2007-06-19

Performance of MIMO Space-Time Coding Algorithms on a Parallel DSP Test Platform

Beau C. Neal
Brigham Young University - Provo

PERFORMANCE OF MIMO SPACE-TIME CODING ALGORITHMS ON A PARALLEL DSP TEST PLATFORM

by

Beau C. Neal

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

Brigham Young University

August 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Beau C. Neal

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	James K. Archibald, Chair
Date	Brian D. Jeffs
Date	Doran K. Wilde

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Beau C. Neal in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

James K. Archibald
Chair, Graduate Committee

Accepted for the Department

Michael A. Jensen
Chair

Accepted for the College

Alan R. Parkinson
Dean, Ira A. Fulton College of
Engineering and Technology

ABSTRACT

PERFORMANCE OF MIMO SPACE-TIME CODING

ALGORITHMS ON A PARALLEL DSP

TEST PLATFORM

Beau C. Neal

Department of Electrical and Computer Engineering

Master of Science

Commercial Off The Shelf (COTS) hardware has the advantages of low cost, modularity, and is easily upgraded. For Multiple-Input Multiple-Output (MIMO) space-time algorithms to be practical they must have the processing capability to execute in real-time. This makes COTS ideal for real-time MIMO research where the processing power increases exponentially with a linear increase in antennas. The BYU Electrical Engineering wireless lab has designed and built an eight processor transmitter and a twenty processor receiver to research and develop MIMO wireless communication.

The Alamouti, 2×2 and 4×4 differential space-time MIMO algorithms have been partially implemented on the receiver using a variety of common parallel processing topologies to include: bus, line/ring, star, grid, hypercube, binary tree, and pyramid. Processor and inter-processor communication benchmarks were measured and used to quickly explore the performance of the previously mentioned topologies

without expending time and effort on a full implementation of these MIMO algorithms using each topology. This methodology has the benefit of the creation of software libraries that can be used for testing or for complete MIMO algorithm implementation in the future.

This thesis shows that a simple bus-based topology gives the best results when combined with the 4×4 differential space-time algorithm. This thesis also shows that if the number of receiving channels and processors increase at the same rate as the 2×2 to the 4×4 differential cases, then the ratio of decoding processing time to inter-processor communication time is reduced. If this trend continues, inter-processor communication will require more processing time than the actual space-time decoding algorithm.

Due to the exponential increase in required processing, doubling the processing requirements obtained from the 4×4 case is not an adequate solution to implement real-time 8×8 differential decoding. As such, the BYU wireless lab's test system does not have enough processors to implement real-time 8×8 differential decoding. The BYU wireless lab should concentrate on a complete 4×4 implementation with increased bandwidth to make full use of the available processing power. The 8×8 case should also be explored but without the expectation of real-time communication. However, with the test system, additional DSP processors can easily be added to allow for increased processing requirements.

ACKNOWLEDGMENTS

I would like to express my gratitude to all of the people that kept pushing me to finish this thesis. I would also like to thank those that made this possible through their hard work, dedication, and love. I thank you all.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The explosive growth of the Internet has accustomed users to fast download and upload speeds at negligible costs. However, cellular wireless internet access is comparatively still slow. This is not surprising as those cellular systems were designed for low bandwidth/high user throughput and not high speed data access [1]. As cellular wireless technology becomes more popular for text messaging, networking, and low cost telecommunications in developing countries, more bandwidth is always needed to accommodate users.

The high cost of radio spectrum is another concern for current wireless data access. In the year 2000, third generation (3G) wireless spectrum (1800 MHz, 1900 MHz, and 2100 MHz) made headlines when Germany auctioned off its 3G spectrum for a total of 48 billion dollars. The UK auctioned off its spectrum for 33 billion dollars. In the United States, Verizon Wireless offered 1.6 billion dollars for one of three 10 MHz licences available in New York [2]. With this spectrum acquired, billions have been invested to develop and implement the 3G wireless network. Clearly, maximizing the capacity of available Radio Frequency (RF) spectrum is both desirable and necessary for companies to make a profit on purchased spectrum. New transmission techniques are required to make efficient use of available bandwidth.

Multiple-Input-Multiple-Output (MIMO) transmit techniques, in which several antennas are used at both the transmitter and receiver (see Figure 1.1), look to be the most promising methods of high bandwidth wireless communication. It has been shown that single-user achievable data rates grow linearly with the number of uncorrelated transmit and receive antennas with special space-time processing tech-

niques [3]. The benefits of MIMO wireless are not limited to an increase in data rate. Data rate may be traded for reliability and/or distance.

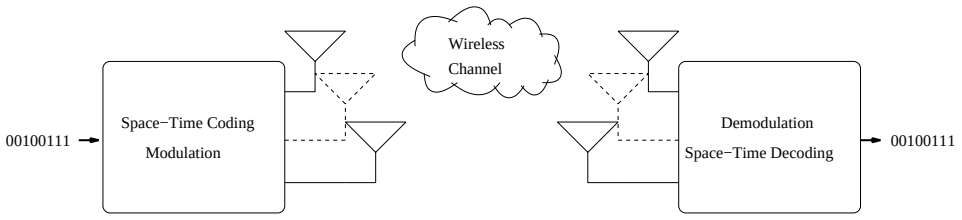

Figure 1.1: MIMO Wireless Transmission

Wireless home networking has recently exploded with the popularization of IEEE 802.11. One of the latest standards currently making its way through the IEEE standardization process is 802.11n, which takes advantage of MIMO algorithms to increase network throughput or distance. Other so called "pre-n" devices have already come to market and suddenly MIMO has become a marketing term to sell the latest and greatest home networking devices. Although the 802.11n standard makes the use of a 4×4 MIMO system possible, all consumer devices appear to use only the 2×2 configuration. Additional research is needed into higher order (i.e., 4×4, 8×8, 16×16, etc.) real-time MIMO systems.

To study MIMO systems with several antennas, a lot of generic computational power is needed. To support many different MIMO algorithms, a system must be flexible enough to use only a few processors or many different processors to accommodate the computational demanding MIMO algorithms. Due to the high throughput rate of MIMO communication, memory will need to be abundant, easily accessible, and be able to be upgraded. In short, the ideal research system must be capable enough to support any MIMO algorithm that is being researched.

In today's fast paced market where companies are competing to come to market first, rapid prototyping and development of hardware is a must. Commercial-off-the-shelf (COTS) hardware helps reduce some of the cost and time associated with

developing new hardware and software solutions. COTS systems have the benefit of being both modular and expandable. This type of modularity gives way to many different opportunities to use parallel processing to solve MIMO space-time computational needs and begin experimenting with higher-order real-time MIMO algorithms.

1.1 Commercial-Off-The-Shelf Hardware

Commercial-off-the-shelf hardware is primarily used for testing and development of hardware and software algorithms where hardware flexibility is required. The advantages of COTS hardware lie in its low cost, modularity, and the ease with which it can be upgraded. COTS systems provide additional flexibility by allowing the use of parallel processing when additional computational power is needed.

Parallel processing is a key advantage to using COTS DSP systems. This hardware inherently provides this opportunity by use of modular boards, all connected by a common backplane, that may be added, removed, or upgraded as needed. The function of these boards can range from single-processor application-specific devices to generic multiple-DSP based boards. Combinations of different boards make possible many different parallel architectures.

Some of the common parallel processing topologies that can be found on DSP systems are bus, ring, grid, and star as shown in Figure 1.2. Bus-based parallel processing is realized by backplane communication. Ring-based parallel processing is commonly found on individual multi-DSP boards. Grid and star-based topologies may be found on some boards or created by limiting communications paths on and across DSP boards. For example, board-to-board communication may be required when combining two multi-DSP boards to create one parallel processing system. Although the board-to-board communication is done over a bus backplane, DSP to DSP connections across boards can be treated as *virtual* DSP to DSP connections. Virtual connections make possible the use of other parallel processing topologies that are not inherent in the hardware architecture. Virtual connections will be discussed in more detail in Chapter 3.

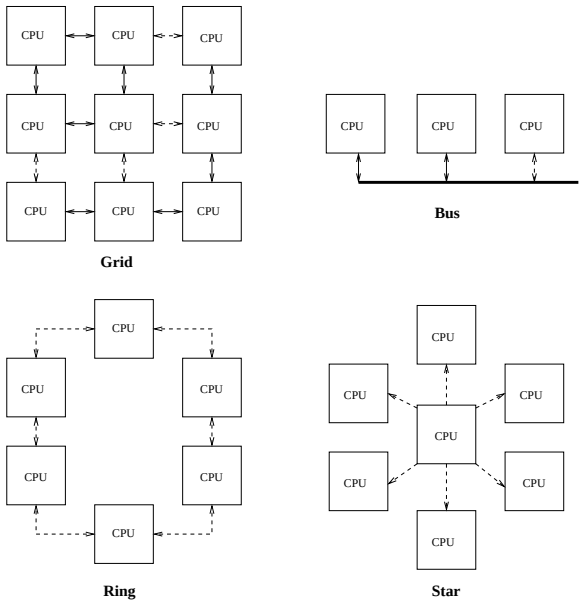

Figure 1.2: Common Parallel Processing Topologies

MIMO wireless devices with two antennas or greater work well with the configurable nature of DSP parallel processing hardware. These systems provide the flexibility needed to experiment with and develop real-time MIMO wireless space-time coding algorithms. Processing power may be added or taken away as needed, creating an ideal experimental platform.

1.2 MIMO Space-Time Codes

To take advantage of the available spatial capacity, special codes spread over space and time are used. This spreading of information over space and time is called spatial multiplexing and is possible only in the MIMO channel. Spatial multiplexing makes possible a linear increase in data rate with the addition of more antennas. Signal power and bandwidth need not be increased to realize this increase in data rate, unlike common wireless communication techniques. A rich scattering environment is needed and makes MIMO wireless perfect for multi-room buildings and urban environments.

This thesis will discuss two of the more popular space-time coding techniques: Alamouti and differential space-time coding. Mapping of these techniques onto several multiprocessor architectures will be studied. Alamouti requires information about the MIMO channel at the receiver to decode the space-time information. Differential space-time coding does not require channel information at the transmitter or receiver.

Each of these space-time coding algorithms will be divided into their most basic functions. The most computationally complex parts of these algorithms will be discussed in greater detail and matched to parallel processing topologies.

1.3 BYU Wireless Lab

The Brigham Young University (BYU) Electrical and Computer Engineering Department's wireless lab is working on research and development of MIMO systems. Most of the previous work has been theoretical research into space-time coding techniques and practical work into narrow band channel modeling. Prior work at BYU includes the performance of space-time coding [4] [5] [6], characterization of the MIMO wireless channel [7] [8] [9], and modeling the MIMO wireless channel [10] [11] [12].

A departure from this previous work is now possible with the acquisition of DSP test equipment that will enable a focus on real-time implementation. Brigham Young University's Wireless Lab has designed and built an $8-10$ DSP COTS transmitter and a $16-20$ DSP COTS receiver. The transmitter is capable of transmitting on one to ten channels and receiving on one to eight channels. An RF transmission device and an RF front end have previously been developed at BYU for channel measurements [13].

The initial publication of real-time MIMO system research was also accomplished by Jon Wallace et al. [14]. This research took advantage of the Wireless Lab's two card cages with multiple Digital Signal Processors (DSPs) and transmitting and receiving capabilities. Real-time video streaming, symbol error rate measurements, and channel measurements were accomplished.

We model our MIMO system using the uplink scenario as shown in Figure 1.3. The uplink scenario can be compared to the wireless communication between a cell phone and a base station. This assumes that we do not know the channel at the transmitter, but that we are able to obtain the channel information at the receiver with training data. With this scenario, the most complex computation is the space-time decoding on the receiver. For this reason, extra computing power is needed at the receiver. Differential space-time coding is also discussed in this thesis where the channel is not known at either the transmitter or receiver. The most complex computational need still resides at the receiver in this algorithm, matching the uplink scenario and receiver hardware configuration.

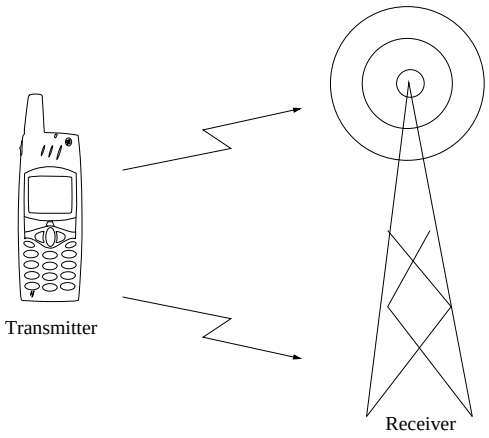

Figure 1.3: Wireless Up-Link Scenario

Figure 1.4 shows a diagram of BYU's MIMO test transmission system. The RF front end is shown as well as our GPS frequency reference receiver that enables true wireless capability without the need for carrier phase recovery. Each channel on the transmitter has access to a dedicated DSP for all communications-related computations as well as any space-time coding. Each channel on the receiver has access to a dedicated DSP as well as shared access to two additional DSPs. A possible configuration could include one DSP for communications-related processing and one

for space-time decoding. When one processor is not enough, an extra board with four processors is available for additional computations.

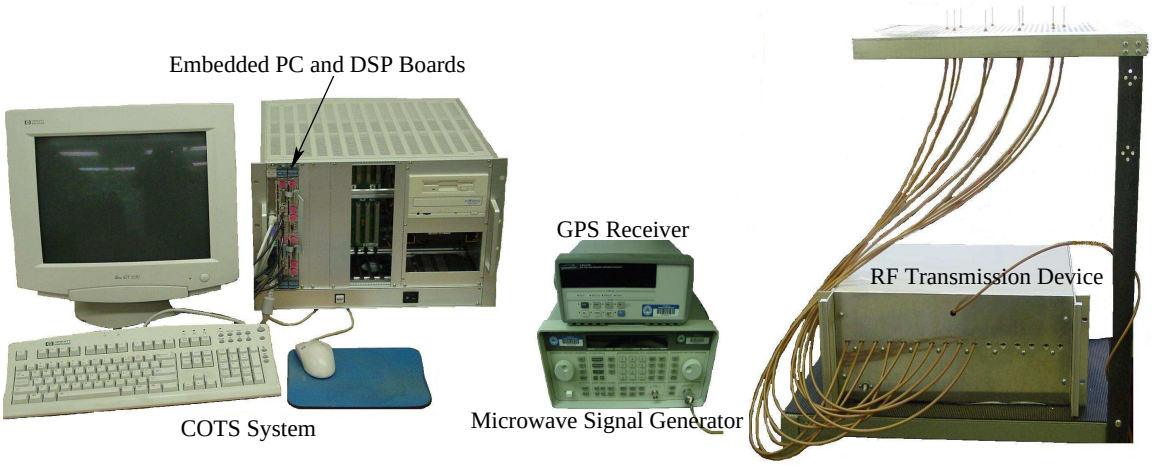

Figure 1.4: BYU Real-Time MIMO Transmission System

1.4 Objective

Early published research into the real-time implementation of high order MIMO systems was limited to Bell Lab's V-BLAST laboratory prototype [15]. Foschini also mentions parallel processing as a method of implementing a D-BLAST receiver [16]. However, it appears that a D-BLAST receiver has not been implemented by Bell Labs. No research was found that addresses parallel processing architecture impacts to MIMO algorithm implementation.

