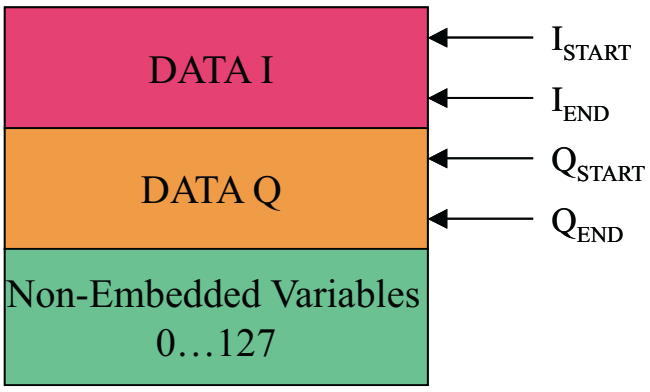


Figure 4.8: Data memory of the configuration layer

4.2.5 Configuration Memory

The structure of the configuration memory is shown in Figure 4.9. The configuration memory can store a maximum of 256 different Stallion configurations. Each configuration contains a header, list of embedded variables and the configuration streams. The configuration header is stored in a single memory location in a 32 bit word.

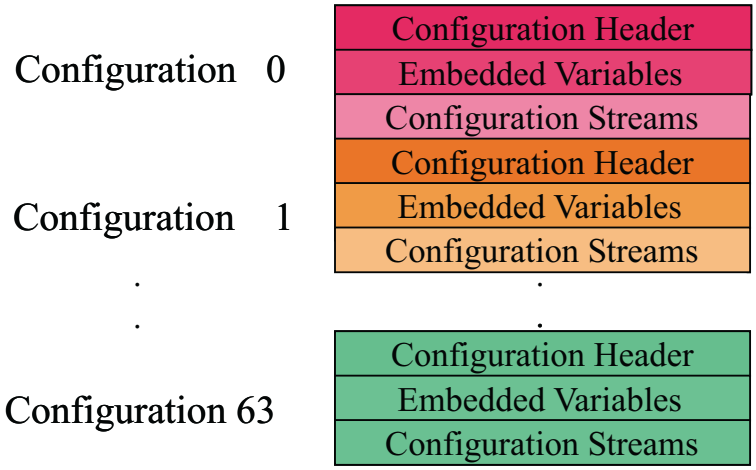


Figure 4.9: Configuration memory of the configuration layer

The configuration header contains the number of stallion clock cycles the configuration needs, the number of embedded variables used and a six-bit vector indicating the nature of the stream. A bit denoting an input stream is set to '0' and an output stream bit is set to '1'.

The structure of the embedded variable is similar to the packet structure of the embedded variable in the configuration module packet. There can be a maximum of 60 embedded variables per configuration, hence six bits are used for indexing into the embedded variables.

The embedded variables are followed by the configuration streams. Each Stallion configuration can have a maximum of six different streams corresponding to each data port.

A configuration stream is composed of a stream header followed by the actual programming stream. The programming stream bits are sent to the corresponding data port to configure Stallion. The stream header can be of two different types depending on whether the stream is an input or an output stream. The header indicates the length of the stream (number of packets) and the source of the input to the data port. The input can be I data, Q data or an NEV.

The header is followed by the programming stream bits with one 18 bit word in each memory location. The header of the output stream essentially contains the number of output packets to be expected.

The output stream itself consists of an entry for each output packet expected from Stallion. If the discard flag is set, then output is discarded. The location vector indicates where the output is directed. The output can be written to one or all of the following locations: an NEV, SRI layer, embedded variable or sent back to a data port.

4.3 Implementation of the Processing Layer

The processing layer is responsible for performing all the data processing and other demodulation functions. The processing layer consists of a series of processing elements, possibly of different sizes. Each processing element can be configured during run-time using configuration bit streams. The structure of the processing element was described in detail in Section 3.3.3. The configuration layer, formats the configuration bits as well as the data bits for the processing layer. The formatted stream packets are used directly in the processing elements without any further modifications.

4.4 Conclusions

An algorithm for the implementation of the layered radio architecture described in Chapter 3, is presented here. The algorithm is based on the design approach used for the layered radio architecture. The algorithms demonstrate the feasibility of implementing the layered radio architecture and describes the actual procedures needed for storing a new configuration in the radio, loading an existing configuration, controlling algorithm execution, and the interaction between the three layers. The algorithm describes in detail the messaging structure, organization of the memory, and the setup of the configuration and SRI layers that can be implemented in hardware. The next chapter discusses some key receiver implementations based on the layered radio architecture using Stallion for the processing layer and looks at some performance issues.

Chapter 5

Hardware Implementation of Receiver Architectures

The layered radio architecture is designed to support multimode radios with different transceiver structures that can be programmed and downloaded from memory. The procedures and algorithms discussed in Chapter 4 can be used to implement the architecture on hardware. This chapter describes the hardware implementation and memory management for two different complex receiver structures based on the layered radio architecture. The implementations demonstrate the feasibility of the architecture using practical hardware and indicate the performance of such systems.

The first receiver is an adaptive single user CDMA receiver using Least Mean Squares (LMS) based adaptation. The complex receiver structure performs despreading, tracking as well as equalization. The LMS algorithm tracks the signal and updates the filter coefficients every symbol resulting in an extremely complex and computationally intensive system compared to most practical receivers.

The second receiver, a more practical system, is a W-CDMA rake receiver with channel estimation. The rake receiver is one of the most computationally intensive sections

of a practical CDMA receiver including 3G receivers that are used in existing as well as future wireless systems. The implementation of the W-CDMA rake receiver using 32 bit spreading codes uses just 8.9% of Stallion's processing power demonstrating the feasibility of the system architecture.

In the absence of the Stallion chip during the development of the systems, emulation and simulation tools were used that accurately model the functioning of Stallion. The Stallion simulator described in Section 3.1.2 can be used to implement algorithms on Stallion, debug systems and evaluate performance issues. The Stallion emulator is on a Xilinx Virtex1000 FPGA that is part of the SLAAC1V FPGA board designed by ISI, Inc. Since a complete Stallion could not be accommodated on the Virtex FPGA, only parts of stallion actually in use were emulated on the FPGA. However, the Stallion emulation could be used to verify the hardware interface of the processing layer to the configuration layer to complement the simulator in modeling Stallion.

The receiver implementations examine the processing power of Stallion, memory requirements and tradeoffs between area, power consumption, and processing power. The chapter concludes with the hardware requirements for the layered radio architecture and some suggested improvements for Stallion.

5.1 Implementation of a Single-User Adaptive Receiver Using Stallion

A single user CDMA receiver using complex adaptive filtering is implemented using the layered radio architecture [51]. The processing layer is implemented on Stallion to achieve fast run-time reconfiguration. The implementation demonstrates the feasibility of the layered radio architecture using existing hardware components.

A reconfigurable transmitter, called Completely COnfigurable Transmitter (COCOT),

designed specifically for the soft radio is used to generate transmit signal. The transmit signal is a turbo code encoded, spread spectrum signal with differentially encoded BPSK modulation. The signal is mixed at an Intermediate Frequency (IF) of 68 MHz and then upconverted to an RF frequency of 2.05 GHz. At the receiver, the Rockwell Miniature Radio Codec (MRC) is used to convert the signal to baseband. The baseband signal at the receiver has a residual frequency since the MRC does not use coherent demodulation. To handle the residual frequency, the receiver is designed to use differential decoding. The single user receiver uses a complex adaptive filter based on the Least Mean Squares (LMS) algorithm to perform demodulation and despreading. The differentially decoded signal estimates are then sent to a turbo decoder that gives the final bit estimates.

5.1.1 Receiver Structure

Figure 5.1 shows the functional block diagram of the baseband receiver. The acquisition process uses a matched filter to determine the position of the symbol. This information is used in the tracking loop using the complex LMS filter. Figure 5.2 shows the algorithm used for acquisition. The matched filter is initialized with spreading code of the user. The magnitude of the output of the matched filter is averaged over L symbols. To simplify the hardware implementation and avoid a square root operation, the square of the magnitude is used, in place of the actual magnitude. A maximum search is performed over the averaged output to determine the peak location. If this peak exceeds the preset threshold, the system switches to the tracking loop.

The tracking scheme uses a complex filter that is two symbols long with LMS adaptation [23]. The long filter is an FIR filter used for despreading, tracking, as well as synchronization. The position of the non-zero weights within the long filter depends on the actual position of the symbol. The long filter is initialized to the spreading code of the user. The initial position of the spreading code in filter is decided by the position of the symbol located during acquisition. Once the signal is acquired, the position (and

value) of the weights within the long filter is decided by the LMS adaptation algorithm. The oversampling in the symbol helps compensate the drift in the system clock. The position of the weights are updated using LMS adaptation.

The received signal is demodulated non-coherently by the MRC, which results in a residual frequency on both the I and Q channels. The differential encoding of the I and Q signals helps retain the correct sign of the bits. However, the traditional LMS structure has to be modified to accommodate the differential decoding in the adaptive filter [33]. Correspondingly the weight update equations are modified as

$$y(n) = w^H(n)r(n) \tag{5.1}$$

$$w(n+1) = w(n) + \mu e^*(n)r(n)y^*(n-1) \tag{5.2}$$

where, $y(n)$ corresponds to the output of the filter at each symbol, $r(n)$ is the received signal, $w(n)$ is the weight vector, $e(n)$ is the error vector and μ is the step size. The weights are updated at the symbol rate.

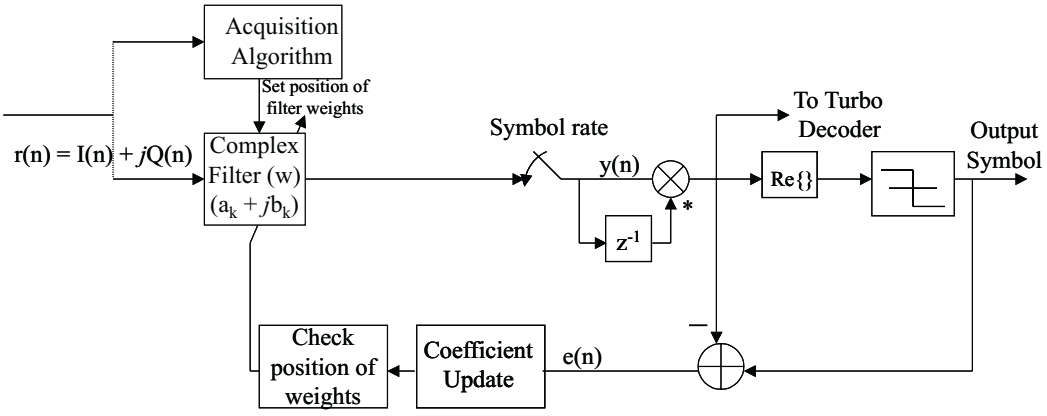


Figure 5.1: Block diagram of the single user adaptive receiver

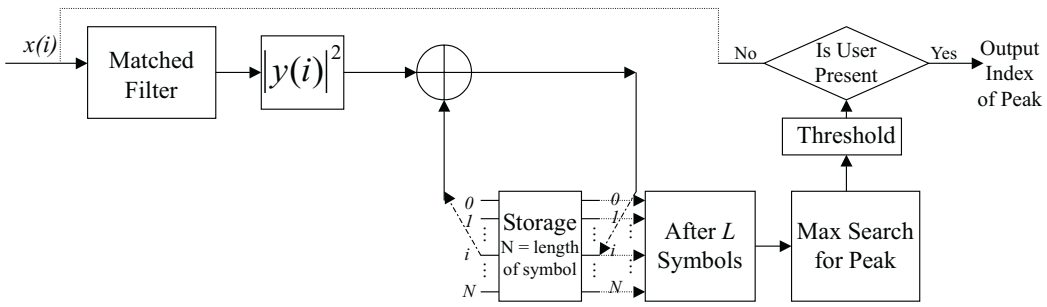


Figure 5.2: Acquisition algorithm for the receiver

The differentially decoded signal estimate is then decoded using a turbo decoder. However, the complex filter is updated using the hard limited filtered signal thus operating in decision directed mode. The tracking loop also needs to ensure that the filter weights do not shift beyond the edge of the increased length filter. In case this happens, the filter weights are reset, and the received signal is adjusted for signal continuity, by varying the number of samples shifted into the filter.