Recently, additional MIMO test beds are beginning to appear. The results appear to be limited to implementing and testing the draft IEEE 802.11n standard with the 4×4 case as the maximum number of antennas possible. These give little detail into the implementation and parallel processing requirements. Clearly, more information about the real-time performance of higher order MIMO space-time coding algorithms on parallel processing systems is needed to help close the gap between the digital signal processing and parallel processing communities. Higher order antenna

configurations (greater than the 4×4 case) is where unique MIMO research can easily be found. This thesis lays the foundation for the implementation of higher order systems.

When deciding how to implement MIMO algorithms on parallel processing systems, a review of simple common parallel processing topologies is needed in order to understand how to architect the system. More advanced topologies will also be considered to evaluate their applicability for our test system and possibly give insight into higher order MIMO systems for future work. This thesis lays the basis for a parallel processing MIMO test system.

The methodology used is to carefully measure and record the processor and inter-processor communication benchmarks. These benchmarks include inter-processor communication tasks like memory block transfers between processors, as well as arithmetic operations benchmarked on the processors. Starting with a solid base of benchmarks allows us to make educated decisions of how and when processors should communicate and how algorithms can be optimized to the available hardware. The Alamouti, 2×2 and 4×4 differential space-time decoding algorithms will be discussed, analyzed, and benchmarked on our system as well.

Piecing together each MIMO algorithm plus its associated inter-processor communication using benchmarks has the benefit of being able to quickly create and judge topologies without expending wasted time and effort of poor implementation. This methodology also has the added benefit of software libraries being created along the way that can be used for testing or for actual MIMO algorithm implementation. The drawbacks include the extra time and effort that is expended at the beginning of the project setting the system up for future use. Ideally, all of the preliminary characterization of the system will pay large dividends later on in research. However, if they don't, time is wasted that could be used implementing "gut feeling" ideas that may work adequately for their intended purpose.

1.5 Overview

This thesis is divided into two parts: the first part (Chapters 2, 3, and 4) deals with the background information needed to understand and develop real-time MIMO communication. Chapter 2 goes into greater detail on the real-time platform that the BYU wireless lab has designed and built. The functionality of individual processing boards in the test system will be discussed. The RF transmission, RF front-end, GPS receiver, and signal generators will also be presented to give the reader a better understanding of how the entire system works together and the parallel processing capabilities inherent in the modular design and the many possibilities that they offer.

Chapter 3 deals with multiple processor architectures and topologies. The advantages and disadvantages of each topology including inter-processor communication will be discussed. An example of each topology as it could apply to the BYU system will be illustrated to help the reader understand the parallel processing possibilities with our test system. Chapter 4 examines space-time encoding and decoding algorithms in depth. Alamouti and differential space-time codes will be discussed. Each of these space-time coding algorithms will be broken down into their most basic functions and analyzed so that they may be implemented on our test system.

The second part (Chapters 5, 6, and 7) gives insight into the real-time performance aspects of the BYU system. Chapter 5 will combine the knowledge from Chapters 2, 3, and 4 to make a decision as to how each space-time algorithm will be implemented following the specific multi-processor topologies discussed in Chapter 3. Chapter 6 discusses and analyzes the results achieved from piecing together the inter-processor communication benchmarks and an actual decoding algorithm implementation on the test platform. This thesis concludes with Chapter 7, including recommendations and future work. Appendix A documents the real-time benchmarks obtained.

Chapter 2

MIMO Real-Time System

The BYU wireless lab has assembled two card cage chassis capable of multi-channel transmission and reception. One of the card cages has been designated the transmitter and the other the receiver. These systems are available for research into real-time MIMO wireless techniques and narrow/wide-band channel modeling.

The real-time platform has been developed to fulfill the following requirements:

1. *Variable bandwidth.* The system must be able to run in narrow-band and wide-band modes.

2. *Variable sampling frequency.* The system must be able to change the sample clock on the transmitter and receiver to facilitate variable symbol rate transmission and reception.

3. *Variable modulation techniques.* The system must be able to transmit using different modulation techniques such as BPSK, QPSK, as well as 16QAM and 64QAM. User programmable modulation schemes must be possible.

4. *Variable number of channels.* The number of transmit and receive antennas must be changeable. Single channel transmission and reception as well as up to 16 x 16 channel transmission and reception.

5. *Variable space-time coding techniques.* Researchers must be able to experiment with different space-time coding techniques. These could include Alamouti, differential space-time coding, and even V-BLAST.

6. *Real-time or post-processing.* Researchers must be able to capture wide-band unprocessed data and store it for post processing. Researchers must also be able

to capture and process narrow-band data in real-time. Wide-band real-time processing must also be possible in a data-block processing mode (i.e., capture data block, stop capture, process, start capture again) when the bandwidth exceeds the processing power of the DSPs.

7. *Modularity.* The system needs to be able to adapt to the user's needs. It must be able to be updated in the future as faster processors become available and new hardware is introduced.

8. *True wireless capability.* The system must have true wireless capabilities. This means that the system must be able to operate in either 10 MHz frequency reference tethered mode (cable connection between the transmitter and receiver) or 10 MHz true wireless, non-tethered mode. The 10 MHz reference signal will help take care of system timing issues.

The following sections describe the hardware and software that is used to meet these goals.

2.1 System Level Diagram

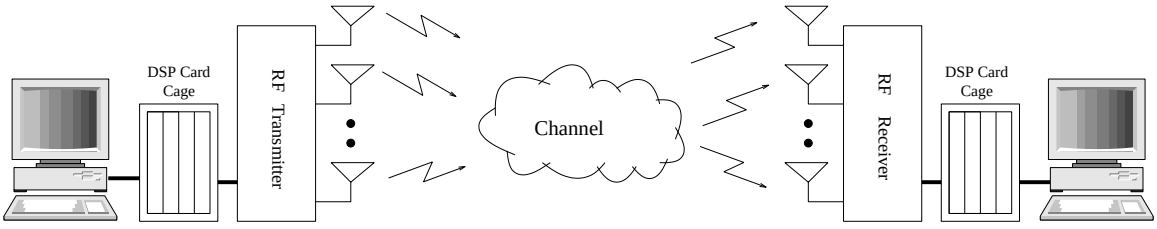

Figure 2.1: System Level Diagram

Figure 2.1 shows a system level diagram of the MIMO system. Data is transmitted on up to 10 channels using the DSP transmitter. There is one DSP processor responsible for each channel on the transmitter. This DSP is programmed to modulate Root-Raised Cosine (RRC) symbols to QPSK over a 12 MHz Intermediate Frequency

(IF). The 12 MHz IF ensures adequate spacing for 16 MHz wide-band signals, 8 MHz above 12 MHz and 8MHz below. The signal is then mixed up to a 2.4 GHz carrier frequency and transmitted using dipole antennas.

At the receive end, the signal is captured using up to 8 dipole antennas. Only 8 antennas are possible on the receive end due to the limited number of digital receiver modules. The signal is then mixed down from 2.4 GHz to 12 MHz and demodulated back into I and Q at baseband. The captured data may then be used for real-time decisions or transferred to the host computer for post processing.

In order to simplify research and development, carrier frequency offset will be ignored. For this assumption to be valid we need to ensure that all of the system's clocks have the same frequency reference. A cable is connected to the *in* and *out* of each signal generator's 10 MHz frequency reference. One of the signal generators is assigned the status as master and its 10 MHz frequency reference is passed along to all of the other signal generators. This mode of operation will be considered *tethered*. For *non-tethered* wireless communication, two GPS frequency reference receivers are used to send a common 10 MHz sync signal to each system. The problem is also sufficiently mitigated when the transmitter and receiver are in close proximity.

The transmitter and receiver are each connected to embedded PCs with 80 gigabyte-byte hard drives and CD burners to record data. The embedded PCs also have 1 GHz of processing power for any pre- or post-processing. It is possible to transfer data between the PC and the DSPs at up to 17 MB/s, thus allowing quick pre- and post-processing communication.

Figure 2.2 shows all of the components of the real-time MIMO wireless system. Notice that it is still possible to transmit in narrow band using a data pattern generator that was purchased and used for narrow band channel modeling [13]. Transmission on up to 16 channels is possible when using the data pattern generator.

2.2 Transmitter

One of the card cages is designated the transmitter and is capable of transmission on up to 10 different channels using 12 DSP processors. Each channel consists

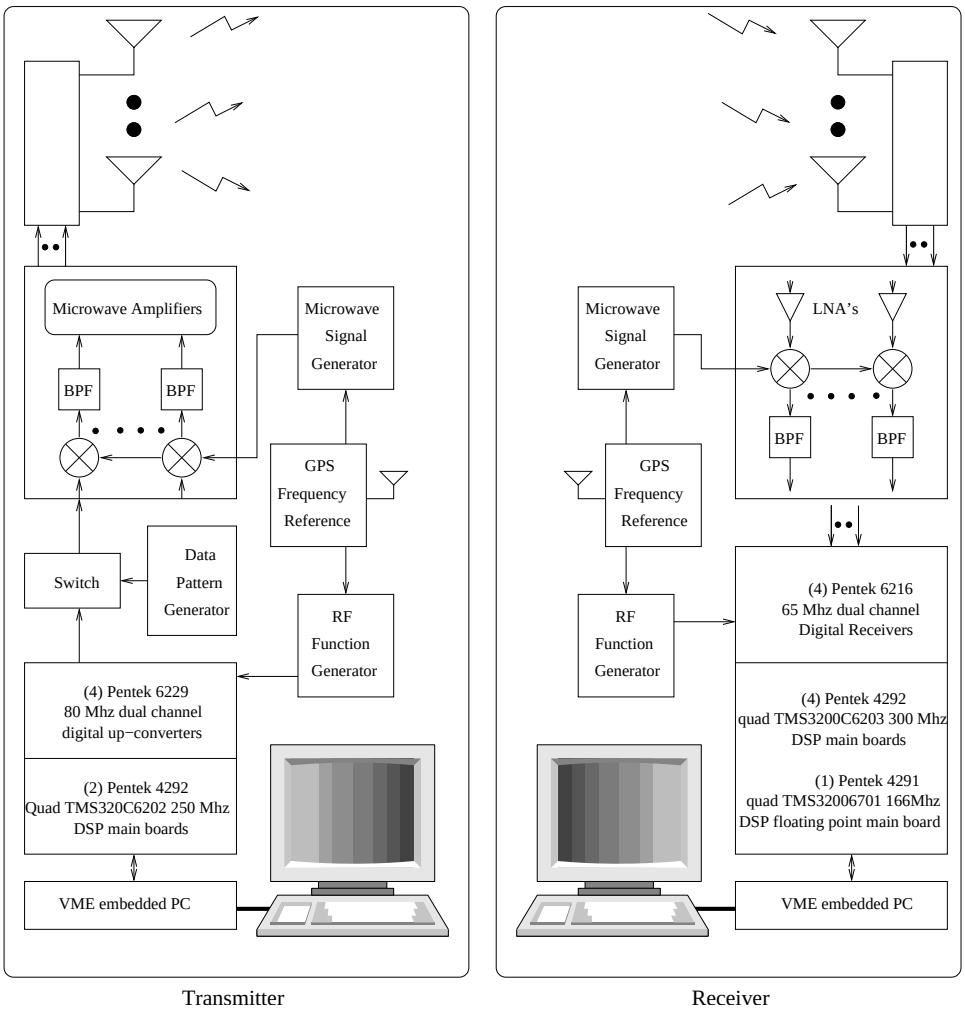

Figure 2.2: BYU Real-Time MIMO System

of one Texas Instrument (TI) 6202 250 MHz DSP processor which feeds data (in I and Q format) into a digital up-converter to modulate a baseband signal onto an IF carrier and then through a digital to analog converter (DAC).

This generic approach allows us to send any type of modulated signal at variable symbol rates. This system is available from Pentek Inc. by purchasing and installing three 4292 multi-DSP main boards with two 6229 daughter boards on each 4292. Figure 2.3 shows a picture of the transmitter card cage with attached DSP boards.

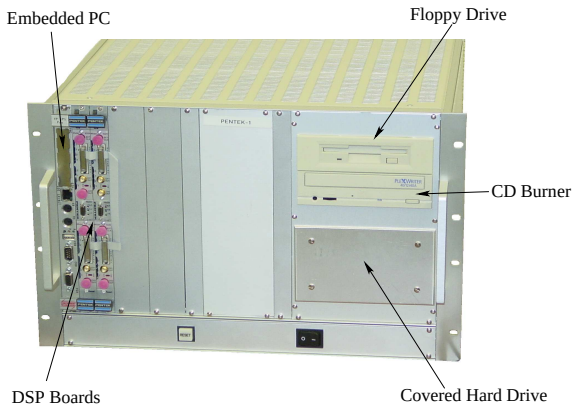

Figure 2.3: Transmitter Card Cage

2.2.1 DSP and Host Hardware

Embedded PC

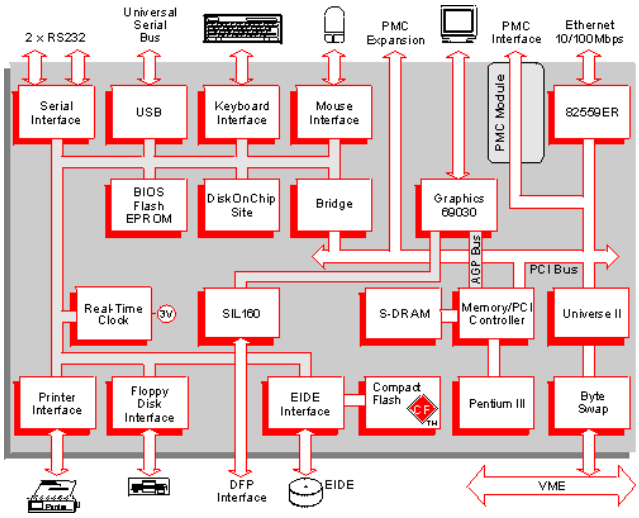

Figure 2.4: Embedded PC [17]

The transmitter is controlled by an embedded PC, as shown in Figure 2.4, with a Pentium III 1 GHz processor. Input controllers include a keyboard, computer mouse, USB port, floppy drive, CDROM drive, hard disk, and ethernet port. Output

controllers include a monitor, USB port, serial port, ethernet, writable CDROM drive, and a hard drive. The embedded PC is attached to a VME bus card cage. The VME bus allows fast communication with other cards in the card cage.

DSP boards

Pentek 4292 DSP boards are used for DSP processing. These boards each come with four 250 or 300 MHz processors connected as shown in Figure 2.5.

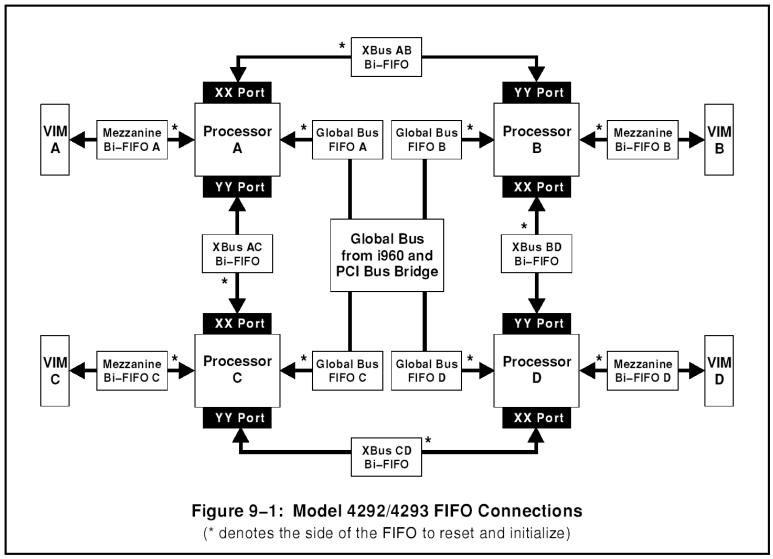

Figure 2.5: 4292 Processor Interconnects Block Diagram [18]

Data is passed from the embedded PC, through the VME bus, and into the 32 MB of global memory on each board. Data may then be transferred to each DSP's local memory. The local memory consists of 256 KB of fast on-chip memory and 16 MB of local off-chip memory.

For DSP board-to-board communication two methods exist. The first is to use the VME bus. However, there also exists a Raceway bus that can connect up to 4 DSP boards together. Board-to-board communication of up to 128 MB/s for writes and 29 MB/s for reads is possible when using the Raceway bus [18]. Unlike

the VME backplane bus, the Raceway bus can handle up to two different routings (when 4 boards are used) at one time allowing for full duplex communication. Figure 2.6 shows the Raceway bus and its routing capabilities. Slot letters E and F are not used in our current configuration. They are reserved for the connection to additional raceway modules.

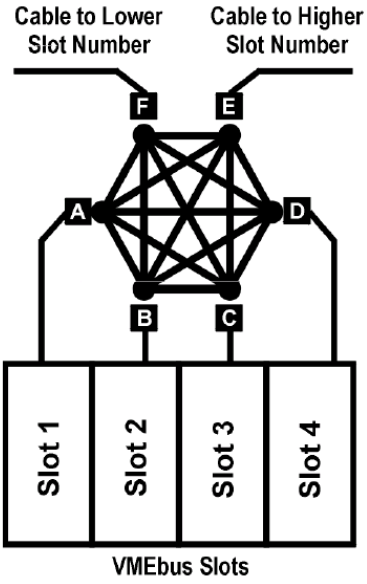

Figure 2.6: Raceway Bus Interconnect [19]

Software Architecture and Partitioning Schemes

For each channel on the transmitter, data generally flows as shown in Figure 2.7. The embedded PC will perform any pre-processing on the data to be transmitted and then transfer the data into the 4292's global memory on each board. Each processor will then transfer the appropriate data from the global memory to its local memory as space permits. Each processor will then convert the data to symbols and separate the symbols into I and Q data according to the designated modulation technique. Throughout this thesis, QPSK modulation is assumed. The 6229 will then take care of transferring the digital data onto an IF frequency and converting

the digital data into an analog signal that will later be up-converted to the carrier frequency and sent over the wireless channel.

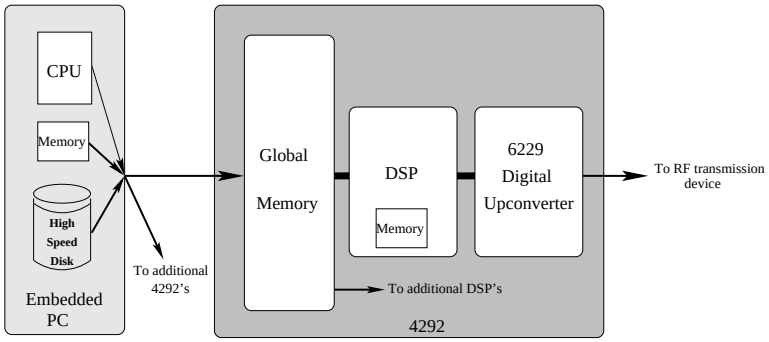

Figure 2.7: Possible Data Flow on Transmitter Computing Hardware

Digital Up-Converter

The Pentek model 6229 contains two identical, independent channels of interpolation and frequency translation suitable for linking a DSP to a radio transmitter. The 6229 can translate I and Q digital signals to IF frequencies as high as 80 MHz. At the heart of the 6229 is an Analog Devices 200 MHz Quadrature Digital Up-converter that makes all of this possible. A 12-bit 200 MHz D/A is used to output an analog IF signal. Multiple channel synchronization is possible with the use of a sync cable [20].

2.2.2 RF Transmission

Two custom RF chassis were built as part of an earlier research project into narrow band MIMO channel modeling [13]. The main components of the RF chassis include a broadband backplane that distributes the local oscillator (LO) and supplies power to the N_T transmit mixer cards [21]. The transmit mixer boards are capable of handling a 1-3 GHz LO used as the carrier frequency. We are using a LO of 2.43 GHz.

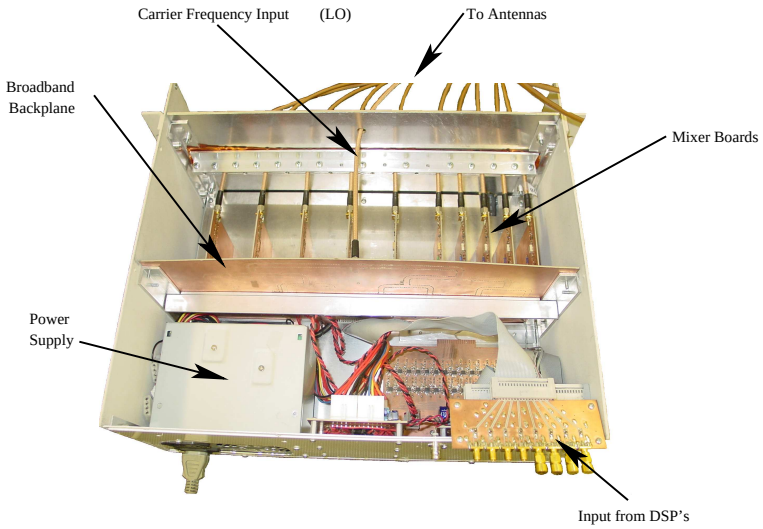

Figure 2.8: RF Transmission Device

Amplifiers may be added to the transmit card for additional transmitting power when needed. Figure 2.8 highlights the important parts of the RF transmitter.

2.3 Receiver

After a signal is received and mixed down to the IF frequency, each channel on the receiver converts the analog input signal into a digital signal for processing with the digital down converter (DDC). The DDC converts the IF frequency signal to baseband I and Q data. This data may then be processed with multiple TI 6203 300 MHz DSP processors. Some of this processing will include match filtering and symbol timing detection to recover the data sent. Figure 2.9 illustrates the receiver's primary role of digital down-conversion to I and Q data, symbol timing detection, and hard decision making. Figure 2.9 also shows an eye diagram and a constellation plot, two communication tools for displaying and debugging received data.

This receive system is available from Pentek Inc. by installing four 4292 multi-DSP main boards with one 6216 daughter board per main board. This configuration leaves an extra processor available to each channel for computation. The receiver has an additional main board, a Pentek 4291, with four floating point TI DSP's for complicated computations where a fixed point processor is not adequate.

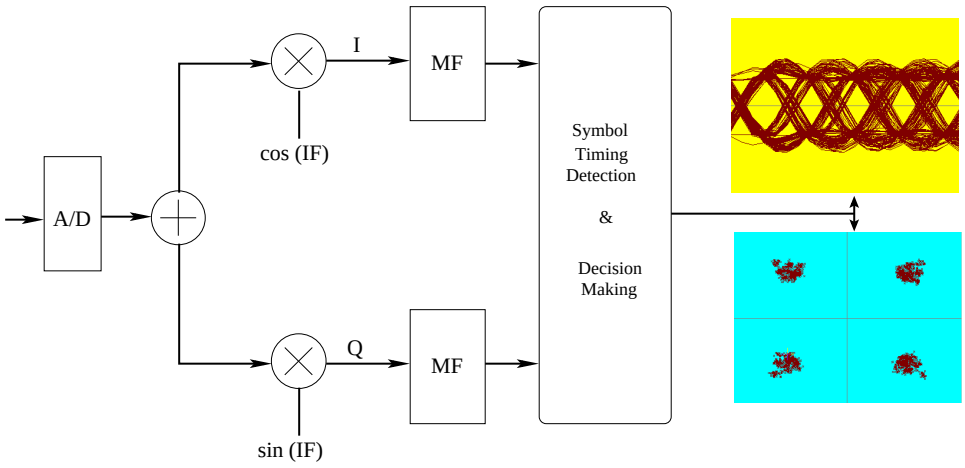

Figure 2.9: Receive Block Diagram

To communicate across DSP boards, the VME system bus or the high speed Raceway bus may be used. There is also a Pentek 6226 Front Panel Data Port (FPDP) that is available for hardware configurable DSP-to-DSP communication across main boards on the receiver. The FPDP has been configured to allow communication from the extra processors that are available on channels 5, 6, 7, and 8 to two of the processors on the 4291 DSP board. This allows additional fast communication from two of the 4292s to the 4291.

2.3.1 RF Front-End

The RF front-end is identical to the RF transmitter except that it functions in reverse (See Figure 2.8). Once again, we use a LO of 2.43 GHz.

2.3.2 DSP and Host Hardware

Figure 2.10 shows a picture of the receiver card cage with its embedded PC and 5 DSP boards.

Digital Down-Converter

The Pentek model 6216 is a complete 2-channel software radio system including tuning, filtering and demodulation [22]. The 6216 can down-convert up to a 25

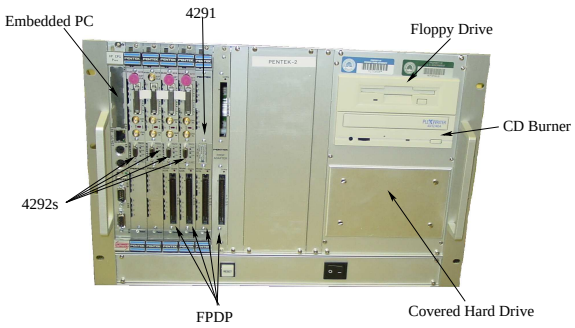

Figure 2.10: Receiver Card Cage

MHz wide-band signal, low pass filter, amplify if needed, and output digital I and Q baseband data. Multiple channel synchronization is possible with the used of a sync cable.

At the heart of the 6216 is a TI (formerly Graychip) GC1012A all digital tuner which can downconvert and band limit signals. The input signal can be down-converted to zero frequency, low pass filtered, and then output at a reduced sample rate [22].

Software Architecture and Partitioning Schemes

For each channel on the receiver, the general flow of data is shown in Figure 2.11. Data comes from the RF front-end into the 6216. The 6216 down-converts the incoming IF signal into baseband I and Q data. This data is then available to processors A and B on each 4292. These processors are first responsible for match filtering and symbol timing recovery. The data are then moved off the 4292 main board, moved to the extra processor on the board assigned to that channel, or processed on the same DSP. When data is moved off board, it first goes to the board's global memory and then either to the embedded PC for post processing or to the 4291 for fixed point calculations.

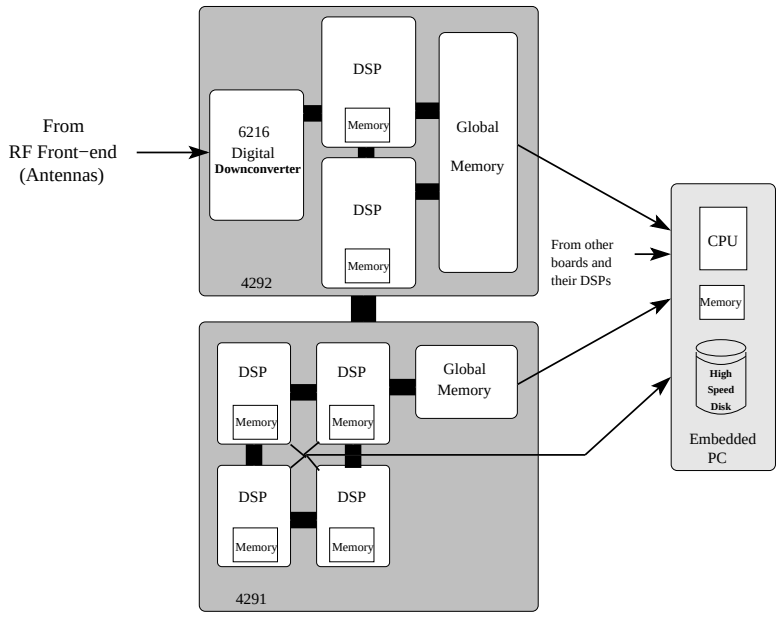

Figure 2.11: Flow of data from RF front-end to embedded PC

DSP Boards

The receiver card cage is equipped with three newer Pentek 4292 and one older 4292 main boards. The newer boards use 300 MHz TI processors. These boards have more on-chip memory, 512 KB, but less local memory, 8 MB. Additionally, the receiver has a Pentek 4291 fixed point DSP board that contains 4 TI 6701 167 MHz floating point processors. These processors are connected together in a similar fashion as the 4292, shown in Figure 2.5.

The 4291 has 2 MB of global memory and each DSP's local memory consists of 256 KB of fast on-chip memory and 16 MB of local off-chip memory. There also exists a fast 256 KB SBSRAM that can be used for data storage.

For DSP board-to-board communication three methods exist. The first is to use the VME bus, the second is the Raceway bus, and the third is by using the Pentek 6226 FPDP. The FPDP enables multi-processor communication from the third and fourth 4292s to the 4291 as previously explained. The Raceway bus can handle only 4 boards at a time, so the FPDP is used for additional fast board-to-board communication.

Embedded PC

The receiver is controlled by an embedded PC exactly like the one described in Figure 2.4 and in the transmitter section. The main difference between the embedded PC in the receiver and the PC in the transmitter is that the receiver's PC is primarily used for data acquisition. Both PCs have the same software and hardware installation. However, the receiver's PC uses Matlab for post processing and the hard drive and CD-burner for data storage.

2.4 Summary

The hardware and capabilities of the BYU wireless lab's parallel processing test system has been examined. There are many options available on how to implement a MIMO space-time algorithm on our system. Some of these include the number of processors needed for each algorithm, the functions each processor will perform in the algorithm, and how each processor interacts with the other processors implementing the same algorithm. The next step is to understand common parallel processing topologies that are possible on our system.

Chapter 3

Parallel Processing

Rapid prototyping and development of hardware and software is extremely important as competing companies wrestle for a share in the marketplace. The use of commercial-off-the-shelf (COTS) hardware and software is commonplace due to the resulting reduction of costs in time and money associated with software and hardware development. The price of COTS systems are decreasing while their modularity and the ease with which they can be upgraded are increasing. The ease of upgrading COTS hardware results from the ability to switch out obsolete boards with newer, state-of-the-art hardware, without replacing the whole system. The alternative to COTS systems, application-specific devices, may take too long to develop and test before any real software development can begin.

With a COTS approach, the same development environment may be used multiple times over multiple projects without wasting time and money on a new developmental platform for each project. Alternatively, each application-specific device could require new hardware, new firmware, and a new software development platform. With each COTS software component, less code needs to be designed and implemented by the developers [23]. Software reuse may be simplified over multiple projects when using the same development environment.

COTS systems can diminish the need to develop unique hardware and software components while at the same time ensuring fast and efficient acquisition of comparably priced component implementation. COTS hardware typically comes packaged with firmware, component drivers, and other software routines for basic board functionality, shortening the development time-line. Systems and components that already exist with similar capabilities may be used [24]. Thus, lower costs, access to

state-of-the-art technology, and readily available components/sub-systems are three of the more compelling reasons to use COTS [25].

COTS hardware provides another advantage by allowing the use of parallel processing when the computational power of a single processor is not sufficient. As illustrated in Figure 3.1, parallel processing is easily provided through the use of modular boards, all connected by a common backplane, that may be added, taken away, or upgraded as needed. Combinations of different boards may be used to create many different parallel systems.