The radio is capable of changing the spreading code, length of the filter as well as the convergence speed through programming. Additionally, the receiver is also able to vary the adaptation algorithms used, thus demonstrating the features of a true soft radio.

The receiver is implemented on the SLAAC FPGA board by ISI, Inc. The PCI based SLAAC board contains three Xilinx Virtex-1000 FPGAs. The Stallion is emulated on the SLAAC board prior to the fabrication of the chip. Figure 5.3 shows the system level design of the single user adaptive receiver. The receiver can accept data either from the MRC or directly from the PC. The figure shows the main components of the SLAAC board.

The SRI and configuration layers are implemented directly on the Xilinx FPGA, while the processing layer is implemented on the emulated Stallion. Various parameters of the radio like the spreading code, filter length, adaptation coefficient for the LMS algorithm,

and number of symbols used for acquisition are variable in the system. This information is passed on to the configuration layer in the form of embedded and non-embedded variables. The number of cycles the acquisition configuration executes for determines the number of symbols used in acquisition. The use of the Stallion allows run time reconfigurability and gives the radio the desired flexibility. Since Stallion lacks internal memory, the Xilinx FPGA containing the SRI and configuration layers are used for storage. However, all the computational tasks are performed by Stallion. The turbo decoder is implemented on the third Xilinx FPGA. The output of the processing layer is sent back to the SRI layer through the configuration layer. The de-packetized output of the SRI layer is then sent to the turbo decoder which generates the final bit estimates.

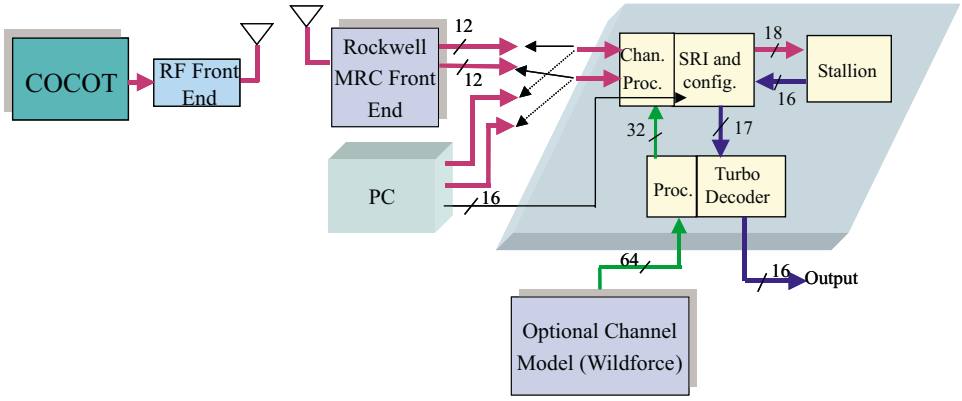


Figure 5.3: GloMo soft radio

The concept of hardware paging is demonstrated as Stallion is configured into the following modes to implement the CDMA receiver: acquire and synchronize, filter, demodulate I and Q, alter μ, calculate error and new weights, update the weights, and locate the position of the weights. The alter μ configuration can be used to dynamically vary the value of μ for each iteration. The number of configuration and processing clock cycles are shown for each configuration in Table 5.1, where n denotes the number of symbols used for acquisition.

Configuration	Configuration Clock Cycles	Processing Clock Cycles
Acquire and synchronize	106	4440*n
Filter	132	132
Calc. of error and new weights	132	132
Flexible	85	132
Demodulate I, Q	104	15
Update new weights	120	20
Locate position of new weights	106	134
Total	785	565+4440*n

Table 5.1: Configuration and processing clock cycles for each symbol. Note n is the number of symbols used in acquisition

The receiver structure demonstrates a very high use of hardware paging in a complex receiver structure. The tracking algorithms updates the filter coefficients once every symbol which is more frequent than most practical systems. Further, the receiver uses only one Stallion and has to frequently page algorithms in and out of the system several times during each symbol. As a result, there is an overuse of hardware paging (due to lack of availability of more Stallions), and the programming clock cycles are comparable to the processing clock cycles. Ideally, the system has to be designed such that the processing clock cycles are much higher than the programming clock cycles so that the vast majority of time is spent in computing and not reprogramming. This receiver structure can be implemented more efficiently with the use of multiple Stallions. Assuming a clock speed of 50MHz, during acquisition, the system can support data rates up to about 11 Kbps, and during tracking, the system can support data rates of 40 Kbps. The clock speed, and mapping area are two parameters that can be controlled by the designer while implementing algorithms on the processing layer. A higher clock speed allows for higher bit rates at the cost of a higher power consumption. Alternatively, for a fixed clock speed, a higher data rate can be obtained by using a larger area and increased parallel processing, thus reducing the number of programming clock cycles for each symbol.

5.2 Implementation of a W-CDMA Rake Receiver with Channel Estimation

The implementation of a W-CDMA rake receiver with channel estimation is geared to demonstrate the ability of the proposed soft radio architecture to support existing and future high data rate systems [52]. The rake receiver is one of the most complex structures in a CDMA receiver that places high demands on the processing power of the system. The implementation shows that the layered radio architecture with the use of CCMs can support robust high data rate systems without trading flexibility for

high-speed processing which frequently occurs when designers use ASICs for high speed designs. The system is implemented using the Stallion simulator for the processing layer while the SRI and configuration layers are implemented in software. The Stallion with 12 data ports is used in this system instead of 6 data ports to demonstrate how the parallel processing features of Stallion can be exploited in the design. Further, addition of these additional data ports to the actual chip is relatively straightforward.

Figure 5.4 describes the structure of the downlink rake receiver with channel estimation. The rake receiver acts on the de-scrambled data of the Dedicated Physical Data Channel (DPDCH) [60]. Length 32, Orthogonal Variable Spreading Factor (OVSF) codes over-sampled by a factor of 2 with 8 pilot bits per slot are used for the system. The three finger downlink rake receiver operating on a time varying channel with Rayleigh fading uses the averaging method [6] for channel estimation. The channel estimates are formed as

$$\widehat{h}_{avg}(n) = \frac{1}{N_p} \sum_{m=0}^{N_p-1} \widehat{h}(m,n) \tag{5.3}$$

where $\widehat{h}(m,n)$, is the instantaneous or rough estimate of the m^{th} symbol in the n^{th} slot, and N_p is the number of pilot bits in the slot. The instantaneous channel estimate vector for each slot is formed as

$$\widehat{h}(m,n) = R(m,n)/pilot\_symbol(m) \tag{5.4}$$

where, $R(m,n)$ is the despread bit estimate, and $pilot\_symbol(m)$ is the value of the pilot known a priori and is either +1 or -1. Once the average channel estimate is computed, the multipath signals are added using maximal ratio combining to generate the symbol estimate $z(n)$ as

$$z(n) \sum_{i=1}^{f} R(n,i) * conj(\widehat{h}_{avg}(i)) \tag{5.5}$$

where, the number of rake fingers f, is set to 3, and $conj(.)$ refers to complex conjugate(.). The receiver is implemented using a single Stallion for the processing layer. The implementation uses four configurations per slot that are consecutively paged into Stallion based on the packets received from the configuration Layer. The four configurations, identified in Figure 5.4 are matched filter, channel estimation, channel compensation, and Equal Gain Combining (EGC). The channel compensation configuration and EGC configuration together perform the function of Maximal Ratio Combining (MRC). The matched filter configuration despreads the signal (using spreading code c_i) for one slot where the delays are tuned to the most significant multipaths. The 32-bit spreading code is stored as a pair of 16-bit constants in two functional units where each bit represents one of the chips in the spreading code. A new chip sample is shifted out every other clock cycle (oversampling factor = 2) and used to despread the received samples. Routing within the configuration layer separates out the pilot bits after despreading for use in the channel estimation configuration to form the channel estimates. The channel estimation configuration operates in a manner similar to the matched filter configuration, storing the training sequence as an 8-bit constant and then averaging over the 8 pilot bits. After the despread channel estimates are formed, the channel compensation configuration multiplies the despread data bits with the complex conjugate of the channel forming the MRC estimates. Finally, the maximal ratio estimates from each finger are added in the EGC configuration to generate the final bit estimates.

5.2.1 Performance of the Receiver

The performance of the W-CDMA receiver gives an indication of some of the performance bounds and issues for high data rate systems. The performance of a soft radio receiver can be analyzed by examining metrics like area, power consumption, and clock speeds

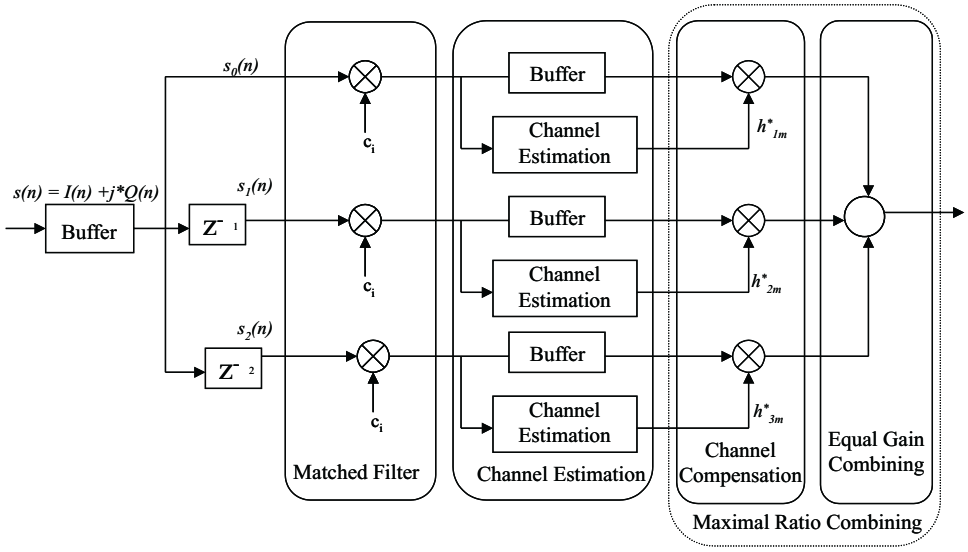


Figure 5.4: Downlink rake receiver with channel estimation

needed to support required data rates without trading flexibility. In addition to the conventional metrics, ease of reconfiguration, reconfiguration overhead, and amount of reconfiguration possible also need to be considered. Though feasibility of a system is dependent on the exact system to be implemented, examining the performance of the W-CDMA receiver gives a good indication of the capabilities of the architecture for addressing real-world problems using existing hardware.

The implementation of the W-CDMA receiver uses four configurations to process data from one slot, where each slot requires a certain amount of overheard programming bits before the data bits can be processed. The W-CDMA receiver has complete reconfigurability since the configuration bits are downloaded for each receiver structure. The minimum clock speed needed to support W-CDMA data rates, required silicon area, and power consumption gives a good indication of the feasibility of the implementation and illustrates design tradeoffs.

Table 5.2 lists the programming and processing cycles needed for each configuration

and the corresponding Stallion utilization in terms of the number of functional units used for each configuration. The bulk of the receiver processing takes place in the despreading operation in the matched filter configuration. The matched filter configuration makes excellent use of Stallion's resources by simultaneously operating on all three fingers' I and Q channels, resulting in an extremely low cycle count for that configuration. The implementation of the 3-finger rake takes 5976 cycles per slot, where each slot contains 160*32*2 or 10240 samples with 5120 chips. This yields an aggregate processing rate of 5976/10240 = 0.5836 cycles per sample. The W-CDMA standard uses a chip rate of 3.84 Mchips/sec. This sets the minimum operating speed of Stallion to (5976/5120)*3.84Mchips/sec = 4.48 MHz for the current configuration. A Stallion implementation of this system operating at a typical speed of 50 MHz will be operating at only 8.96% of the maximum processing rate (which we define as the percentage cycle utilization), demonstrating the architecture's capacity to support very high data rates. It has to be noted that while the minimum processing clock speed is 4.48MHz, data needs to be clocked in at the W-CDMA rate. The reduction in the processing clock speed is due to the parallel implementation of the three rake fingers. Table 5.2 lists the memory requirements for the programming bits. The implementation uses about 18Kbits for programming and 8Kbits for storing data in-between configurations resulting in low memory requirements.

Configuration	Programming Bits	Programming Clock Cycles	Processing Clock Cycles	Fus Used (60 max)
Matched Filter	5632	73	5287	36
Channel Estimation	3008	69	42	22
Channel Compensation	7552	149	228	56
EGC	1536	42	86	10
Total	17728	333	5643	124

Table 5.2: Stallion implementation statistics

Figure 5.5 shows the percentage utilization of a single Stallion implementing the W-CDMA downlink rake receiver with channel estimation for different spreading codes and oversampling factors. There is a slight reduction in percentage utilization in going from spreading codes lengths of 4 to 64 due to the reduction in overhead for each slot. A shift register is used to shift out the spreading code, which needs to be reset more often for shorter length spreading codes, contributing to an increase in overhead operations. There is an abrupt increase in processing cycles for spreading codes of length 128 since different mapping structures are needed for these longer length codes. The spreading code generator and the rake fingers can no longer fit on one Stallion and have to be split across multiple Stallions or configurations.

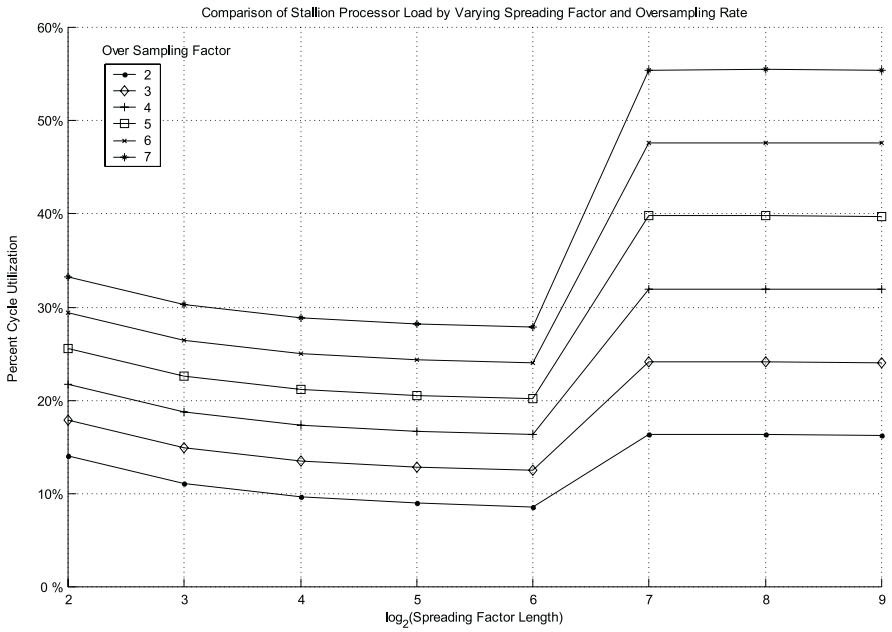


Figure 5.5: Percentage utilization of Stallion for different spreading factors

In the absence of the Stallion chip, power consumption for the chip is estimated using simulation techniques. Power consumption is given by $\frac{1}{2}CV^2f$, and is estimated based

on the capacitance C, the supply voltage V (3.3V), and frequency of bit flips f. Figure 5.6 shows the programming and processing power consumption as a function of time (clock cycles), using this approximation. As can be seen from the figure, the Stallion is idle for most of the slot duration and is active for about 8.9% of the slot duration. The average power consumption during the active period is 2.9 Watts, and the overall power consumption during operation (considering the active and inactive periods) is about 0.25 Watts. The power consumption during programming depend on the number of FUs that need to be programmed and the programming path itself. The matched filter configuration executes for nearly 6000 clock cycles and the periodic nature of the graph in Figure 5.6 corresponds to each symbol being despread. After each symbol is despread, certain initializations are performed which correspond to the drop in power consumption before the start of the next symbol. The power consumption for the other configurations are marked on the figure. As expected, the average power consumption in each configuration is proportional to the number of FUs used for that configuration and can be verified from Table 5.2. The channel compensation configuration that uses 56 FUs has the highest average power consumption.

In the absence of rigorous power estimation and comparison schemes, it is useful to compare the Stallion with a CMOS ASIC radio for W-CDMA built at UC Berkeley, specifically for low power implementation. This radio uses 4.5mW for a three finger rake receiver with a supply voltage of 1.5 Volts or about 22mW for a supply voltage of 3.3Volts [57]. The power consumption of the ASIC CMOS rake implementation with 3.3V is lower than the estimated power consumption of the Stallion rake receiver by a factor of 11. Although Stallion exhibits a higher power consumption for the same rake task, this additional power consumption is a reasonable penalty for the added programmable flexibility. The Stallion circuits wasn't designed explicitly for power consumption and there is much room for improvement in the design of Stallion with the use of low power circuit design methodologies to make it suitable for commercial low power wireless applications and more competitive with an ASIC regarding power consumption.

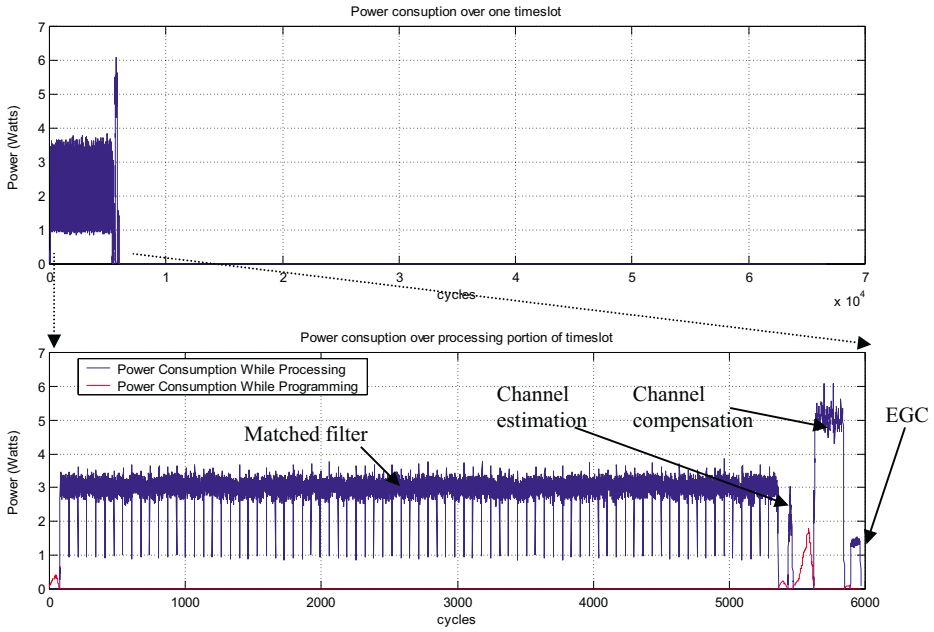


Figure 5.6: Power consumption for the downlink rake receiver

A soft radio based on the layered radio architecture using multiple Stallions can use Stallion's processing power efficiently by implementing hardware paging not just across systems, but within a system as described above in the two receiver structures. The unused portions of the hardware can be turned off to conserve power while the active sections of the hardware can be optimized for efficiency.

The architecture of Stallion is scalable and multiple Stallions can be cascaded simply by connecting the data ports of the Stallions for increasing the processing capacity as needed for complex systems. Only the system in use is programmed in hardware and consumes power, while the other configurations are stored in memory.

There are various tradeoffs in implementing on the processing layer. These include the partitioning of the system across multiple configurations, selection of the number of Stallions (or other processing elements), and choosing the operating clock speed. Using a large number of configurations to process a small number of data bits is not efficient since clock cycles (and power) are wasted in overhead programming. On the other hand using too few configurations results in parts of the system being idle for a long durations, especially when different parts of the systems are operating at different sample rates. For instance, after despreading the sample rate reduces by a factor proportional to the spreading gain. If the system is implemented without any hardware paging, then parts of the system (after despreading) operating on lower sample rates, will remain idle and have a low utilization factor. There is a similar drop in sample rates in systems using decoders with coding gain. For maximum efficiency, all components should be operating at 100% utilization factor, but this is hard to achieve in practice.

We define the paging ratio as the ratio of the number of data bits (or samples) processed to the number of configurations used to complete the processing. The chosen paging ratio determines the area and the minimum clock speed. As a larger area is used, system tends to be "laid out flat", i.e., implemented without the use of any hardware paging, and the clock speed tends to decrease. While lower clock speeds reduce the power consumption

of the individual components, the increased area adds to the power consumption. Thus a good design approach is to choose a paging ratio that maximizes utilization, while minimizing area, and clock speed. The paging ratio metric can be further refined with the knowledge of the configuration bit stream which is specific to each implementation.

The W-CDMA downlink rake receiver uses four configurations for processing 160 bits resulting in a paging ratio of 160/4 = 40 bits per configuration, or 160*32*2/4 = 2560 samples per configuration. The single user CDMA receiver using LMS adaptation uses 7 configurations to process 1 bit, resulting in a paging ratio of 0.1429 bits per configuration or 60/7 = 8.57 samples per configuration. This paging ratio is very low and reflects an inefficient implementation as seen in Section 5.1.1.

Figure 5.7 shows the tradeoff in area and clock speed on % cycle utilization in a 3-dimensional graph. The mappings are done in increments of one quarter Stallion, with 100% indicating a complete Stallion. It can be seen that the percentage utilization decreases with increase in clock speed and area. In this particular receiver architecture, the clock cycles, and hence percentage utilization is mainly dictated by the matched filter configuration. Thus while partitioning algorithms, it is important to focus on the dominating configuration which, if mapped efficiently leads to the most substantial gains in efficiency.