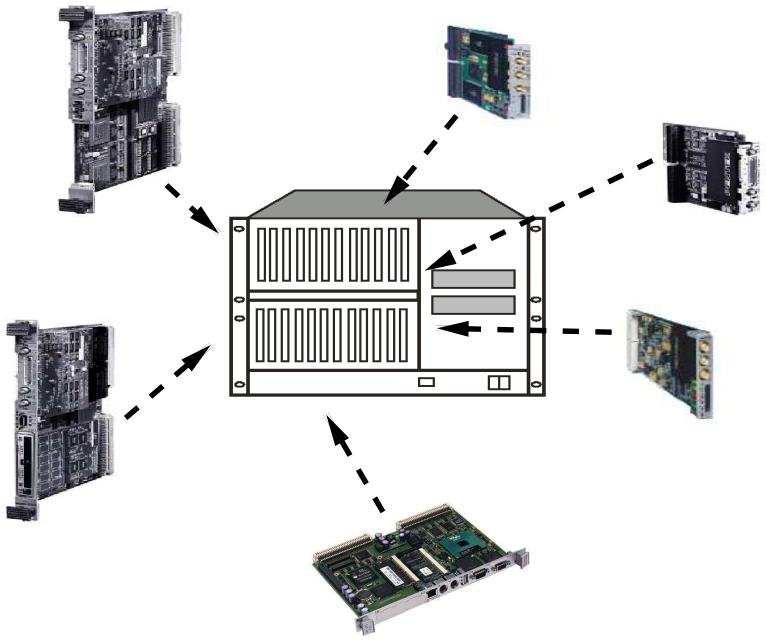

Figure 3.1: Modular COTS System

3.1 Need For Parallel Processing

Computational demand is continuing at a steady pace. Current programming practices continue to emphasize delivery time and not efficient coding. This is unlikely to change in the future as processing power never seems adequate for advanced modern applications and functionality. As silicon technology slowly approaches its

limits, parallel processing is one of the few options for meeting high computational requirements [26]. This is becoming even more evident with the prevalence of multi-core processors in commercial systems.

MIMO space-time coding algorithms increase in complexity and computational requirements as the number of antennas is increased. The additional information obtained from the increase in antennas may be too much for just one processor to handle in real-time. Parallel processing is the answer for real-time MIMO communications with multiple transmit and receive antennas.

Due to the dynamic nature of MIMO wireless technology, parallel processing appears to naturally lend itself to real-time processing. Antennas may easily be added or removed in a flexible parallel processing environment. Processing power may also be added or taken away as needed, creating an ideal experimental platform. Parallel processing COTS systems are an ideal choice for low cost research and development of real-time MIMO wireless technology.

3.2 Parallel Processing Taxonomy

A wide variety of parallel architectures can be found. One of the most popular classification schemes of parallel systems was introduced by Flynn [27]. Flynn groups computing into three different types:

- **SISD (Single-Instruction Stream/Single-Data Stream)**

  This architecture is commonly referred to as a Von Neuman computer. A single CPU functions as a SISD system.

- **SIMD (Single-Instruction Stream/Multiple-Data Stream)**

  In a SIMD system, one processor generally controls all of the nodes, feeding each node with the same instruction and executing the program synchronously [26].

- **MIMD (Multiple-Instruction Stream/Multiple-Data Stream)**

  In MIMD systems, each processor acts separately, executing different instructions. There exist two main sub-categories of MIMD systems:

  - **Shared Memory Systems**

    In shared memory systems, processors share the same globally available memory. Processors communicate with each other through this memory and synchronization is required to insure no conflicts between the processing elements exist.

  - **Distributed Memory Systems**

    In distributed memory systems each processor has locally available memory and thus avoids global memory contentions. Processors must then be connected to be able to communicate with each other, although the connections need not be direct.

  MIMD interconnection networks are commonly based on buses or direct connections between processors. In bus-connected systems, all processors, parallel memories, and any other devices are usually connected to the same bus. In directly connected systems, the interconnection networks could consist of crossbars, partially connected grids, or multistage networks [26].

MIMD systems are the most common parallel architecture in use today. This thesis concentrates on MIMD parallel processing topologies.

3.3 Parallel Processing Topologies and Architectures

An interconnection network refers to the mechanism of connecting processors and memory together in a parallel processing architecture. The ideal interconnection network connects all processors to each other, ensuring rapid communication between processors. However, as the number of processors grows large, this interconnection network grows expensive [28].

Parallel processing topologies and interconnection networks found on common parallel processing systems, and more importantly realizable on our system, are bus, ring, grid, and star. Combinations of these topologies are also found or can easily be created on our hardware and include hypercube, binary tree, and pyramid. Our system has both shared and distributed memory systems, so both will be used.

Desirable characteristics of these parallel processing topologies include extensibility, efficient routing, and reliability. Extensibility is the ability to add or take away processing elements from the topology with little change to the software algorithms. Efficient routing in this context will mean the ability of the processing elements to communicate without extensive overhead. The reliability of the topology will be determined by the ability of the topology to function without the use of one or more of its processors and/or boards [29].

3.3.1 Bus Parallel Processing

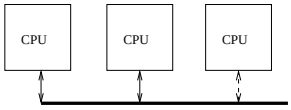

Figure 3.2: Bus Parallel Processing

Figure 3.2 shows a simple bus-based parallel system. The advantages of buses are that routing and extensibility are trivial [29]. Bus based parallel processing can be extended by simply adding additional processors to the bus. Existing software will only have to be updated with the new number of processors on the bus. Fault tolerance is also good as defective processors on the bus may be ignored. This is true as long as these defective processors do not tie up the bus and cause complete system failure.

A disadvantage with multiprocessing busses is the clock latency problem. The majority of the time, processors access their local cache or off-chip memory in a bus

based system and only use the bus occasionally. Thus, processors are optimized to communicate based on memory clocks rather than bus clocks. This inevitably leads to asynchronous access with the bus clock. Thus, for a processor to access the bus, its interface must synchronize the bus and the memory. This causes a delay in waiting for the first valid bus clock edge. On average, this delay will be half a bus clock cycle [30].

Another disadvantage of bus-based parallel processing is the worst case access time. Only one processor can have access to the bus at a time and bus arbitration delays are inevitably added. Thus, worst case bus access times grows with the number of processors that are added to the bus.

Shared memory systems cause even more delays as all processors have to compete for the bus to access global memory. Distributed memory systems do not have this problem; however, depending on the application, there is still a need for frequent bus accesses to get data from other processors' memories. A combination of shared and distributed memory systems can help alleviate some of these problems without much additional cost. Bus-based parallel processing may be relatively slow, but the systems can be very reliable and relatively simple to implement.

Bus-based parallel processing is inherent from backplane communication in parallel processing systems. Bus-based parallel processing systems are attractive for several reasons. Low cost, a standard interconnect that allows multiple vendor board designs, and the ease of additional computing power by adding more boards are among the most important reasons [30]. Figure 3.3 highlights a bus-based parallel processing topology (board-to-board communication) found in the BYU wireless lab's test platform.

3.3.2 Line and Ring Parallel Processing

Figure 3.4 shows a line and ring parallel processing topology. Compared to a bus-based parallel architecture, more communication is possible in a ring architecture because adjacent processors can communicate with each other with no wait time. Multiple simultaneous transactions involving different processors are thus possible,

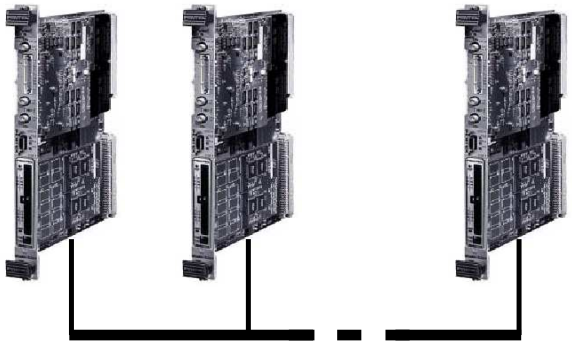

Figure 3.3: Bus Parallel Processing

unlike bus-based systems. Point-to-point processor connections make higher data rates possible and alleviate the need for bus arbitration and worse case bus delays.

Rings may be unidirectional or bidirectional. In a unidirectional ring topology, data is passed in only one direction. In a bidirectional ring topology, data may be passed in either or both directions [28].

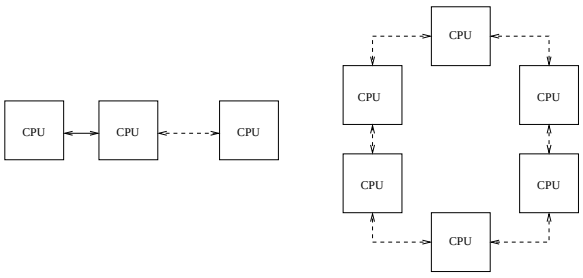

Figure 3.4: Line and Ring Parallel Processing

In the ring architecture there is a reduction in communication delays compared to lines. There are two paths to any other processor, clockwise and counter-clockwise, thus fault tolerance is good. Routing remains simple and extensibility is still good. The line and ring topology are simple, but data must typically pass through multiple processors in order to reach the destination. On average, this creates long communication delays.

The line and ring based topologies are inherent using the multi-DSP boards available for many parallel processing systems, including those found in the BYU wireless lab. Figure 3.5 illustrates a four DSP ring topology found within one Pentek 4292 board.

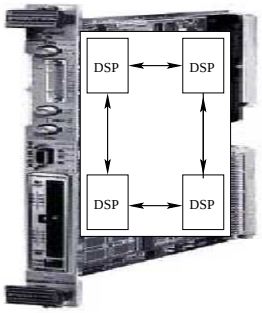

Figure 3.5: Ring Parallel Processing

3.3.3 Mesh Parallel Processing

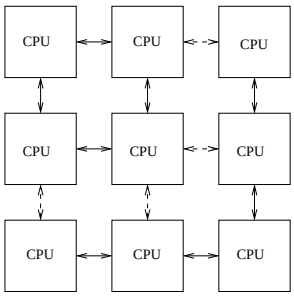

Figure 3.6: Mesh Parallel Processing

Figure 3.6 shows a mesh topology. Processors in a mesh topology are arranged in 2-dimensional matrix configuration. Each processor is generally connected to four of its neighbors, although boundary processors may not be. Some 2-dimensional mesh

topologies allow for wrap-around connections between boundary processors, usually within the same row or column [28].

Delays are added when passing data diagonally. However, an advantage of this topology is that multiple paths of communication are available. Fault-tolerance is therefore good as there are multiple paths in which processors can communicate.

Mesh topologies are more highly connected than bus, line, or ring. Inter-processor communication delays may be reduced due to the proximity of at least two and possibly three or four processors. Extensibility is a more complex issue as software may have to be modified depending on the number of processors and where those processors are added.

The mesh based topology is inherent in the BYU wireless lab's system from the interconnection of bus and ring parallel processing networks. Figure 3.7 highlights a mesh parallel processing topology found in the BYU wireless lab's test system. Note that the dotted lines in Figure 3.7 are the virtual connections previously mentioned in Chapter 1. A virtual connection is the term used to explain the flexible connections afforded by the DSP equipment in the BYU lab. Virtual connections are simply the communications paths created when communicating over a fixed parallel processing topology to emulate a different parallel processing topology. Virtual connections allow the interconnection of multiple processors in multiple configurations. For example, in Figure 3.7, a 2×6 mesh is created using three DSP boards. Although the board-to-board communication is actually done over a common bus architecture backplane (VME or Raceway bus), a virtual connection is made from DSP to DSP to emulate a mesh connection. A three dimensional $2 \times 2 \times 2$ grid could also easily be created as shown later in Figure 3.12.

3.3.4 Star Parallel Processing

Figure 3.8 shows a star topology. An advantage of the star topology is that it is easily extensible. The star topology is characterized by a central processor that divides data and computation up into smaller segments for its child processors to compute. When all computations are complete, the central processor collects the

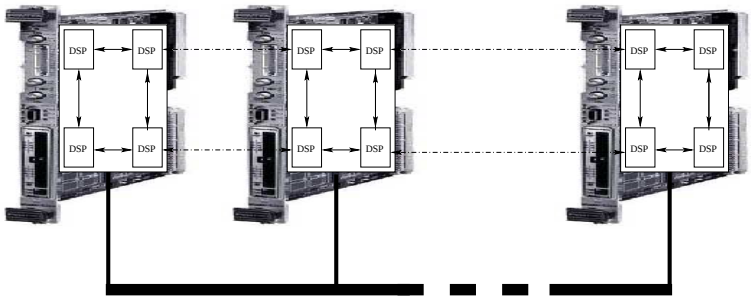

Figure 3.7: Mesh (Grid) Parallel Processing

data from its child processors and performs any final calculations to recompile the data. When routing data, each processor experiences the same delay from the central processor. No child processor has any distinct advantage over another.

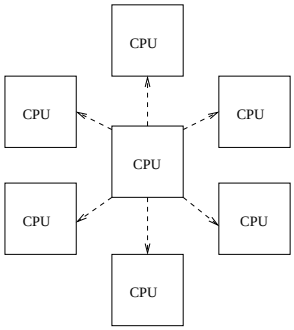

Figure 3.8: Star Parallel Processing

For some algorithms, this can be a very efficient topology as the central processor can start the computation, divide the task to all other processors, and then recompile all of the information when all processing is done. As long as the central processor is not damaged, reliability is good as the computational load may be distributed among the remaining functional processors. However, if the central processor malfunctions, a system based on a star topology is completely inoperable.

Star parallel processing may be created from the interconnection of multiple DSP boards in a parallel processing system. In our system, the central processor may

be a DSP processor or the embedded PC's CPU. Figure 3.9 shows a star topology using the embedded PC computer's CPU as the central processor. The dotted lines represent the virtual connections. Data may be passed over the VME bus and onto the DSP boards global memory using one bus transfer. If the data is partitioned for all of the DSP processors, each corresponding data block is then transferred from the board's global memory to each DSP processor. Although this is truly a series of bus-based transfers, they can be modeled using virtual connections.

Figure 3.10 highlights a star topology realizable using one DSP board within the BYU wireless lab real-time system. The virtual connection is created by passing data through the global memory to the diagonal DSP. This allows the central processor in the figure to communicate with all three DSPs on its board at the same time. The virtual connection will be slower than the IPBIFO direct connections, but simultaneous transfers are possible.

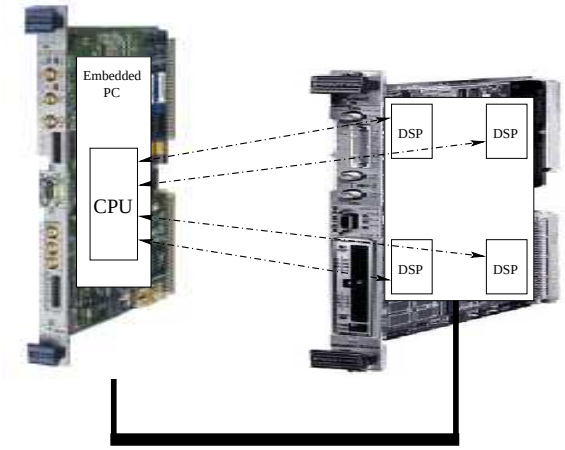

Figure 3.9: Star Parallel Processing

3.3.5 Hypercube or n-Cube Parallel Processing

Figure 3.11 shows a hypercube topology. A hypercube is formed by N processors where N is a power of 2. For $N = 2^q$, where $q \geq 0$ then a q-dimensional

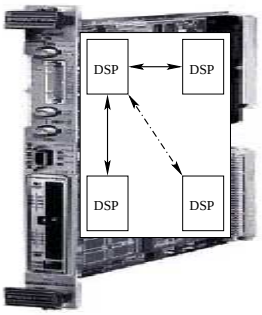

Figure 3.10: Single Board Star Parallel Processing

hypercube is formed. Each processor is connected to q neighboring processors, commonly referred to as each processor having a degree of q. The network of N processors is formally called a binary n-cubed network [28].