5.3 Hardware Requirements for Implementing the Layered Radio Architecture

The previous sections described the layered radio architecture, the functions of each of the three layers, and some example receiver implementations. The actual processing of the data is performed in the processing layer, while the SRI layer and the configuration layer together aid in configuring, controlling and updating the radio. All three layers use

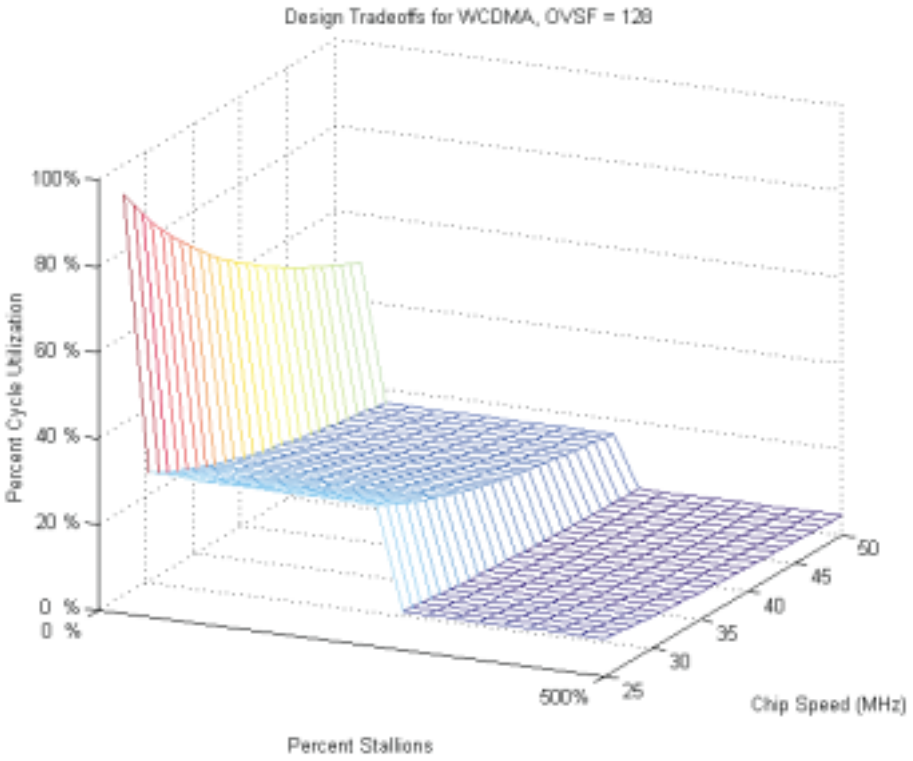


Figure 5.7: Tradeoffs in Stallion for W-CDMA rake receiver implementation

stream based processing. The processing layer needs to be implemented using run-time reconfigurable hardware and one possible hardware platform for the processing layer is the CCM Stallion described in previous chapters.

The functions of the SRI and configuration layer are well defined within the stream based architecture. The functions include loading a new configuration to the radio (possibly through over-the-air updates using software validation), configuring the radio using a stored configuration, controlling the execution of the processing layer, and transmitting processed data back to the higher layers. Implementing such functionalities does not necessitate the use of run-time reconfigurable hardware and these two layers can be implemented using fast fixed point DSPs or micro-controllers. They do,however, have to be capable of fast memory access and should have the ability to access to multiple scalable memory units.

5.3.1 Suggested Improvements for Stallion

The receiver implementation examples using Stallion indicate that the processing power of Stallion is sufficient to handle current and future high data rate systems since the arrangement of the functional units allows for massive parallelization of algorithms. Stallion also supports fast reconfiguration and has relatively low programming overhead. The availability of fast run-time partial (as well as complete) reconfigurability supplemented with high processing power makes the chip useful for soft radio applications.

However, full utilization of Stallion's processing power and fast reconfiguration is limited by the number of I/O ports. In it's present form there are six bidirectional data ports. While Stallion's I/O bandwidth is higher than that of conventional DSPs, increasing the number of data ports would aid in better utilization of the chip's processing power. Increasing the number of dataports, which is equivalent to increasing the width of the stream bus in the processing layer, can substantially speed up processing by enabling

better parallelization of algorithms.

Alternatively, having separate local memory access to functional units reduces the dependency on data ports for bringing intermediate variables into the chip. Each processing element (or Stallion for instance) can be designed to have access to certain memory banks. For instance by having four memory banks, as shown in Figure 5.8, the filter coefficients for the multipliers can be stored in the memory and input to the multipliers, thus freeing the data ports. Similarly, the intermediate results can then be stored in the local memory rather than in the higher layers. This will substantially reduce the number of clock cycles that are inefficiently used for memory storage. A possible architecture for addition of memory is shown in Figure 5.8.

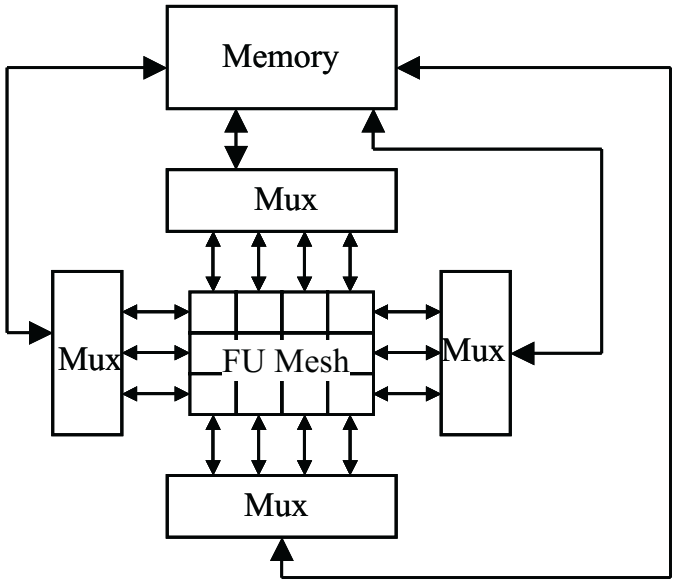


Figure 5.8: Possible approach for adding memory to Stallion

The mappings of receiver algorithms on Stallion, included in Appendices A and B, show that a large number of functional units are being used to route variables to particular functional units, data ports or multipliers. The FU is essentially being "wasted" for routing and its full processing power is not being used. Instead having more skip busses,

for instance, in all four directions avoids their inefficient use for routing purposes, and can allows the FUs to be used for computation intensive operations.

Sign regeneration circuits are needed in the absence of signed multipliers on Stallion, which take up additional FU space. Implementation of multipliers that can handle signed and unsigned number formats will eliminate the FUs to be used for sign regeneration circuits, which can then be used for other computational tasks instead. Finally, the option to power down unused FUs to save power when they are not being used will reduce the overall power consumption. The unused FUs can be powered down by temporarily suspending the clock to those FUs. In addition, other low power design techniques need to be investigated to make the CCM suitable for use in commercial mobile applications.

5.4 Conclusions

The implementation of two receivers and their performance analysis indicates the capabilities of the run-time reconfigurable architecture using the CCM Stallion. While the system is capable of handling existing and future high data rate systems, design of low power CCMs with large I/O bandwidth is desirable for soft radio implementations. The layered radio architecture has many tradeoffs, particularly in the implementation of the processing layer. The tradeoffs include silicon area, clock speed, percentage utilization and power consumption. It is especially important to balance the paging ratio with the percentage utilization to create an overall low cost system.

Chapter 6

Conclusions

This dissertation began with a review of key research issues in software radios. This review pointed to the need for resources such as reconfigurable computing devices that offers flexibility and performance tradeoff between a DSP, microprocessor and an ASIC. However, the current design approaches to using FPGAs are based on ad-hoc designs and lack a systematic design procedure which stifles their widespread use across platforms and standards. Further, existing architectures that use configurable computing support only subsets of fixed systems and features and are not truly flexible radios. After examining various architectures and methodologies it was found that a run-time reconfigurable platform using stream based processing provided an architecture that supports all the desired features of a software radios while keeping the architecture simple, scalable, and maximizing the utilization of the radio hardware. Having identified the design issues and the design methodology, a three layered hardware architecture with the Soft Radio Interface (SRI) layer, the configuration layer and the processing layer, is developed using custom computing machines that enables hardware paging. Hardware paging aids in maximizing performance using minimum hardware. Desired sub-systems are split into two levels of abstraction and stored in memory for downloading to the processing layer when the sub-systems are needed.

The layered architecture presents an open yet formal and unified structure for implementing soft radios on reconfigurable platforms. The layered approach lends itself easily to standardization of soft radio systems and also allows code reuse. The processing modules and header information can also be easily standardized, allowing support from a variety of vendors. One of the main advantages of the structure of the processing modules is that the interface between two modules is very simple. Each module is able to accept a stream packet and send out a stream packet in the forward and reverse directions in each cycle. The layered radio architecture is a flexible, scalable soft radio architecture that can support all the desirable features of a soft radio at the cost of additional programming overhead and slightly higher clock speeds to accommodate the overhead information. However, the disadvantage of higher clock rates can be overcome with the use of massive parallel structures that can reduce clock speeds.

The architecture describes in detail the functionalities of each of the three layers followed by algorithms for their implementation in existing hardware platforms. The architecture is then validated with the development of two receiver structures. The CCM Stallion, fabricated by Virginia Tech's configurable computing group, is used for the processing layer to implement hardware paging. The implementation of the W-CDMA receiver demonstrates that the architecture with the use of Stallion can support existing high data rate systems. The dissertation also contains a discussion of the design techniques needed to implement such systems using hardware paging to maximize performance. While the individual implementations of each system on the layered radio architecture may not be optimal compared to a "Velcro" ASIC approach, as the number of systems increase, the layered radio approach becomes a lower cost solution. Partitioning the subsystems by selecting the appropriate paging ratio results in maximum silicon efficiency and provides overall performance.

The report also describes the hardware requirements for implementing the layered radio architecture and outlines design characteristics that can improve the utility of Stallion. While the basic structure of Stallion is useful for soft radio applications, it can be

improved by providing additional signed multipliers, skip busses, data ports and memory and reducing the number of shift registers. The power consumption can also be improved using suggestions described in the previous chapter.

6.1 Scope for Future Research

The development of an RF front end to support software radios and the design of the interface between the RF hardware and the baseband sections of the radio is a challenging task. The digital baseband section of the radio can be designed to compensate for poor dynamic range with the use of smart A/D converters and predistortion algorithms that can be implemented in reconfigurable hardware. The design of a smart interface and algorithms that control the interaction between the two sections of a radio is an important research area for software radios.

The layered radio architecture addresses most of the baseband issues in the development of a soft radio. However, the development of a flexible RF front end and/or very high speed A/D converters is crucial to the implementation of soft radios. In the absence of very high speed A/D converters that can digitize signals at the antenna and digital hardware that can operate at such high speeds, the focus is on designing flexible RF front ends. While there is no solution so far, there have been many attempts at designing such front ends and is still an open research issue.

The layered radio architecture uses CCMs in the processing layer because of their ability to support run-time reconfiguration to enable hardware paging. This dissertation uses a CCM called Stallion to validate and demonstrate the architecture. It is however possible to design efficient CCMs specifically tailored for wireless communication applications, that have an optimum mix of the basic computational elements and use low power design techniques. This would involve detailed profiling of algorithms as well as development of CCM architectures with distributed resources and flexible interconnects as mentioned in

Section 5.3.1. Design of CCMs using multiple DSP cores with flexible interconnects can also provide good hardware platforms for the processing layer. I/O bandwidth is a major challenge for DSPs. The Stallion architecture allows for expansion of I/O data ports to increase I/O bandwidth. In contrast, expanding the I/O bandwidth in conventional DSPs may involve major changes to the architecture. Expanding Stallion's capabilities based on the suggestions in Section 5.3.1 would be a useful research direction.

This report presents a scheme where the processing modules are connected in a sequential manner. Connecting the modules in a sequential manner is a first step in designing soft radios, but a more sophisticated and powerful approach is to network these modules in the processing layer. For example, creating a mesh of processing modules with 4-connectivity, can increase the processing power tremendously. This of course opens a new area of research where the stream packets no longer travel in just one (or two) directions. This design approach would have to consider routing aspects, packet collision and prevention, and packet ordering, which are the same issues encountered in packet radio communication systems. However, this change can be viewed as just an extension of the layered architecture. A new layer, a *network layer*, can be inserted between the configuration layer and the processing layer that takes care of the packet routing and collating of information obtained from the processing layer. Other aspects of the architecture can remain unchanged.