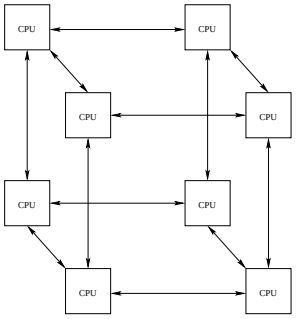

Figure 3.11: Hypercube Parallel Processing, $q = 3$

Reliability is excellent as there is always another path to other processors if one processor fails. As q increases, so does the reliability of the network. The main disadvantage with the hypercube topology is the increased complexity in routing and software algorithms.

Figure 3.12 highlights a hypercube topology possible in the BYU Wireless lab's system. The hypercube topology on our system makes use of virtual connections to connect DSPs together over the system bus, preferably the RACEway bus.

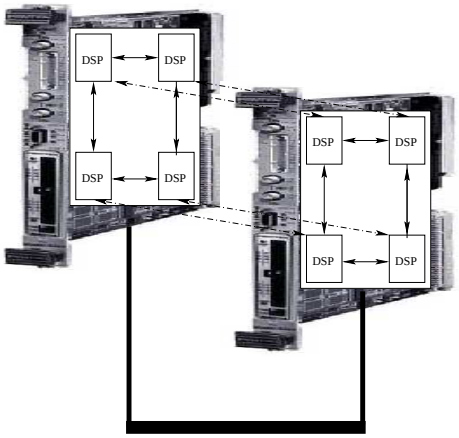

Figure 3.12: Hypercube Parallel Processing, $q = 3$

3.3.6 Binary Tree Parallel Processing

Figure 3.13 shows a binary tree topology. A binary tree interconnection network is formed when N processors are available and $N = 2^d - 1$ vertices are formed, where d is the number of levels in the tree. The levels are numbered from 0 to $d - 1$ where 0 is the root processor. Figure 3.13 shows a binary tree with $d = 3$ connected processors.

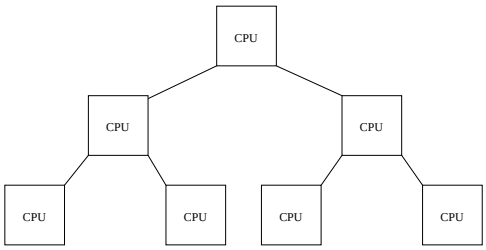

Figure 3.13: Binary Tree Parallel Processing

Every processor in the tree can communicate with its two children (except for leaf node processors) and every processor except for the root can communicate with its parent. Processors are assigned degrees according to the number of adjacent

processors they are connected to. Each interior processor has a degree 3 (two children and one parent), each terminal processor has degree 1 (parent), and the root processor has degree 2 (two children) [28].

Figure 3.14 highlights a binary tree topology possible in the BYU Wireless Lab's system. Multiple bus-based data transfers are required to satisfy all of the virtual connections used in this example. The virtual connections are highlighted using dashed lines and actually travel over the bus shown in the figure.

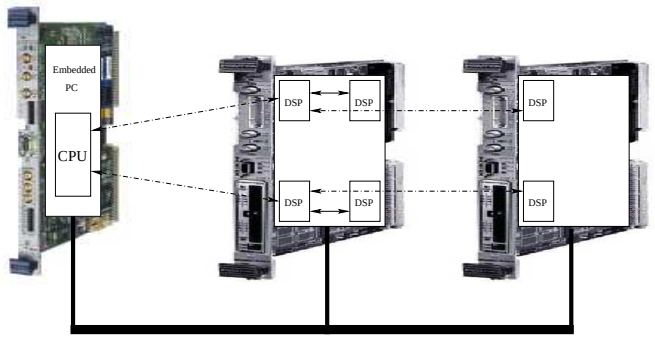

Figure 3.14: Binary Tree Parallel Processing

3.3.7 Pyramid Parallel Processing

Figure 3.15 shows a pyramid topology. A two-dimensional pyramid contains $N = (4^{d+1} - 1)/3$ processors in $d + 1$ levels (level numbers increase starting at the base of the pyramid from bottom to top) with these properties:

- There are 4^{d-2} processors at level d.
- There are 4^{d-1} processors at level $d - 1$.
- There are 4^{d} processors at level $d - 2$.

In general, any processor at level x:

- Is connected to up to four neighboring processors at the same level if $x < d$,

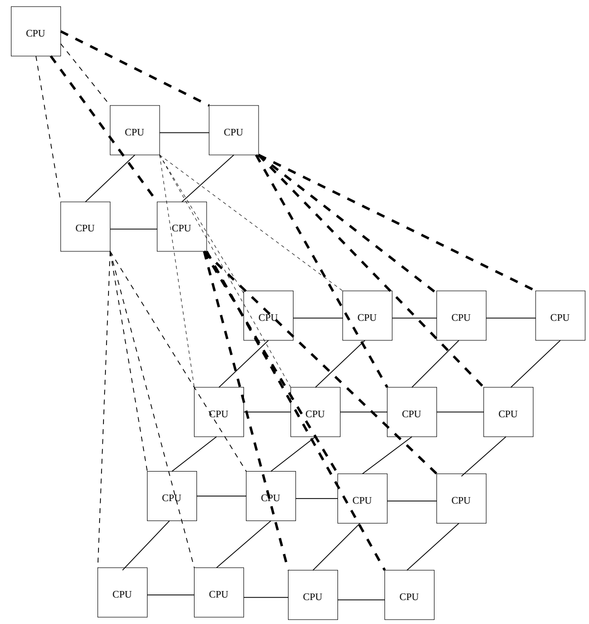

Figure 3.15: Pyramid Parallel Processing

- Is connected to four children at level $x - 1$ if $x \geq 1$, and it
- Is connected to one parent at level $x + 1$ if $x \leq d - 1$.

It can be seen in Figure 3.15 that a pyramid has a base made up of a two-dimensional mesh containing d^2 processors. An advantage of the pyramid interconnection network compared to a two-dimensional mesh network is that messages may travel up and down the tree and across the mesh rather than only across the mesh, resulting in fewer link traversals [28].

Figure 3.16 highlights a pyramid parallel processing topology that is possible in the BYU wireless lab's system. Notice the extensive use of virtual connections in our system to realize a pyramid topology. The dashed lines in the figure are all virtual connections that actually travel over the illustrated bus.

3.3.8 Summary

Topologies are key distinguishing features in parallel systems since they affect communication algorithms and computational efficiency. We have seen that a wide

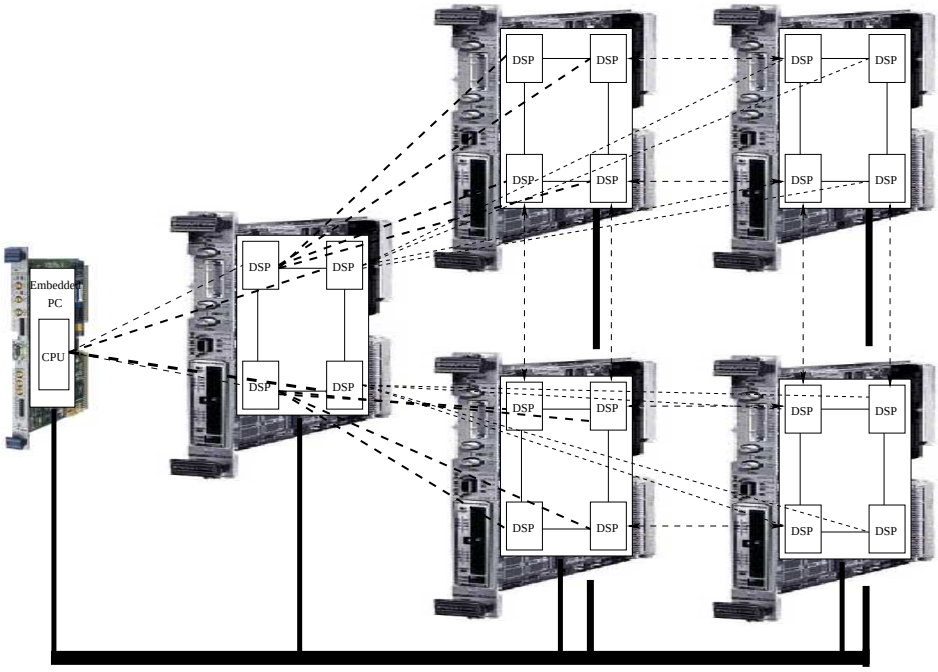

Figure 3.16: Pyramid Parallel Processing

array of different topologies can be accomplished with the test equipment available in the BYU's wireless lab using the inherent bus topology of the VME backplane or Raceway module (board-to-board communication), line/ring topology between DSPs using bidirectional inter-processor connections (called IPBIFOs), and star topology by writing/reading from global memory. This wide selection of inter-processor connections affords flexibility in designing the software to support real-time computational needs.

The next step is to understand the space-time coding algorithms so that they can be implemented and benchmarked on our test system.

Chapter 4

Space-Time Algorithms

As illustrated in Figure 4.1, there are four main antenna configurations for space-time (ST) communications systems. SISO (Single-Input-Single-Output) has a single transmit and a single receive antenna. This is the most familiar wireless communication system. MISO (Multiple-Input-Single-Output) has multiple transmit antennas but only a single receive antenna. SIMA (Single-Input-Multiple-Output) employs one transmit antenna with multiple receive antennas. MIMO (Multiple-Input-Multiple-Output) has multiple transmit and multiple receive antennas. This thesis focuses on MIMO algorithms.

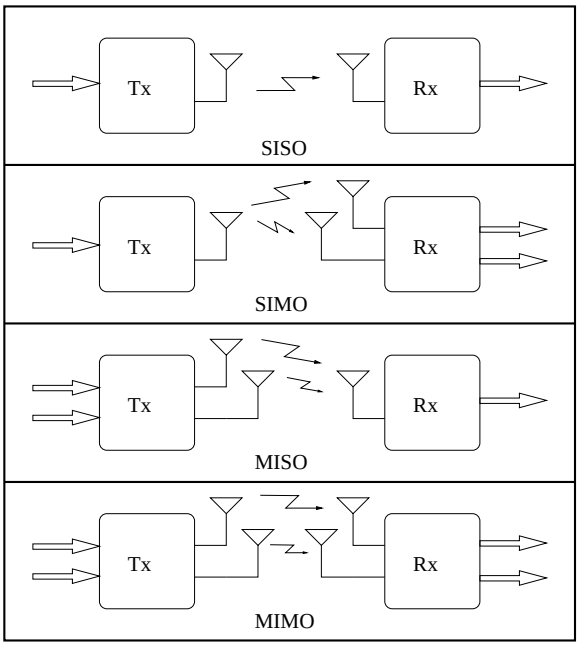

Figure 4.1: ST Antenna Configurations

Spatial multiplexing, the spreading of information over space and time and possible only in MIMO channels, may achieve a linear increase in data rate (or capacity) with additional antennas. The signal's power and bandwidth do not need to be increased to achieve this higher data rate, but special ST codes need to be employed to take advantage of spatial multiplexing. Common algorithms that accomplish these tasks are discussed in this chapter.

4.1 Background

There exist two main concepts in ST communications that are essential to understand the benefits of ST algorithms: array gain and diversity gain. Array gain is defined as the average increase in SNR at the receiver that arises from the coherent combining effect of multiple antennas at the receiver or transmitter or both [31]. For example, consider a single message signal transmitted in a SIMO communication system. At the receiver, neglecting multipath effects, the same signal will arrive at each of the multiple receive antennas, although with different amplitudes and phase. Combining these signals at the receiver correctly (with proper phase and gain adjustments to insure coherent addition) will result in one signal with increased SNR proportional to the number of receive antennas.

Diversity gain is used in wireless communications to combat the fading of wireless channels. Fading is defined as a significant signal power drop in the channel [31] and is usually caused by destructive interference of multipath signals. Again using the SIMO example, one receive antenna may fade in and out over time. During this fading, no information may be passed to the receiver. However, the signal at another receive antenna may not have any significant fading, or may fade at times other than that of the first antenna. Correctly combining the signals at the receiver will compensate for the different fading patterns of each transmit antenna, thus allowing reliable transmission [31].

4.2 Space-Time Coding Algorithms

It is beyond the scope of this paper to describe all of the known space-time coding algorithms. However, two of the most popular and well known algorithms will be introduced and examined. These include the Alamouti [32] and differential [33] space-time schemes.

Alamouti coding is well known due to the Alamouti algorithm's incorporation into the latest CDMA standard, W-CDMA [34]. The benefits of diversity gain are also well known in the wireless networking world and it is used extensively. Diversity gain is a very important part of the IEEE 802.11n standard that incorporates MIMO space-time algorithms. In fact, MIMO has become so popular that any device that has multiple physical antennas and uses some method of diversity gain self brands themselves as "MIMO" or "pre-n" wireless networking devices. Although the Alamouti algorithm may not be used for these devices, this simple space-time block code is worth implementing for insight into real-time space-time block codes.

For the Alamouti scheme, the transmitter is modeled as having no knowledge of the channel, while the receiver is assumed to be able to detect and model the channel by interpreting training data from the transmitter. Differential space-time coding is very useful when channel information is not known at the transmitter or receiver. As far as this author knows, no research into real-time differential space-time coding has been reported in the literature.

4.2.1 Alamouti Space-Time Codes

The Alamouti space-time algorithm is a simple transmit diversity scheme which improves the quality of the received signal by transmitting across two or more transmit antennas. The Alamouti algorithm can be seen as a maximal-ratio receiver-combining (MRRC) algorithm [32] with two transmit and two receive antennas. As the channel is rendered orthogonal by the transmitter, an otherwise complex maximum likelihood detection problem is decoupled into a simple scalar detection problem at the receiver. Orthogonality is the key to Alamouti space-time code and greatly

simplifies the receiver decoding. For two transmit antennas, full $2M_R$ order diversity, where M_R is the number of receive antennas, is achieved.

The Alamouti scheme can improve the error performance, data rate, or capacity of wireless communication systems. This is achieved by decreased sensitivity to fading, by allowing higher modulation schemes due to increased resistance to noise, and increased system performance in a multi-cell system.

Space-time codes for Alamouti-type schemes can also be created using orthogonal codes for greater than two transmitters [31]. Although less processor intensive to decode than most space-time algorithms, real-time implementations of Alamouti require parallel processing on our test system due to the multi-antenna requirements. A 2×2 scheme will be discussed that is easily extensible to the $2 \times M_R$ case.