Bibliography

[1] Peter M. Athanas, J. H. Reed and W. H. Tranter, "A prototype software radio based on configurable computing," *Advancing Microelectronics*, pp. 34-39, 1998.

[2] Peter M. Athanas, I. Howitt, T. Rappaport, J. H. Reed and B. Woerner, "A high capacity adaptive wireless receiver implemented with a reconfigurable computer architecture," *ARPA GloMo Principle Investigators Conference*, San Diego, CA, November 1995.

[3] D. Bhatia, "Reconfigurable Computing," *Proceedings of the* 10^{th} *International Conference on VLSI Design*, pp. 356-359, 1997.

[4] Ray Bittner, "Wormhole run-time reconfiguration: Conceptualization and VLSI design of a high performance computing system," Ph.D. Dissertation, Department of Electrical and Computer Engineering, Virginia Tech, 1997.

[5] Vanu Bose, Michael Ismert, Matt Welborn, and John Guttag, "Virtual Radios," *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 4, April 1999.

[6] Mike Butler, James Providakes, Gary Blythe, "The layered radio," *MILCOM* 98.

[7] J. K. Cavers and M. Liao, "Adaptive compensation for imbalance and adaptive losses in direct conversion tranceivers," *Proceedings of the* 41^{st} *IEEE VTC*, pp. 578-83, 1991.

[8] *http://www.chameleonsystems.com/newspg.htm*, Web site of Chameleon Systems Inc.

[9] DeCegama, Angel L., "Parallel processing architectures and VLSI Hardware," Englewood Cliffs, NJ, Prentice Hall, vol. 1, pp. 157-160.

[10] Peter G. Cook and Wayne Bonser, "Architectural overview of the SPEAKeasy system," *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 4, April 1999.

[11] Mark Cummings, Shinichiro Haruyama, "FPGA in the software radio," *IEEE Communications Magazine*, pp.108-112, February 1999.

[12] Jonathan P. Cummings, "Software radios for airborne platforms," *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 4, pp.732-747, April 1999.

[13] *http://www.darpa.mil/ito/research/acs/index.html*, Darpa's reconfigurable computing web-site.

[14] Chris Dick, "High performance communications using FPGAs," *International Symposium on Advanced Radio Technologies*, Chapter 11, Sept. 1998.

[15] Chris Dick, Fredric J. Harris, "Configurable logic for digital communications: Some signal processing perspectives," *IEEE Communications Magazine*, pp. 107-111, August 1999.

[16] Vijay K. Garg, P.E. Samuel Halpern, Kenneth F. Smolik, "Third Generation Mobile Systems," *IEEE ICPWC*, pp. 39-43, 1999.

[17] George, V., Hui Zhang, Rabaey, J. k, "The design of a low energy FPGA," *Proceedings of International Symposium on Low Power Electronics and Design*, pp. 188-193, 1999.

[18] S.A. Guccione and M.J. Gonzalez, "Classification and performance of reconfigurable architectures," *Proceedings of the* 5*th International Workshop on Field Programmable Logic and Applications*, pp. 439-447, 1995.

[19] Michael S. Gudaitis, Richard D. Hinman, "Tactical software radio concept," *Proceedings MILCOM*, Vol. 3, 1997.

[20] James E. Gunn, Kenneth S. Barron, and William Ruczczyk, "A low power DSP core-based software radio architecture," *IEEE Journal on Selected Areas in Communication*, Vol. 17, No. 4, April 1999.

[21] Scott Hauck, "The roles of FPGA's in reprogrammable systems," *Proceedings of IEEE*, Vol. 86, no. 4, pp. 613-238, April, 1998.

[22] Heiko Erben, Kyriacos Sabatakakis, "Advanced software radio architectures for 3^{rd} generation mobile systems," *VTC*, pp. 825-829, 1998.

[23] Michael Hosemann, Investigation of synchronization techniques for direct-sequence spread-spectrum signals and implementation on the layered soft radio architecture using reconfigurable hardware, Internal report, Mobile and Portable Radio Research Group, Department of Electrical and Computer Engineering, Virginia Tech, 1999.

[24] Chenhong Huang, "An Analysis of CDMA 3G Wireless Communications Standards," *Proceedings of the* 49^{th} *IEEE Vehicular Technology Conference*, Vol. 1, Page(s): 342 -345 vol.1, 1999.

[25] B.L. Hutchings and M.J. Wirthlin, "Implementation approaches for reconfigurable logic applications," *Proceedings of the* 5^{th} *International Workshop on Field Programmable Logic and Applications*, pp. 419-428, 1995.

[26] *EIA/TIA IS-136 Cellular Standard.*

[27] Andy Ivers, Dave Smith, "A practical approach to the implementation of multiple radio configurations utilizing reconfigurable hardware and software building blocks," *Proceedings, MILCOM,* vol. 3, pp. 1327-1332, 1997.

[28] Stamatis Kourtis, Pat McAndrew, Phil Tottle, "Software Radio 2G and 3G Inner Receiver Processing," *Proceeding, IEE Colloquium on UMTS Terminals and Software Radio* pp. 6/1-6/7, London, UK., 1999.

[29] Uming Ko, Mike McMahan, Edgar Auslander, "DSP for the Third Generation Wireless Communications," *Proceedings IEEE International Conference on Computer Design: VLSI in Computers and Processors* pp. 516-520, 1999.

[30] Koushanfar, F., Prabhu, V., Potkonjak, M., Rabaey, J.M., "Processors for mobile applications," *Proceedings IEEE International Conference on Computer Design* pp. 603-608, 2000.

[31] Raymond J. Lackey and Donald W. Upmal, "Speakeasy: The military software radio," *IEEE Communication Magazine*, pp. 56-61, May 1995.

[32] Pentti Leppanen, Jaakko Reinila, Asko Nykanen, Visa Tapio et. al., "Software radio-An alternative for the future in wireless personal and multimedia communications," *IEEE International Conference on Personal Wireless Communications*, Jaipur, India, pp. 364-368, February 1999.

[33] Nitin Mangalvedhe, Development and analysis of adaptive interference rejection techniques for direct sequence code division multiple access systems, Ph.D. Dissertation, Virginia Tech, 1999.

[34] J. Mitola III, "Software radios survey, critical evaluation and future directions," *IEEE AES Systems Magazine*, pp. 25-35, April 1993.

[35] Joe Mitola, "The software radio architecture," *IEEE Communication Magazine*, pp. 26-38, May 1995.

[36] Joseph Mitola III, " Software radio technology and applications to law enforcement," *Proceedings SPIE, Command, Control, Communications, Intelligence Systems for Law Enforcement*, vol. 2938, pp. 239-249, Nov 1996.

[37] Joe Mitola, "Software radio technology challenges and opportunities," *First European Workshop on Software Radios*, Keynote address, May 1997.

[38] Joseph Mitola, III, "Software radio architecture: A mathematical perspective," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 4, April 1999.

[39] Joseph Mitola, III, "Technical challenges in the globalization of software radio," *IEEE Communications Magazine*, February 1999.

[40] *http://www.morphics.com/html/tech.html*, Web site Morphics Inc.

[41] Richard E. Nance, James D. Arthur, "The establishment and management of a software quality assessment and prediction program," Technical Report SRC-97-001, Systems Research Center and Department of Computer Science, Virginia June 1997.

[42] Shigeaki Ogose, "Application of Software Radio to the Third Generation Mobile Telecommunications," *Proceedings of the 1999 IEEE 49th Vehicular Technology Conference*, Houston, TX, Vol. 2, pp 1212-1216, 1999.

[43] John V. Oldfield, Richard C. Dorf, *Field programmable gate arrays*, John Wiley & Sons Inc., NY, 1995.

[44] John J. Patti, Robert M. Husnay, and Joseph Pintar, "A smart software radio: Concept development and demonstration," *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 4, pp. 631-649, April 1999.

[45] *http://bwrc.eecs.berkeley.edu/Research/Configurable_Architectures*, Web site of University of California Berkeley.

[46] Javad Razavilar, Farrokh Rashid-Farrokhi, and K. J. Ray Liu, "Software radio architecture with smart antennas: A tutorial on algorithms and complexity," *IEEE Journal on Selected Areas in Communications* Vol. 17, No. 4, pp. 662-676, April 1999.

[47] *http://www.siriuscomm.com/*, Web site Sirius Communications.

[48] Srikathyayani Srikanteswara, Peter M. Athanas, Jeffrey H. Reed and William H. Tranter, "Configurable computing for communication systems," *Proceedings of the Wireless Communications Conference*, IMAPS, pp. 180-185, 1998.

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

[50] Srikathyayani Srikanteswara, Michael Hosemann, Jeffrey H Reed, Peter M Athanas, "Design and Implementation of a Completely Reconfigurable Soft Radio," *IEEE Radio and Wireless Conference, RAWCON2000*, Denver, Sept. 10 - 13, 2000.

[51] Srikathyayani Srikanteswara, Jeffrey H Reed, Peter M Athanas, "Implementation of a Reconfigurable Soft Radio Using the Layered Radio Architecture," *Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, October 29 - November 1, 2000.

[52] Srikathyayani Srikanteswara, James Neel, J. H. Reed, P. M. Athanas, "Designing Soft Radios for High Data Rate Systems and Integrated Global Services," *Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, November, 2001.

[53] Steele, R., "Beyond 3G," *Proceedings. 2000 International Zurich Seminar on Broadband Communications*, pp. 1-7 2000.

[54] Cai Tao, Song junde, Wang hai, "A new architecture of third generation mobile system based on software radio," *International Conference on Communication Technology*, pp. S27-06-1 - 5, October 22-24, 1998.

[55] Hiroshi Tsurumi and Yasuo Suzuki, "Broadband RF stage architecture for software-defined radio in handheld terminal applications," *IEEE Communications Magazine*, pp. 90-95, February 1999.

[56] Thierry Turletti, Hans J. Bentzen, and David Tennenhouse, "Toward the software realization of a GSM base station," *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 4, pp. 603-612, April 1999.

[57] Web site of UC Berkeley, *http://infopad.eecs.berkeley.edu/infopad-ftp/papers/1994/cmos_cdma_radio.wireless94*,

[58] John Villasenor, and Brad Hutchings, "The Flexibility of Configurable Computing," *IEEE Signal Processing Magazine*, Vol. 15, No. 5, pp. 67-84, September 1998.

[59] Ji-Bing Wang, Ming Zhao, Xi-Bin Xu and Yan Yao, "Research on hardware platform for software radio," *International Conference on Communication Technology*, pp. S27-04-1 - 5, October 22-24, 1998.

[60] Third Generation Partnership Project Technical Specification Group Radio Access Network Working Group 1, "Physical Channels and Mapping of Transport Channels onto Physical Channels (FDD)," TS 25.211 V2.2.1 (1999-08)

[61] Li Weidong, Yao yan, "Software radio: Technology and implementation," *International Conference on Communication Technology*, pp. S27-01-1 - 5, October 22-24, 1998.

[62] Xilinx web pages, *www.xilinx.com*,

[63] Kambi C. Zangi and R. David Koilpillai, "Software radio issues in cellular base stations," *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 4, pp. 561-573, April 1999.

Appendix A

Stallion Configurations for the Single User CDMA Receiver

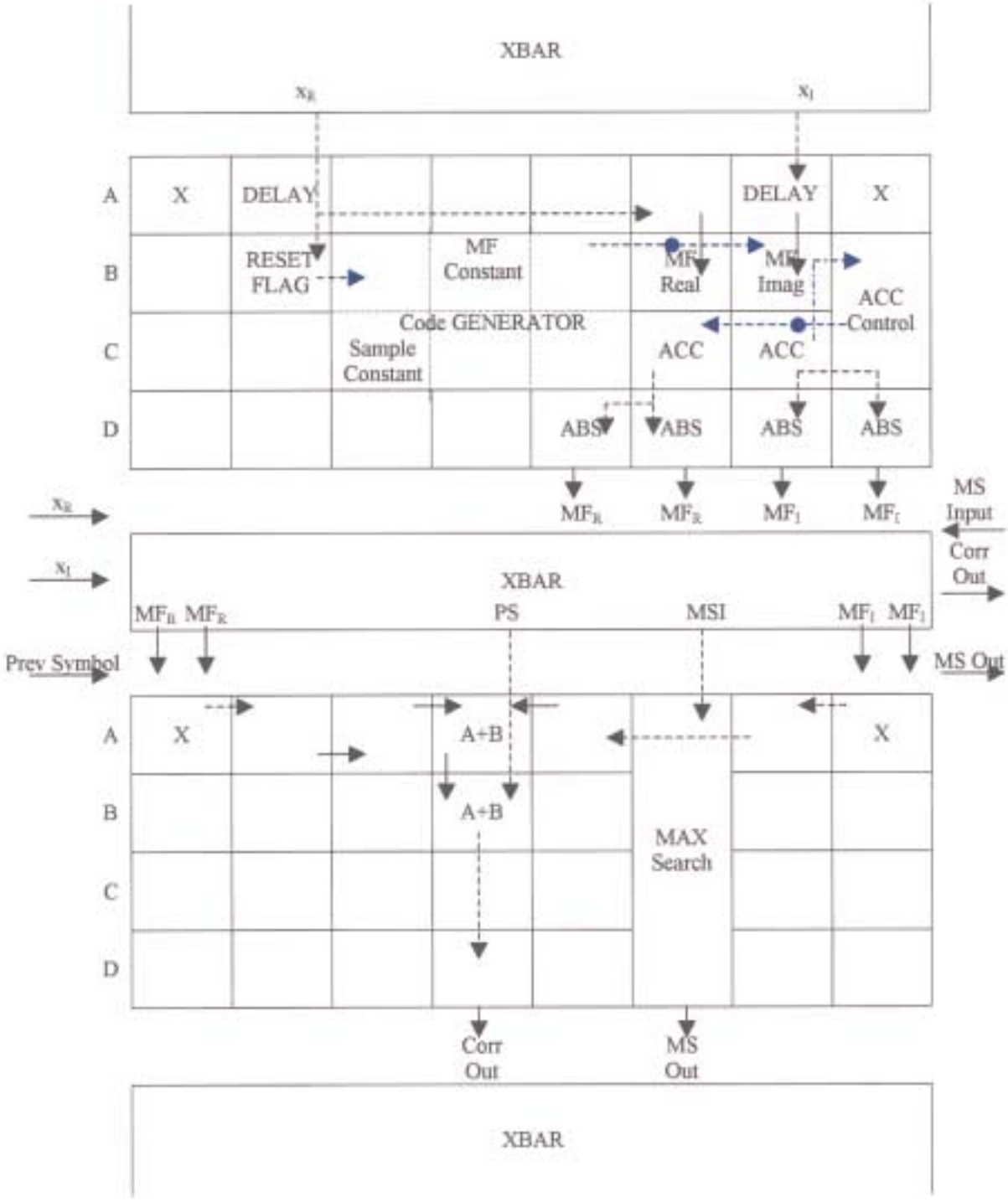


Figure 1.1: Acquisition

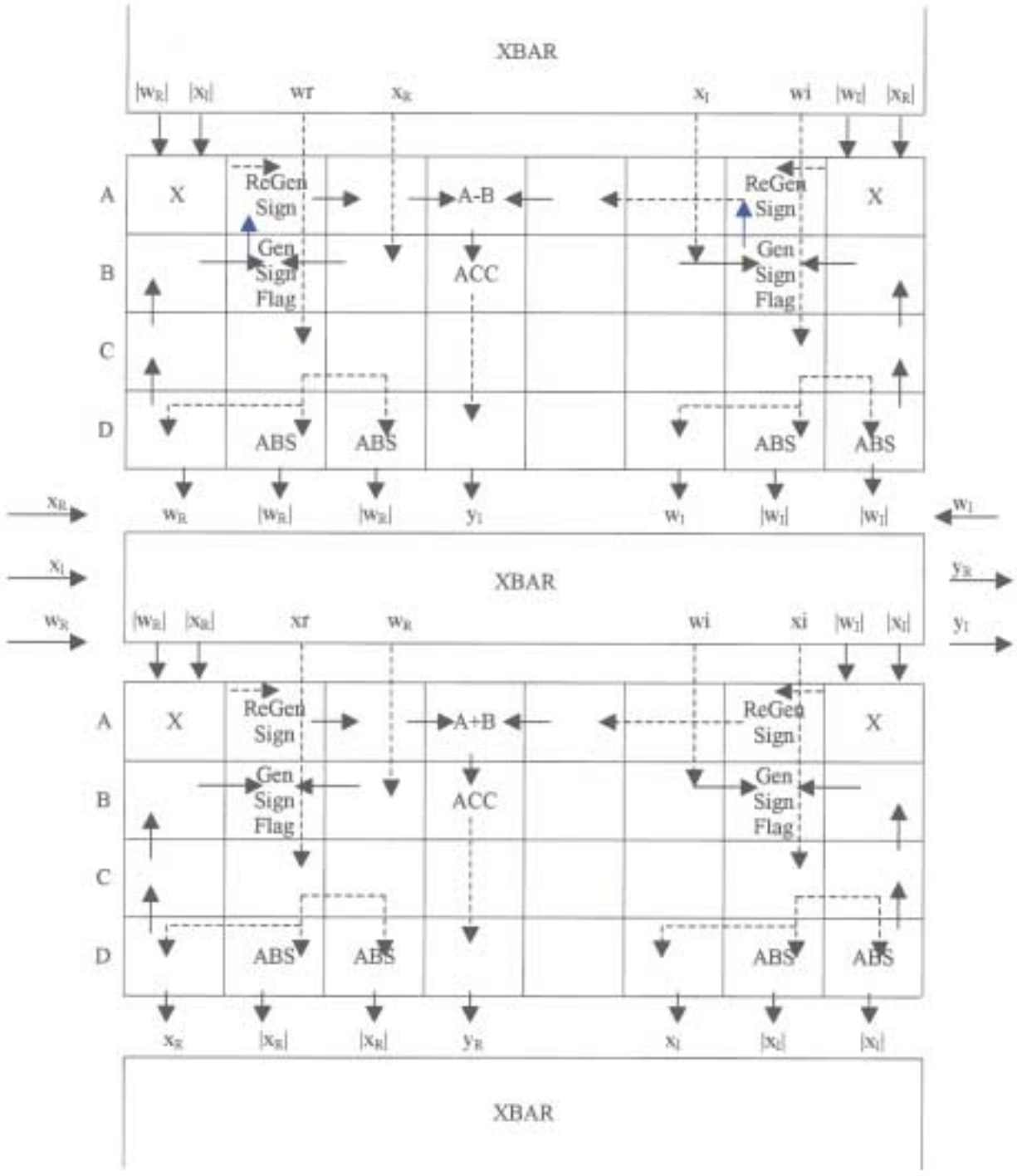


Figure 1.2: FIR filtering

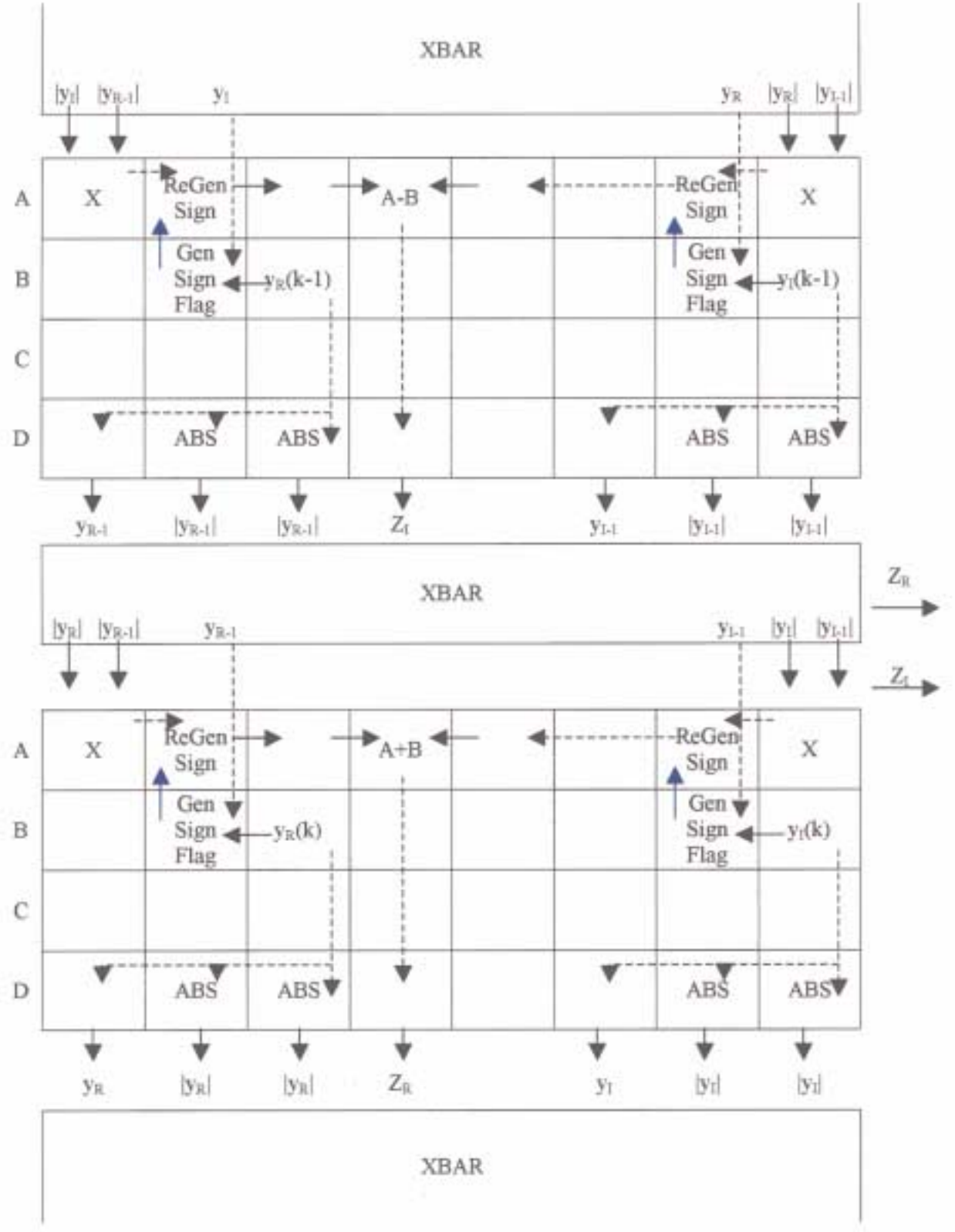