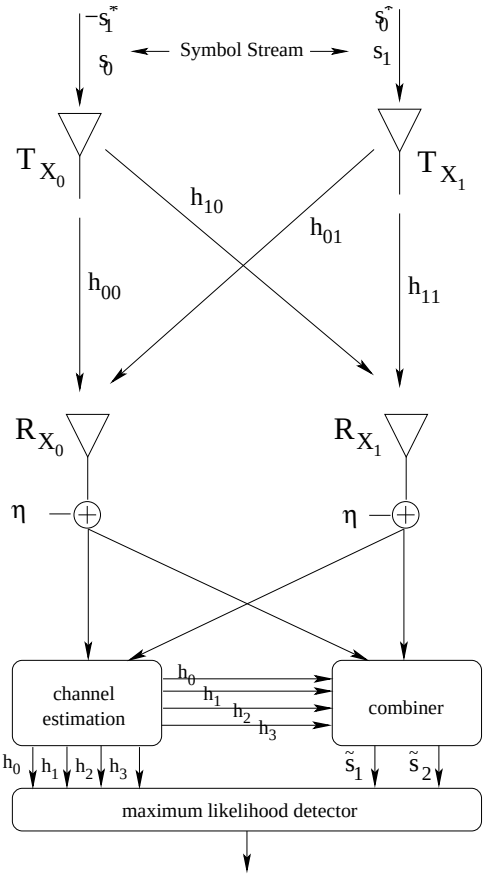

Figure 4.2: 2×2 Alamouti Two Branch Diversity with Two Receivers

Basic Operation

For a given symbol period, where the number of transmit antennas and the number of receive antennas equals two, $M_T = M_R = 2$, two distinct signals are transmitted simultaneously on separate transmit antennas. Figure 4.2 shows the baseband representation of the Alamouti scheme with $M_T = M_R = 2$.

The symbols s_0 and s_1 are transmitted from antenna 0 and antenna 1 respectively at time t. During the next symbol period, $t + T$ where T denotes the symbol period, signal $-s_1^*$ is transmitted from antenna 0 and signal s_0^* is transmitted from antenna 1, where '*' indicates the complex conjugate operation.

In matrix form

$$\mathbf{S} = \begin{bmatrix} s_0 & -s_1^* \\ s_1 & s_0^* \end{bmatrix} \tag{4.1}$$

where the row indices correspond to the transmit channel and the column indices correspond to the symbol timing slot, as shown in Table 4.1.

Table 4.1: The Encoding and Transmission Sequence - Matrix **S**

	T_X antenna 0	T_X antenna 1
time t	s_0	s_1
time $t + T$	$-s_1^*$	s_0^*

The channel as seen by the receiver is represented as a transfer matrix,

$$\mathbf{H} = \begin{bmatrix} h_{00} & h_{10} \\ h_{01} & h_{11} \end{bmatrix} \tag{4.2}$$

assuming narrow band, where h_{ij} represents the complex (contains both amplitude and phase) gain component from transmitter j to receiver i, as seen in Figure 4.2 and shown in Table 4.2. Channel information is obtained by the use of training data

sent out at the beginning of each transmit block. The assumption is made that the channel matrix is known, or has been estimated in a prior processing stage.

Table 4.2: Channel Matrix Components (As Seen by the Receiver)

	R_X antenna 0	R_X antenna 1
T_X antenna 0	h_{00}	h_{10}
T_X antenna 1	h_{01}	h_{11}

The receive matrix $\mathbf{R}$ is formed by the equation

$$R = \mathbf{H}S + \eta \tag{4.3}$$

and broken out into familiar form in Table 4.3. Matrix η represents independent and identically distributed (iid) white gaussian noise due to receiver thermal noise and other interference in the channel.

By exploiting the special structure of S and considering only one channel at the receiver, an effective channel transfer matrix $\mathcal{H}$, over t and $t+T$ combined, we see that Equation 4.3 becomes

$$\vec{r} = \mathcal{H}\vec{s} + \vec{\eta}, \tag{4.4}$$

or

$$\begin{bmatrix} r_{00} \\ r_{01}^* \end{bmatrix} = \begin{bmatrix} h_{00} & h_{10} \\ -h_{10}^* & h_{00}^* \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \end{bmatrix} + \begin{bmatrix} \eta_0 \\ \eta_2^* \end{bmatrix}. \tag{4.5}$$

Table 4.3: Definition of the Received Signals

	R_X antenna 0	R_X antenna 1
time t	r_{00}	r_{10}
time $t+T$	r_{01}	r_{11}

The received signals may then be combined to give signal estimates, $\tilde{s_0}$ and $\tilde{s_1}$:

$$\begin{bmatrix} \tilde{s_0} \\ \tilde{s_1} \end{bmatrix} = \begin{bmatrix} h_{00}^* & h_{01} & h_{10}^* & h_{11} \\ h_{01}^* & -h_{00} & h_{11}^* & -h_{10} \end{bmatrix} \begin{bmatrix} r_{00} \\ r_{01}^* \\ r_{10} \\ r_{11}^* \end{bmatrix}. \tag{4.6}$$

The transmitted signals may be compared to the received signals by substituting Equation 4.5 into 4.6 to yield

$$\tilde{s_0} = (|h_{00}|^2 + |h_{01}|^2 + |h_{10}|^2 + |h_{11}|^2)s_0 + \zeta_0 \tag{4.7}$$

and

$$\tilde{s_1} = (|h_{00}|^2 + |h_{01}|^2 + |h_{10}|^2 + |h_{11}|^2)s_1 + \zeta_1, \tag{4.8}$$

where ζ refers to noise and

$$\zeta_0 = h_{00}^*\eta_0 + h_{01}\eta_1^* + h_2 10^*\eta_2 + h_{11}\eta_3^* \tag{4.9}$$

and

$$\zeta_1 = -h_{00}\eta_1^* + h_{01}^*\eta_0 - h_{10}\eta_3^* + h_{11}^*\eta_2. \tag{4.10}$$

The signal estimates $\tilde{s_0}$ and $\tilde{s_1}$ are then sent to a maximum likelihood decoder for hard decisions on the symbol.

As seen in Equations 4.6 through 4.10, the space-time decoding process is particularly simple due to the orthogonality of $\mathcal{H}$ imposed by the structure of S. Unlike

other methods (e.g., BLAST) no matrix inversion or singular value decomposition is required.

4.2.2 2 x 2 Differential Space-Time Modulation

Differential space-time coding [33] in MIMO wireless communications does not require channel knowledge at the transmitter or receiver. This may be useful, if not necessary, in certain situations when channel estimation is too costly or complex for the hardware. This can be true in dynamic environments where the channel changes so rapidly that channel estimation cannot track changes without excessive estimation error or requires too many training symbols for effective signalling.

Differential space-time coding is a methodology for the design of differential transmit diversity schemes based on groups of unitary matrices. This general approach may be used with any number of antennas or signal constellations if a continuous range of symbol constellations is permitted. For finite symbol alphabets, unitary coding matrices may be known only for a fixed number of channels. Because the real-time digital radio will initially use the QPSK signal constellation, only a 2×2 full rate MIMO scheme is possible.

Basic Operation of the 2 x 2 Differential Space-Time Code

For a system with $M_T = M_R = 2$ antennas, let $\mathtt{G} = \{G_1,...,G_M\}$ be a group of 2×2 unitary matrices, so that

$$G^H G = G G^H = I \text{ for all } G \in \mathtt{G}.$$

where the hermitian operator, depicted by a superscript ^{H}, specifies the matrix complex conjugate transpose. The matrices of $\mathtt{G}$ are then mapped to the set of possible messages for simplicity. To transmit the message $G \in \mathtt{G}$, the 2×2 code matrix

$$C = DG$$

is sent where D is a fixed 2x2 matrix that satisfies $DD^H = 2I$, called the initial matrix. For the QPSK case

$$\mathtt{G} = \left\{ \pm \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \pm \begin{bmatrix} j & 0 \\ 0 & -j \end{bmatrix}, \pm \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \pm \begin{bmatrix} 0 & j \\ j & 0 \end{bmatrix} \right\}, D = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \tag{4.11}$$

is a group code that takes values in from the QPSK signal constellation, namely $C = \{1, j, -1, -j\}$. Hughes calls this code the quaternion code after the algebraic quaternion group [33]. The structure of the code is illustrated in Figure 4.3, where

$$\Theta = \begin{bmatrix} j & 0 \\ 0 & -j \end{bmatrix}, R = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}. \tag{4.12}$$

To start transmission, an initial matrix, $C_0 = D$, is sent which conveys no information. After this initial matrix, messages are differentially encoded. To send $G_k \in \mathtt{G}$ in block k, C_k is sent as follows:

$$C_k = C_{k-1} G_k, k = 1, ..., K.$$

The group structure guarantees that $C_k \in D\mathtt{G}$ whenever $C_{k-1} \in D\mathtt{G}$.

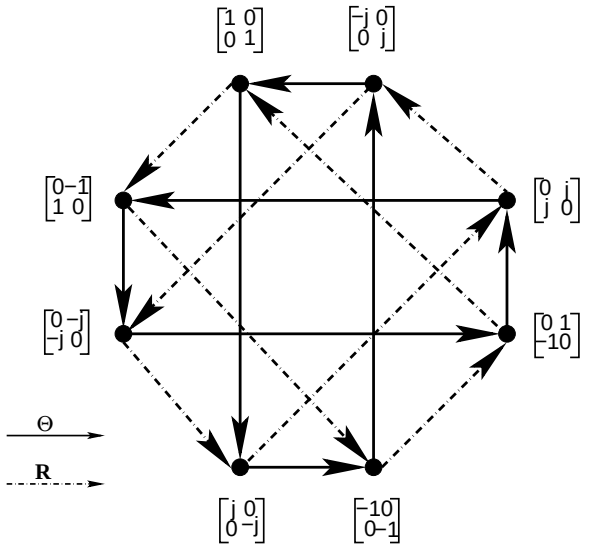

Figure 4.3: The Quaternion Group

The received signals are then decoded as discussed in [33]. The differential receiver estimates for G_k are formed using only the two most recently received blocks (Y_k and Y_{k-1}), where

$$Y_k = \sqrt{\rho_t} H C_k + N_k, k = 0, 1, ..., K.$$

The optimal detector for G_k given Y_k and Y_{k-1} is

$$\tilde{G}_k = \arg \max_{G \in \mathtt{G}} \text{Re } \{\text{Tr}\{G Y_k^H Y_{k-1}\}\}, \tag{4.13}$$

where Re{} and Tr{} refer to the real part of the trace of $GY_k^H Y_{k-1}$. This is illustrated in Figure 4.4.

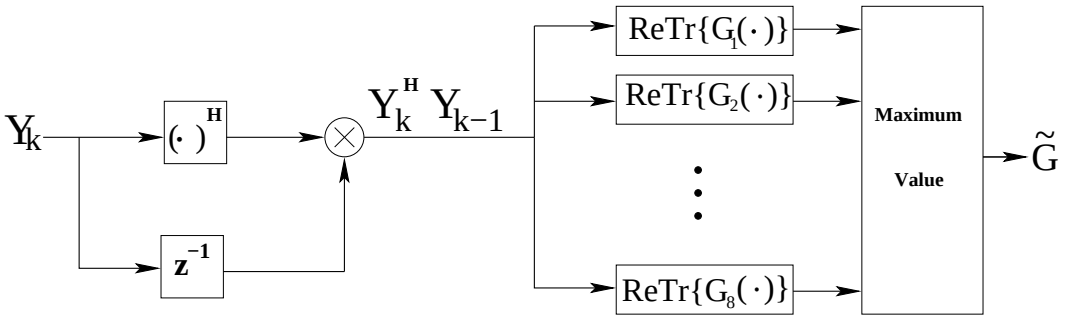

Figure 4.4: A Differential Receiver

4.2.3 4 x 4 Differential Space-Time Modulation

Differential space-time modulation is extensible to any $M_T \times M_R$ configuration where a suitable group code can be found. For the 2×2 case, the QPSK constellation allows the 2×2 unitary group code mentioned in the previous section. However, for the 4×4 case, QPSK will not work. Another constellation is needed to fulfill the requirements for a 4×4 unitary group code. It has been proven that 16QAM fulfills this requirement [35]. Although 16QAM is not possible on the current real-time platform, the 4×4 differential space-time decoding algorithm can be implemented to measure timing and performance.

Basic Operation of the 4 x 4 Differential Space-Time Code

For 4×4 differential space-time modulation the previous discussion must be augmented with a general method to find unitary group codes for different numbers of transmit and receive antennas, as discussed in [35]. For a 4×4 antenna system using 16QAM, a unitary group code of length $L = 16$ is needed. The l^{th} signal in the constellation has the form

$$G_l = \frac{1}{\sqrt{2}} \begin{bmatrix} e^{j(2\pi/L)u_1 l} & 0 & 0 & 0 \\ 0 & e^{j(2\pi/L)u_2 l} & 0 & 0 \\ 0 & 0 & e^{j(2\pi/L)u_3 l} & 0 \\ 0 & 0 & 0 & e^{j(2\pi/L)u_4 l} \end{bmatrix}. \tag{4.14}$$

For $l = 0, ..., L-1$ and $u_1 = 1, u_2 = 3, u_3 = 5, u_4 = 7$, where u_m is derived in [35]. 16 unitary matrices are formed that are sent out in an order determined by the transmit data. In the τ^{th} block of data, antenna m transmits at time $t = \tau M + m$ a symbol that is differentially phased shifted by $(2\pi/L)u_m l$ relative to its previous transmission.

The differential space-time decoder is the same as illustrated in Figure 4.4 with the exception that there are now 16 $\mathbf{G}$ matrices that must be used to find the maximum value $G_k \in \mathtt{G}$. The difference between the 2×2 and the 4×4 differential space-time decoder is the dimensions of the matrices $\mathbf{Y}$ and $\mathbf{G}$ and the number of $\mathbf{G}$ matrices that are needed to find the maximum of $G \in \mathtt{G}$.

4.3 Summary

The Alamouti, 2×2 and 4×4 differential space-time decoding algorithms have been examined in sufficient detail to allow them to be implemented in software. Each decoding algorithm will be implemented using multiple processors. The inter-processor communication for each decoding algorithm will be realized using the parallel processing topologies already discussed.

Chapter 5

Space-Time Algorithms and Parallel Processing

There are many different ways to partition computational loads and exploit the parallelism that exists in computer applications. Much research has been conducted to determine those hardware and software organizations that best fit general purpose parallel processing. Other significant efforts have concentrated on speeding up the solutions of specific problems on special purpose computing systems [30]. This chapter discusses how to implement the common parallel processing topologies discussed in Chapter 3 with the space-time decoding algorithms explored in Chapter 4.

5.1 Benchmarks

Processor computation and inter-processor communication benchmarks have been run on the test system. These benchmarks provide the information needed to determine the performance of each parallel processing topology. A full guide to the benchmarks can be found in Appendix A.

Table 5.1 summarizes the benchmarks run for each different method of processor-to-processor communication. The maximum throughput possible is shown. These numbers are the result of using the maximum block size available for data transfers and are lower depending on the overhead associated with computing smaller blocks of data. The second table, Table 5.2, shows the throughput at which common algebraic functions perform on the DSPs. Notice the throughput difference when these calculations are run from and stored on on-chip IDRAM memory versus off-chip SDRAM memory. When the data is stored in the large, but slower off-chip local SDRAM memory, there is less throughput. Performance for 16K word and 256 word cases are shown to highlight the overhead associated with preparing and using the processor

Table 5.1: Inter-Processor Communication Benchmark Comparison

DSP to DSP Communication Method	Maximum Performance (MB/s)
IPBIFO Write	285
IPBIFO Read	285
VME bus Write	17
VME bus Read	35
Raceway bus Write	128
Raceway bus Read	29
Global Memory Write	101
Global Memory Read	49

and memory for these different block sizes. It is always advantageous to compute and/or transfer the largest block size possible so as to minimize overhead. All of these benchmarks were taken on a 300 MHz TI DSP processor.

Table 5.2: DSP Benchmark Comparison (MB/s)

DSP Process	256 Words		16K Words	
	IDRAM	SDRAM	IDRAM	SDRAM
Addition	33.4	8.7	33.5	8.7
Subtraction	33.4	8.7	33.5	8.7
Multiplication	29.9	8.5	30.0	8.5
Division	19.6	7.4	19.7	7.4

5.2 Assumptions and Methods

The space-time encoding algorithms on the transmitter will not be discussed because pre-processing on the host PC can effectively eliminate the need for real-time encoding. Space-Time symbol blocks can simply be pre-loaded in memory for transmitter read-out. Because of the ongoing research in the lab concerning MIMO symbol timing, the real-time aspects of a complete MIMO system will not be discussed. However, real-time analysis of space-time decoding algorithms on the receiver has been performed. Channel detection will also not be discussed as it is assumed that channel information is available as needed.

Because of the assumption that symbol timing issues and other communications related processing (i.e., matched filtering) have already taken place, a certain amount of processing power will be reserved. For simplicity, it is assumed that one processor will be given the responsibility of all communications-related processing for each channel on the receiver. This assumption is based on the benchmarks from the complex FIR filter needed for the matched filtering of a QPSK signal. In matched filtering there exists a tradeoff between accuracy and speed, which can be adjusted according to need and application. The higher the "tap" value H, the better the resolution, but the more processor intensive the matched filtering. The lower the H value, the less reliable the results (more noise) but the faster the processing. One processor set aside primarily for matched filtering and any other additional functions, per antenna, is a safe and simple assumption that leaves computational room for channel synchronization, experimentation, and future processing requirements.

The following equation will be used to determine the number of cycles each data word takes to compute:

$$\frac{1.2 \times 10^9}{\alpha\left(\frac{MB}{s}\right)} = \beta\left(\frac{clock \quad cycles}{word}\right) \tag{5.1}$$

where α is the benchmark in megabytes per second and β is the resultant number of processor clocks cycles per word data processed. This will be used to determine the

processing required for each space-time algorithm when using benchmarks. Equation 5.1 is true only for 300 MHz processors.

For the narrow band case, a symbol rate in the 10's of kilo symbols per second will be considered sufficient. The symbol rate directly affects the number of processors needed for each space-time decoding as well as the processing power needed for the communications processors, especially for matched filtering.

5.3 Alamouti Parallel Processing

The Alamouti space-time decoder processing can be characterized by expanding Equation 4.6 to yield

$$\tilde{s_0} = h_{00}^* r_{00} + h_{01} r_{01}^* + h_{10}^* r_{10} + h_{11} r_{11}^* \tag{5.2}$$

and

$$\tilde{s_1} = h_{01}^* r_{00} - h_{00} r_{01}^* + h_{11}^* r_{10} - h_{10} r_{11}^*. \tag{5.3}$$

It is assumed that $\mathbf{H}$ is available from a channel estimation phase. From Equations 5.2 and 5.3 it can be seen that no cross-channel multiplies need to be performed on data that is received on different transmitter nodes. For example, one receiver can perform the $h_{00}^* r_{00}$ and the $h_{01} r_{01}^*$ operations from Equation 5.2. At the same time the other receiver can perform the $h_{10}^* r_{10}$ and the $h_{11} r_{11}^*$ operations from the same equation. If necessary, the only inter-processor communication needed is the combining of the individual parts of Equations 5.2 and 5.3 by addition.

To determine the number of processors that are needed for the Alamouti decode, the number of multiplies and adds in the algorithm are needed. Using Equations 5.2 and 5.3, it can be determined that 16 multiplies and 6 additions are required for each data word processed. (Recall that each word contains both real and imaginary data that must be multiplied and added separately. Moreover, 2 words of data per symbol will be used when determining the number of symbols per second the processor can handle.)

The total number of processor cycles required for the Alamouti decoding can be estimated by the following equation:

$$\alpha\left(\frac{cycles}{word}\right) = 16\left(\frac{multiplies}{word}\right)40\left(\frac{cycles}{multiply}\right)16K\left(\frac{words}{buffer}\right) + 6\left(\frac{additions}{word}\right)36\left(\frac{cycles}{addition}\right)16K\left(\frac{words}{buffer}\right) \tag{5.4}$$

where α is the total number of processor cycles required for each 16K buffer full of data from the communications processor - about 14 million processor cycles per 16K buffer of data. About 300 million processor cycles are available every second from the processor, so

$$\beta = \frac{300 \times 10^6}{14 \times 10^6}16K \tag{5.5}$$

where β equals almost 343K symbols per second. This is more than enough symbols per second for the narrow band case and thus only one processor is required for Alamouti decoding.

Only the bus, line/ring, grid, and star topologies will be discussed. The hypercube, binary tree, and pyramid topologies are not required due to the low processing requirements of the Alamouti algorithm.

5.3.1 Bus

Figure 5.1 illustrates the proposed method of decoding for the Alamouti scheme using a bus topology. Wireless data is first processed by each of the communications processors (labeled by their functions: matched filtering, timing, interpolation, etc.) and then sent over the bus to the third processor responsible for Alamouti space-time decoding. Because of the architecture, only one processor can transfer data over the bus at a time. Notice that it makes no difference which processor the antennas are connected to because the bus topology allows for the same communication time between each processor. Bus transfers will be timed by writing and reading from global memory over the VME bus.

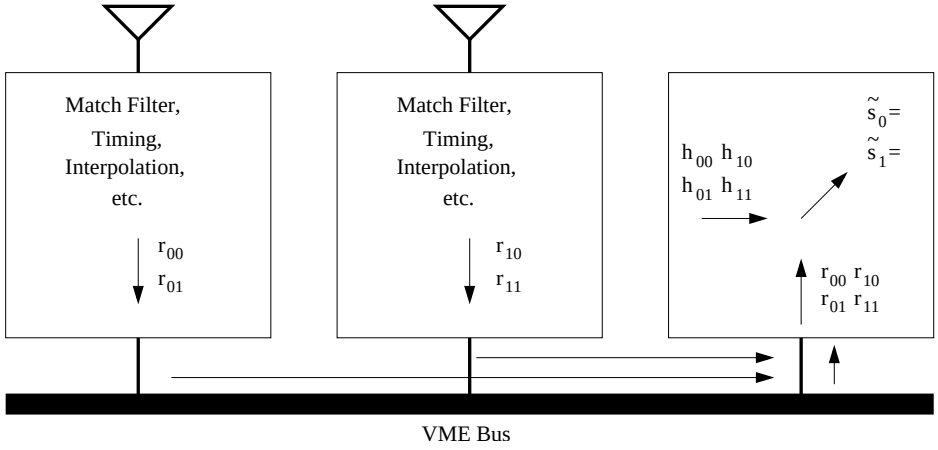

Figure 5.1: Proposed Bus Parallel Processing

5.3.2 Line/Ring

As discussed above, only one processor is needed for Alamouti decoding. Figure 5.2 illustrates the proposed connection of the processors with their respective processes. A full ring architecture is not necessary because only three processors are needed and all three do not need to communicate with each other. Processor communication only needs to go one way from both communications processors (labeled the same as in the bus section) to the processor performing the Alamouti decoding. A line architecture will suffice. This topology will provide a speedup over the bus topology as processor-to-processor communication over the IPBIFOs is much quicker than over the global bus (i.e., through global memory). However, a delay will be added due to the board's DSP-to-DSP interconnections. For one of the communications processors, there does not exist a straight path to the assigned Alamouti decoding processor. Two options are available, one communications processor can write to global memory and the decoding processor can retrieve it from global memory, or the data may be passed through the other communication processor to the Alamouti decoding processor.

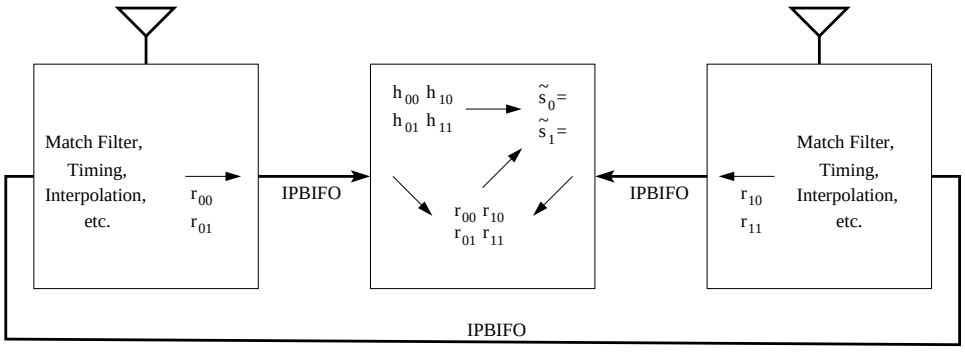

Figure 5.2: Proposed Line/Ring Parallel Processing

5.3.3 Grid/Mesh

In a four processor system, the grid topology is the same as a 4 processor ring topology. Because only 3 processors are needed, the grid topology does not make sense. For this reason the grid topology will not be discussed any farther.

5.3.4 Star

A three processor star topology is essentially the same as the line/ring topology previously discussed. For this reason the star topology will not be discussed further and the results obtained in real-time experiments of the ring topology will be considered sufficient.

5.4 2 x 2 Differential Space-Time Parallel Processing

The 2×2 differential space-time algorithm can be characterized by Equation 4.13:

$$\tilde{G_k} = \arg \max_{G \in \mathbb{G}} \mathrm{Re}\{\mathrm{Tr}\{G Y_k^H Y_{k-1}\}\} \tag{5.6}$$

(repeated here as Equation 5.6 for convenience), and as shown in Figure 4.4 from the previous chapter.

To better understand this equation we need to discuss the dimensions of each matrix and understand where each variable of the equation comes from. Y_k is a 2×2 matrix that contains the signal information obtained from both receive antennas at

times t and $t+T$, similar to the Alamouti receive matrix as shown in Table 4.3. Y_{k-1} is simply the time delayed version of Y_k.

Matrix $\mathbf{G}$ is one of the 2×2 matrices of the quaternion group as discussed in Chapter 4 and shown in Equation 4.11. Because we need to find only the trace of $GY_k^* Y_{k-1}$, we do not need to multiply out all of the values of $\mathbf{G}$. Only the diagonal elements are needed, thus reducing computation.

Because there are 8 members of the quaternion group, there will need to be 8 different calculations of $\mathrm{Tr}\{GY_k^* Y_{k-1}\}$. To find the maximum value of the $\mathrm{Tr}\{GY_k^* Y_{k-1}\}$ equation, a TI optimized assembly routine is used that returns the maximum value of an input vector. Certain restrictions apply to the code that must be taken into consideration when using this assembly code routine. TI has made available two optimized assembly routines that return either the maximum value of a vector (if the vector length is multiple of 4 and greater than or equal to 16) or a pointer to the element that is found to be the maximum value of a vector (if the vector length is a multiple of 3 and greater than or equal to 9).

Like the Alamouti decoder, this decoder is expanded to see where information is coming from, how many multiplies and adds/subtracts are required, and to visualize the flow of data:

$$\begin{bmatrix} r_{02} & r_{12} \\ r_{03} & r_{13} \end{bmatrix}^H \begin{bmatrix} r_{00} & r_{10} \\ r_{01} & r_{11} \end{bmatrix} = \begin{bmatrix} r_{02}^* r_{00} + r_{03}^* r_{01} & r_{02}^* r_{10} + r_{03}^* r_{11} \\ r_{12}^* r_{00} + r_{13}^* r_{01} & r_{12}^* r_{10} + r_{13}^* r_{11} \end{bmatrix}. \tag{5.7}$$

In Equation 5.7, r_{ij} is the received signal from antenna i, where $i = 0, 1$, at time instance j, where $j = 0, 1$ apply to Y_{k-1} and $j = 2, 3$ apply to Y_k.

As seen in Equation 5.7 the diagonal elements of $Y_k^H Y_{k-1}$ come from the same antenna, whereas the off-diagonal elements involve cross-processor multiplies and additions. The $\mathbf{G}$ matrices need not be discussed in greater detail as they are known before hand and can be made available on any processor that requires them. As seen in Equation 5.7, eight multiplications and three additions are required per individual $\mathbf{G}$ matrix. For each, 64 multiplications and 24 additions are needed for each symbol decision.

Once again, as these calculations do not take into account the real and imaginary parts of the symbol, the total number of multiplications equals 256 multiplications plus 128 additions and the total number of additions equals 48. The total number of processor cycles required for the 2×2 differential decoding can be estimated by the following equation:

$$\begin{aligned}\alpha\left(\frac{cycles}{word}\right) = \; & 256\left(\frac{multiplies}{word}\right)40\left(\frac{cycles}{multiply}\right)16K\left(\frac{words}{buffer}\right) + \\ & 128\left(\frac{additions}{word}\right)36\left(\frac{cycles}{addition}\right)16K\left(\frac{words}{buffer}\right) + \\ & 48\left(\frac{additions}{word}\right)36\left(\frac{cycles}{addition}\right)16K\left(\frac{words}{buffer}\right)\end{aligned} \tag{5.8}$$

where α is the total number of processor cycles required for each 16K buffer full of data from the communications processor, approximately 265 million processor cycles per 16K buffer of data. About 300 million processor cycles are available every second from the processor, so

$$\beta = \frac{300\times10^6}{265\times10^6}16K \tag{5.9}$$

where β equals approximately 18K symbols per second. This could be considered enough symbols per second for the narrow band case. However, this does not take into account the process of finding the maximum value of the Re Tr$\{GY_k^*Y_{k-1}\}$, the overhead of running tasks, and transferring data between memory locations. Two processors will be used for 2×2 differential decoding to ensure the computational power required.

Only the bus, line/ring, grid, and star topologies will be discussed. Like the Alamouti decoder, the hypercube, binary tree, and pyramid topologies are too complex for decoding the 2×2 case and the additional processors are not needed.

5.4.1 Bus

Figure 5.3 illustrates the proposed method of implementing the differential space-time algorithm using a bus topology. Wireless data is first processed by the

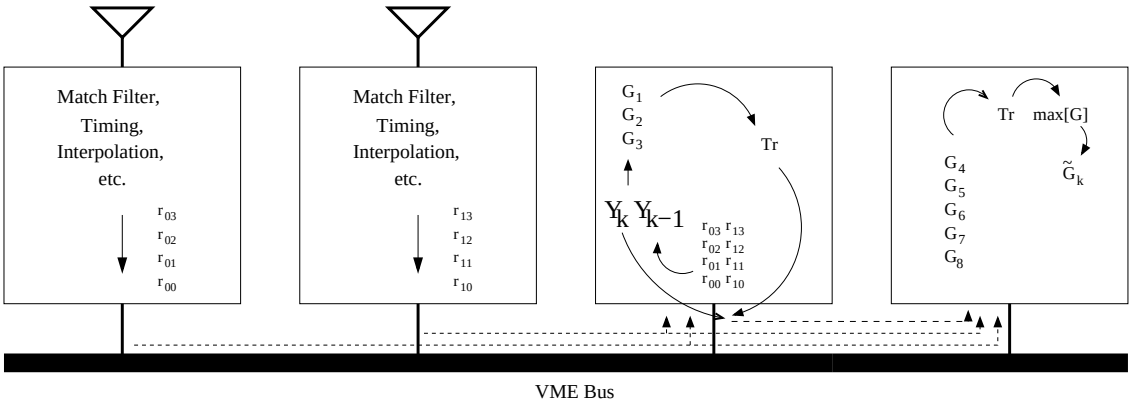

Figure 5.3: Proposed Bus Architecture

two communications processors and then sent over the bus to the remaining two processors responsible for differential decoding. The decoding has been partitioned into two parts in an effort to share the processing power between the two available decoding processors.