Figure 1.3: Complex demodulation

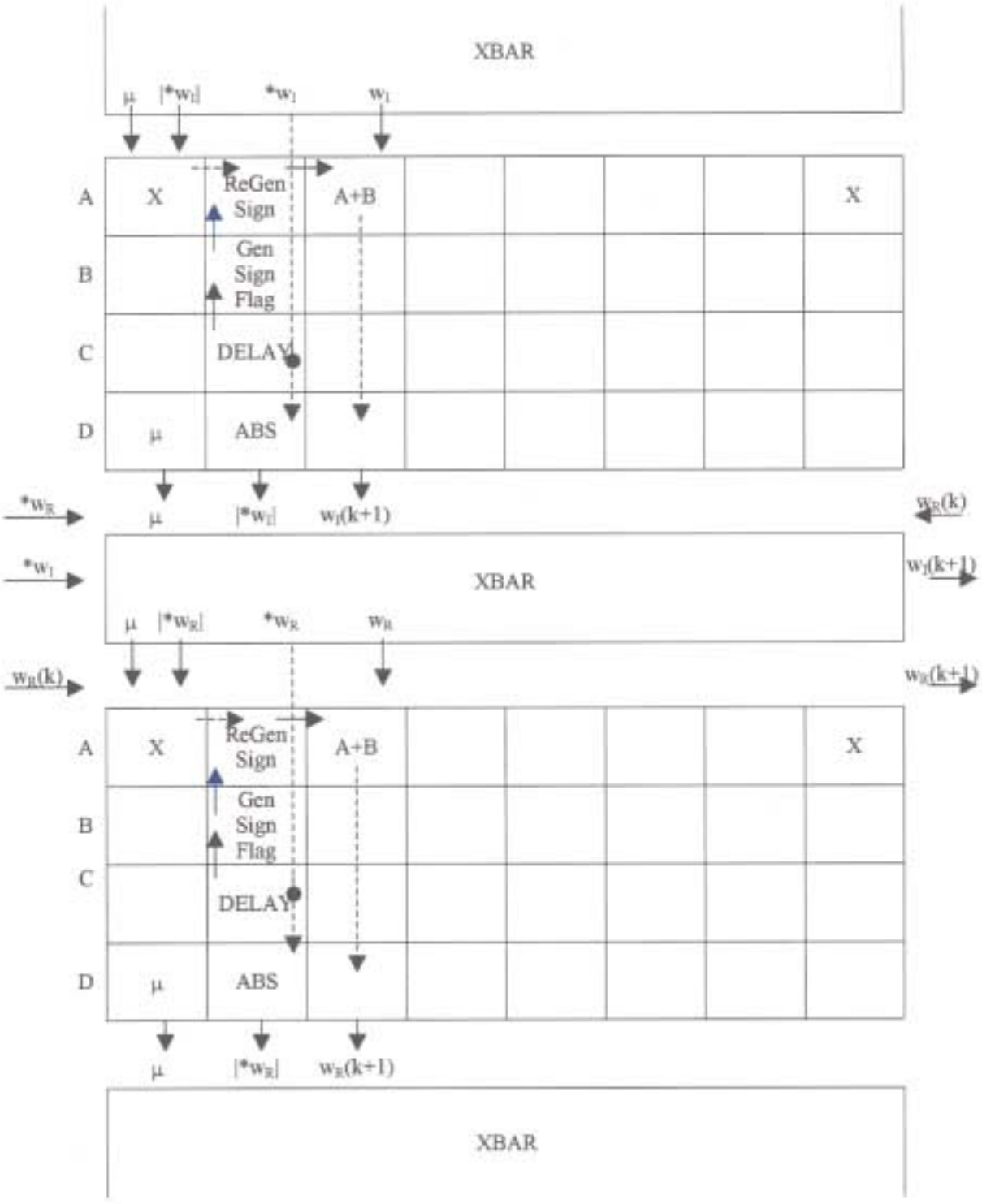


Figure 1.4: Alter μ

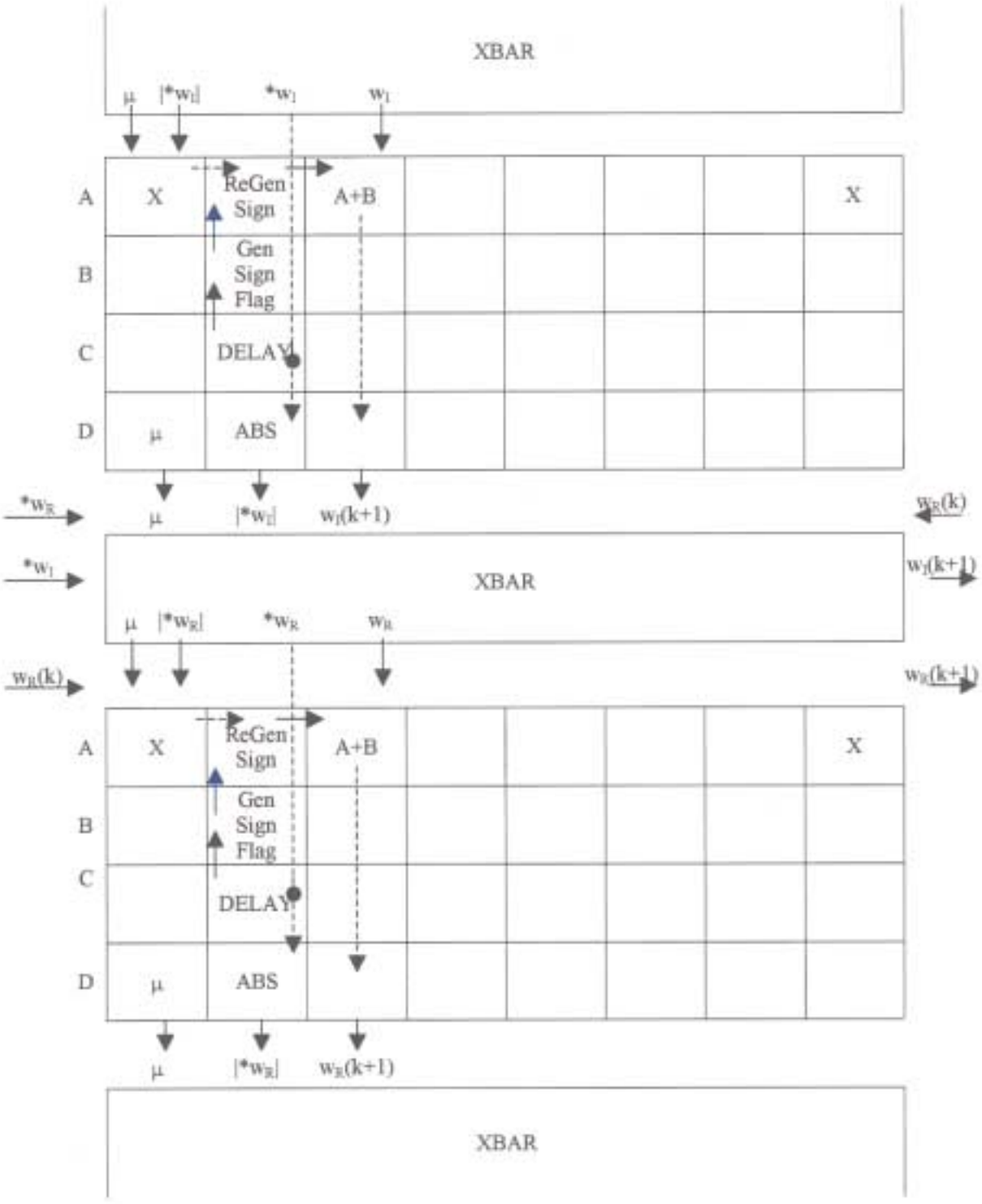


Figure 1.5: Error computation

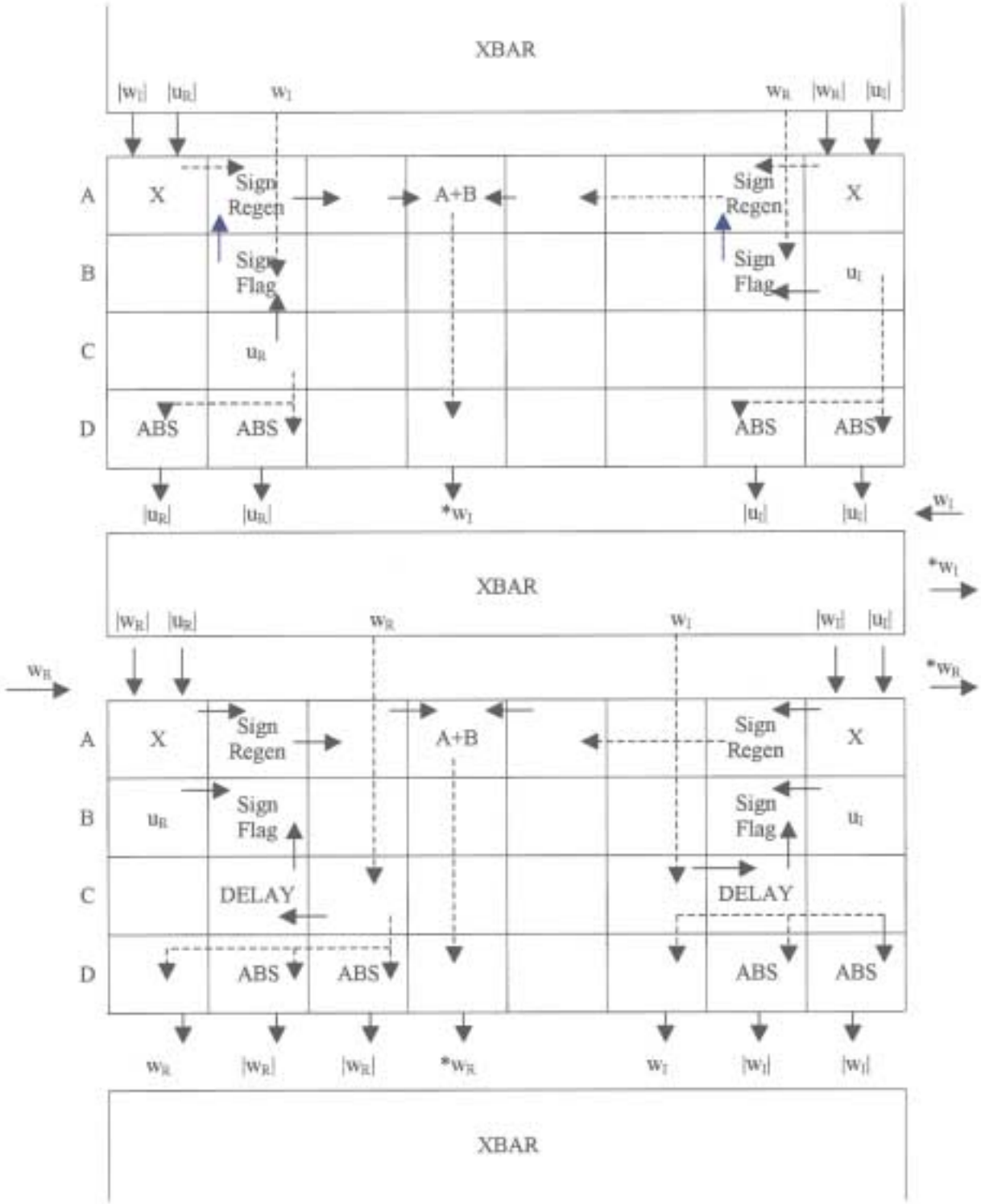
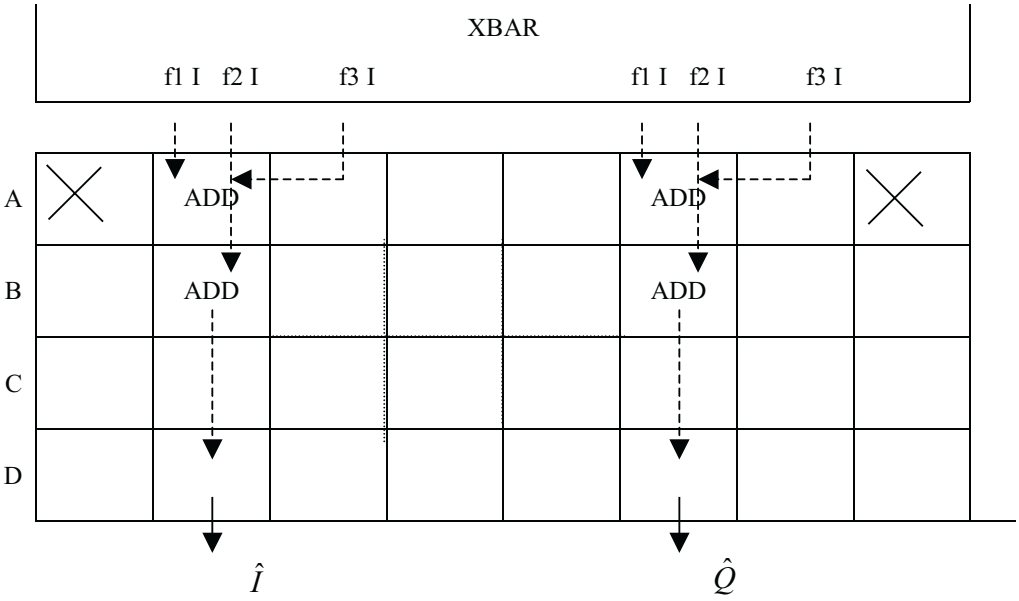


Figure 1.6: Computation of new weights

Appendix B

Stallion Configurations for the W-CDMA Receiver

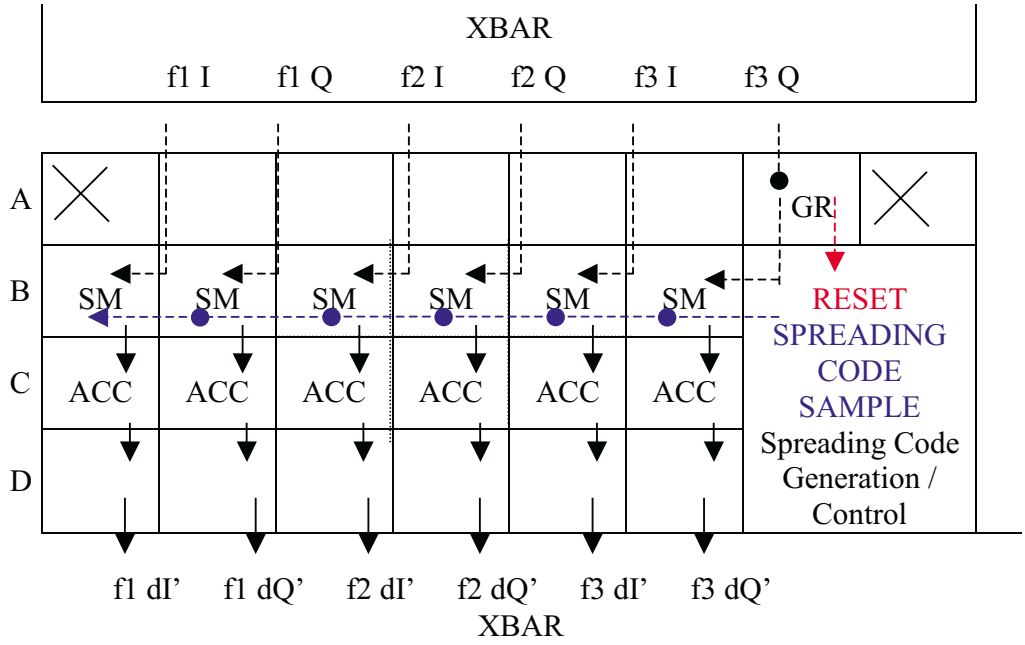


Note: Entire bottom mesh of Stallion is not used, f2 I is delayed by one sample.

LEGEND

⟶	Data (16 + 1 Valid Bit) over Local Bus Clocked Transmission
⇢	Data (16 + 1 Valid Bit) over Skip Bus Instantaneous Transmission
●⇢	Point on Skip Bus indicates information being passed on Skip Bus is used in current Functional Unit
×	Integer Multiplier Unused in this configuration
ADD	ADD Add together the contents of the left and right registers

Figure 2.1: Maximal ratio combining



Note: Entire bottom mesh of Stallion is not used. Performs despreading operation. Spreading code can be set at time of programming by altering stored spreading constants

LEGEND

→	Data (16 + 1 Valid Bit) over Local Bus Clocked Transmission
⇢	Data (16 + 1 Valid Bit) over Skip Bus Instantaneous Transmission
⇢	Flag (1 bit) over F1 Skip Bus Instantaneous Transmission
⇢	Flag (1 bit) over F2 Skip Bus Instantaneous Transmission
●⇢	Point on Skip Bus indicates information being passed on Skip Bus is used in current Functional Unit
×	Integer Multiplier Unused in this configuration
SM	Signed Multiply Effectively Multiplies Input Data by Flag 1 Bit Generates the 2's complement negative of the input data. Selects negated or unaltered data as output based on value in flag1.
ACC	ACCumulate Accummulates input data. Clears accumulate register when an invalid data word is received in its operand register.
GR	Generate Reset Goes high if invalid data is input
Spreading	Generates spreading code samples oversampled by a factor of

Figure 2.2: Matched filtering

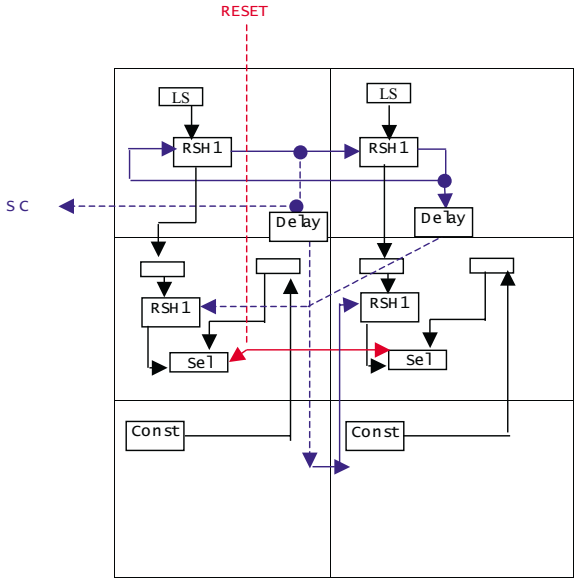
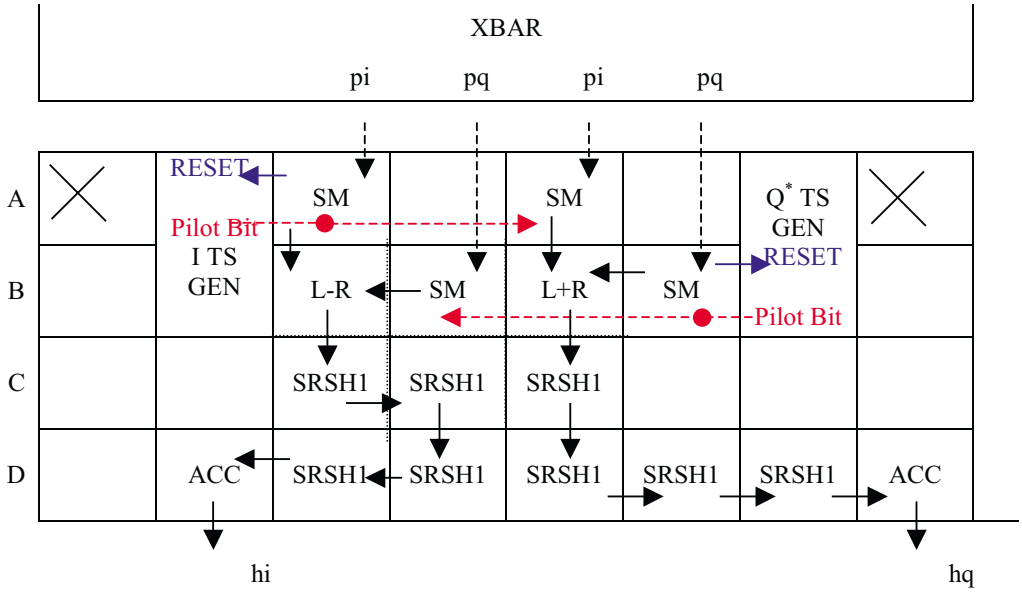


Figure 2.3: Generation of the spreading code



XBAR

Note: Entire bottom mesh of Stallion is not used, Correlates Received Pilot bits with pilot spreading sequence stored in constants in B2 and A7.

LEGEND

→	Data (16 + 1 Valid Bit) over Local Bus Clocked Transmission
⇢	Data (16 + 1 Valid Bit) over Skip Bus Instantaneous Transmission
→	Flag (1 bit) over F1 Skip Bus Instantaneous Transmission
⇢	Flag (1 bit) over F2 Skip Bus Instantaneous Transmission
●⇢	Point on Skip Bus indicates information being passed on Skip Bus is used in current Functional Unit
×	Integer Multiplier Unused in this configuration
SM	Signed Multiply Effectively Multiplies Input Data by Flag 1 Bit Generates the 2's complement negative of the input data. Selects negated or unaltered data as output based on value in flag1.
ACC	ACCumulate Accummulates input data. Clears accumulate register when an invalid data word is received in its operand register.
RSH1	Signed Right SHift 1 Shifts data in Left Register Right by 1. Maintains Sign. Equivalent to a divide by 2.
I TS GEN	I Training Sequence GENerator Generates I portion of Training Sequence for Pilot Bits. 8 bits are stored in a constant and sequentially shifted out. Resets off of reset line generated by invalid data
Q TS GEN	Q Training Sequence GENerator Generates Q portion of Training Sequence for Pilot Bits. 8 bits are stored in a constant and sequentially shifted out.

Figure 2.4: Channel estimation

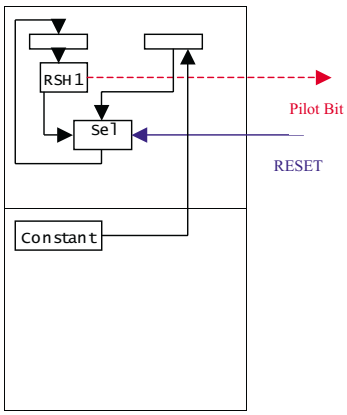


Figure 2.5: Generation of the pilot sequence

Vita

Srikathyayani Srikanteswara received her undergraduate degree in Electrical Engineering, in May, 1994, from the Institute of Technology, BHU, Varanasi, India, where she graduated with honors. During the summers of 92 and 93, she worked interned at the Indian Institute of Science, Bangalore, working in signal and image processing, which also resulted in publications. She was also selected to represent the state of Karnataka at the national level of the Mathematics Olympiad in India. On completing her undergraduate education, she pursued her master's degree at Virginia Tech, in the field of signal and image processing, in the Department of Electrical and Computer Engineering. During the course of her master's program, she developed and implemented image processing algorithms for a color sorting system that won the Challenger Award at the Intl. Wood Working Fair and Exhibition at Atlanta, GA, in August 1996. She received her MS in 1997.

Srikathyayani then joined the Mobile and Portable Radio Research Group (MPRG), at Virginia Tech to pursue a Ph.D. in wireless communications, and work on the topic of software radios. As a Ph.D. student she managed a substantial portion of MPRG's Configurable and Robust Wireless Communication Nodes (GloMo II) project sponsored by (DARPA) and guided MS students' thesis work. She also assisted in writing a software radio textbook with Dr. Jeffrey Reed, and has taught communication courses at Virginia tech. She is now continues her work as a research professor at MPRG. Her research interests include wireless communication systems, software radios, space-time processing, DSP and hardware design for communication systems.