The first decoding processor will solve for each $Y_k^* Y_{k-1}$ by combining the data received from both communications processors and then $\mathrm{Tr}\{GY_k^* Y_{k-1}\}$ where $k = 1, 2, 3$. This processor will then transfer all solved data over to the other decoding processor which will find the remaining five $\mathrm{Tr}\{GY_k^* Y_{k-1}\}$ where $k = 4, ..., 8$. This processor will also be responsible for finding the maximum value of the $\mathbf{g}$ vector formed by solving the $\mathrm{Tr}\{GY_k^* Y_{k-1}\}$ equation. Once again, notice that it makes no difference which processor acts as the communications processor as the bus topology allows for the same communication time between all processors.

5.4.2 Line/Ring

Figure 5.4 illustrates the proposed method of implementing the differential algorithm using a ring architecture. When using a ring topology, there are two options as to where the antenna may be placed in the ring. The first is placing the antennas on every other processor as shown in Figure 5.4. The second option would be to have the antennas connected to adjacent processors. The designs are very similar, but the method chosen, as illustrated in Figure 5.4, requires only one instance of having to

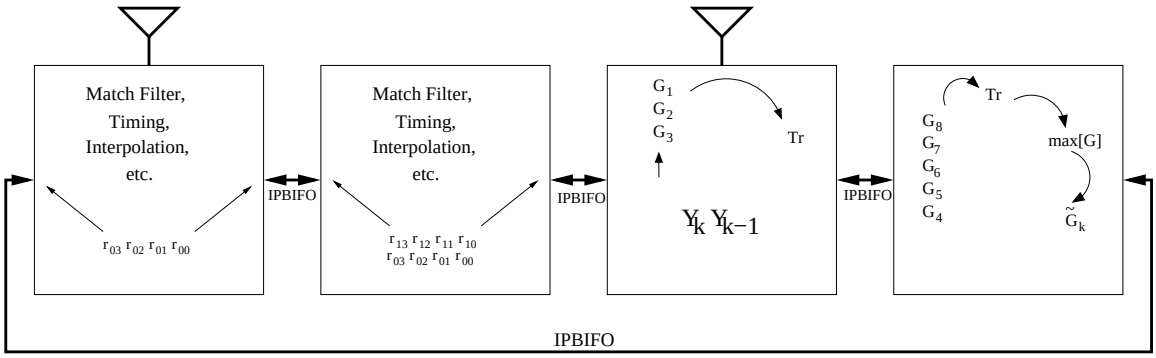

Figure 5.4: Proposed Line/Ring Architecture

pass data through a processor without that processor acting upon the data. The partitioning of the algorithm is the same as in the bus case.

5.4.3 Grid/Mesh

As the discussed in the Alamouti section, the grid topology is the same as a 4 processor ring topology. For this reason the grid topology will not be further discussed and the results obtained in the real-time experiments of the ring topology will be considered equal to what the results of the grid topology would have been.

5.4.4 Star

Figure 5.5 illustrates the proposed method of implementing the differential algorithm using a star architecture. The two communications processors are placed as child processors. The partitioning of the algorithm is the same as in the other 2×2 cases. Their is one virtual IPBIFO connection as illustrated and explained in Figure 5.5. The central processor was chosen to compute the first half of the differential algorithm so that there is no additional communication overhead by having to pass data back and forth from the central processor to the other child decoding processor. The central processor does not end up with the final results in this case.

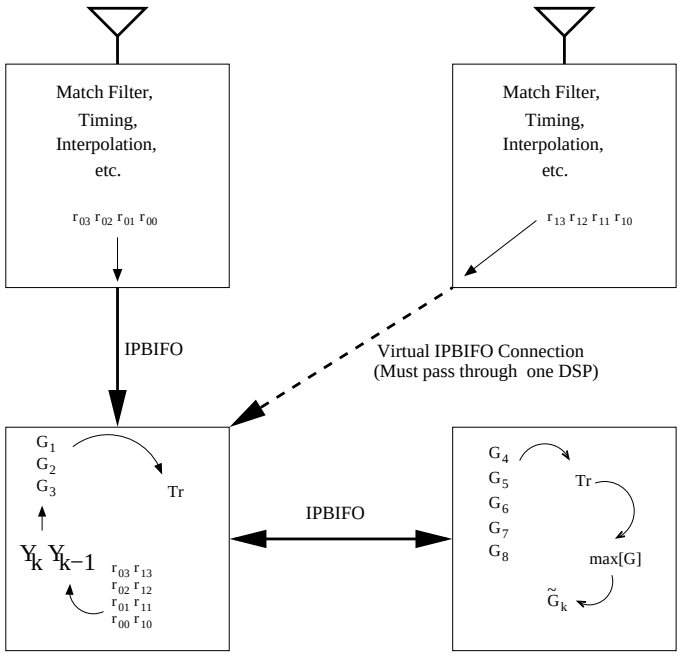

Figure 5.5: Proposed Star Parallel Processing

5.5 4 x 4 Differential Space-Time Parallel Processing

The 4×4 differential space-time algorithm can also be characterized by Equation 4.13, as shown in Figure 4.4. However, the dimensions of the matrices change from 2×2 to 4×4. Y_k is now a 4×4 matrix that contains the signal information obtained from 4 receive antennas at times t, $t + T$, $t + 2T$, and $t + 3T$. Y_{k-1} is still the time delayed version of Y_k.

Matrix $\mathbf{G}$ is now one of the sixteen 4×4 matrices of the unitary group code discussed in Chapter 4. We still only need to find the trace of $GY_k^* Y_{k-1}$, so we do not need to multiply out all of the values of G. Like the 2×2 differential case, the $\mathbf{G}$ matrices need not be analyzed in greater detail as they are known before-hand and can be made readily available on any processor that requires them. The TI optimized assembly routine will again be used that returns the maximum value of an input vector. The assembly routine's restrictions can be ignored for the 16x1 vector case as it will return the maximum value.

This decoder can be multiplied out to visualize where information is coming from, and the calculations that need to take place:

$$\begin{bmatrix} r_{04} & r_{14} & r_{24} & r_{34} \\ r_{05} & r_{15} & r_{25} & r_{35} \\ r_{06} & r_{16} & r_{26} & r_{36} \\ r_{07} & r_{17} & r_{27} & r_{37} \end{bmatrix}^H \begin{bmatrix} r_{00} & r_{10} & r_{20} & r_{30} \\ r_{01} & r_{11} & r_{21} & r_{31} \\ r_{02} & r_{12} & r_{22} & r_{32} \\ r_{03} & r_{13} & r_{23} & r_{33} \end{bmatrix} = \begin{bmatrix} r_A & r_B & r_C & r_D \\ r_E & r_F & r_G & r_H \\ r_I & r_J & r_K & r_L \\ r_M & r_N & r_O & r_P \end{bmatrix} \tag{5.10}$$

where

$$r_A = r_{04}^* r_{00} + r_{05}^* r_{01} + r_{06}^* r_{02} + r_{07}^* r_{03}, \tag{5.11}$$

$$r_F = r_{14}^* r_{10} + r_{15}^* r_{11} + r_{16}^* r_{12} + r_{17}^* r_{13}, \tag{5.12}$$

$$r_K = r_{24}^* r_{20} + r_{25}^* r_{21} + r_{26}^* r_{22} + r_{27}^* r_{23}, \tag{5.13}$$

and

$$r_P = r_{34}^* r_{30} + r_{35}^* r_{31} + r_{36}^* r_{32} + r_{37}^* r_{33}. \tag{5.14}$$

In Equation 5.10, r_{ij} is the received signal from antenna i, where $i = 0, 1, 2, 3$, at time instance j, where $j = 0, 1, 2, 3$ apply to Y_{k-1} and $j = 4, 5, 6, 7$ apply to Y_k.

Just like the 2×2 differential case, and as seen in Equations 5.11, 5.12, 5.13, and 5.14, the diagonal elements of $Y_k^H Y_{k-1}$ come from the same antenna, whereas the off-diagonal elements involve cross-processor multiplications and additions. Also notice that only cross-processor multiplications and additions are needed for the off-diagonal elements across 2 processors, not across all 4. This can be taken advantage of when using parallel processing.

As deduced from Equations 5.11, 5.12, 5.13, and 5.14, for each symbol or I and Q value, 384 multiplications and 48 additions are needed. Once again, these calculations do not take into account the real and imaginary parts of the symbol, so the total number of operations equals 1536 multiplications plus 768 additions and the total number of additions equals 96. The total number of processor cycles required

for the 4×4 differential decoding can be determined by the following equation:

$$\begin{aligned}\alpha\left(\frac{cycles}{word}\right) = \; & 1,536\left(\frac{multiplies}{word}\right)40\left(\frac{cycles}{multiply}\right)16K\left(\frac{words}{buffer}\right) + \\ & 768\left(\frac{additions}{word}\right)36\left(\frac{cycles}{addition}\right)16K\left(\frac{words}{buffer}\right) + \\ & 96\left(\frac{additions}{word}\right)36\left(\frac{cycles}{addition}\right)16K\left(\frac{words}{buffer}\right)\end{aligned} \tag{5.15}$$

where α is the total number of processor cycles required for each 16K buffer full of data from the communications processor, almost 1.5 billion processor cycles per 16K buffer of data. About 300 million processor cycles are available every second from the processor, so

$$\beta = \frac{300 \times 10^6}{1.5 \times 10^9}16K \tag{5.16}$$

where β equals approximately 0.2K symbols per second. Thus a single processor can decode 0.2 x 16K $\approx$ 3400 symbols per second. This is obviously not sufficient bandwidth for even narrow band communications, so 6 decoding processors will be used to decode almost 20K symbols per second leaving a little room for overhead and the maximum value function.

5.5.1 Bus

Figure 5.6 illustrates the proposed method of implementing the differential algorithm using a bus architecture. Like the 2×2 differential case, data is processed by the four communications processors and then sent over the VME bus to the remaining processors responsible for differential decoding. The decoding has been partitioned among all of the remaining processors with the goal of using similar amounts of processing power on each processor.

There are 6 processors available for decoding. One processor will calculate the diagonals, r_A, r_F, r_K, and r_P. Four processors will split up the 16 $GY_k^*Y_{k-1}$ calculations. The remaining processor will be responsible for the maximum value calculation and would be available for additional computations. The maximum value

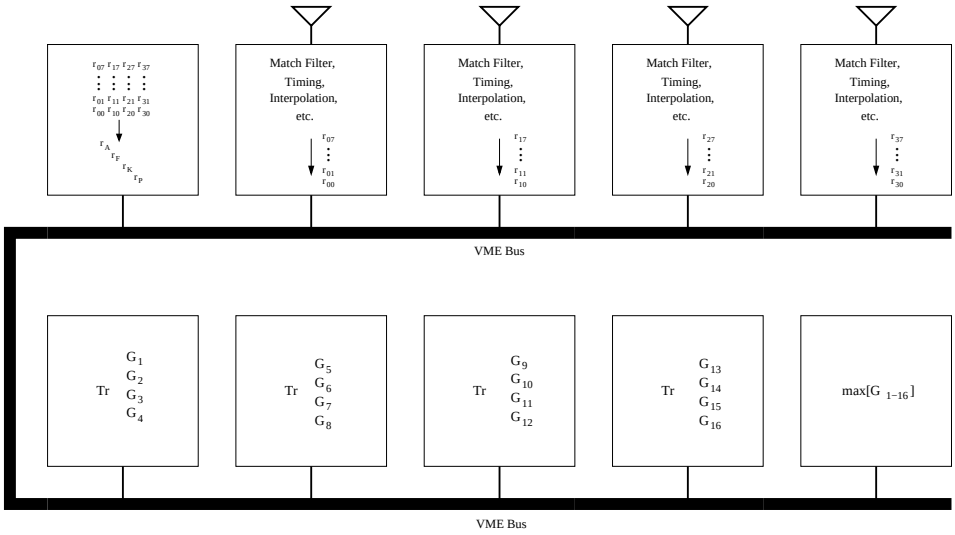

Figure 5.6: Proposed Bus Parallel Processing

function will not require the complete processing power of one DSP and can be used for additional tasks, for example, communicating with the host computer, post processing, or formatting of data.

5.5.2 Line/Ring

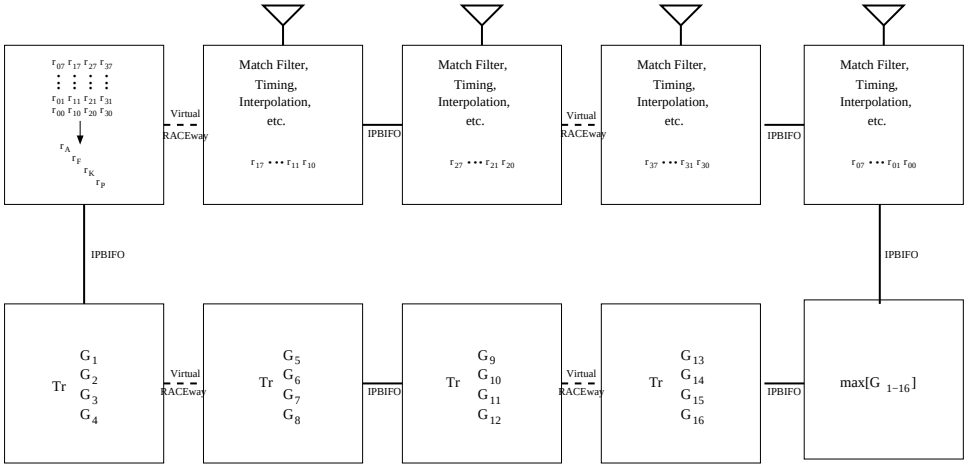

Figure 5.7: Proposed Ring Parallel Processing

Figure 5.7 illustrates the proposed method of implementing the differential algorithm using a line/ring architecture. The partitioning of the algorithm is the same as in the bus case with the only difference being the inter-processor communication paths taken. Because of the inherent pipelining nature of the ring topology, once data has been collected from the communications processors, data may be pipelined through until all computations are finished.

The antennas are placed on one side of the line/ring topology so that the only inter-processor communication overhead comes when trying to communicate the received data to the decoding processors. Data are then easily pipelined through all of the computational processors. A ring architecture isn't really needed as a unidirectional line topology would suffice for this method of decoding.

The placement of the communications processor and decoding processors in the line/ring was determined by matching the hardware paths available in the real-time system as closely as possible to minimize virtual connections. The virtual connections in Figure 5.7 are illustrated as dashed lines and have the actual communication path shown. The four rightmost processors are all connected by IPBIFO connections and are thus all on one board. The second and third processors on each row are also contained on one board. The two leftmost processors would therefore be on a third board. Any IPBIFO connection guarantees that the processors connected must share the same board. The virtual connections between boards are labeled virtual RACEway and thus use the RACEway bus for board-to-board communication.

5.5.3 Grid/Mesh

Figure 5.8 illustrates the proposed method of implementing the differential algorithm using the grid architecture. The decoding of the data is partitioned the same as in the previous 4×4 topologies. There are numerous ways to move data around using a 2×5 grid topology, however only the method shown in Figure 5.8 will be analyzed. This was determined by trying to match the hardware paths actually available in the real-time system and thus minimize any virtual connections. It can be seen in the figure that the leftmost four processors all connect to each other using

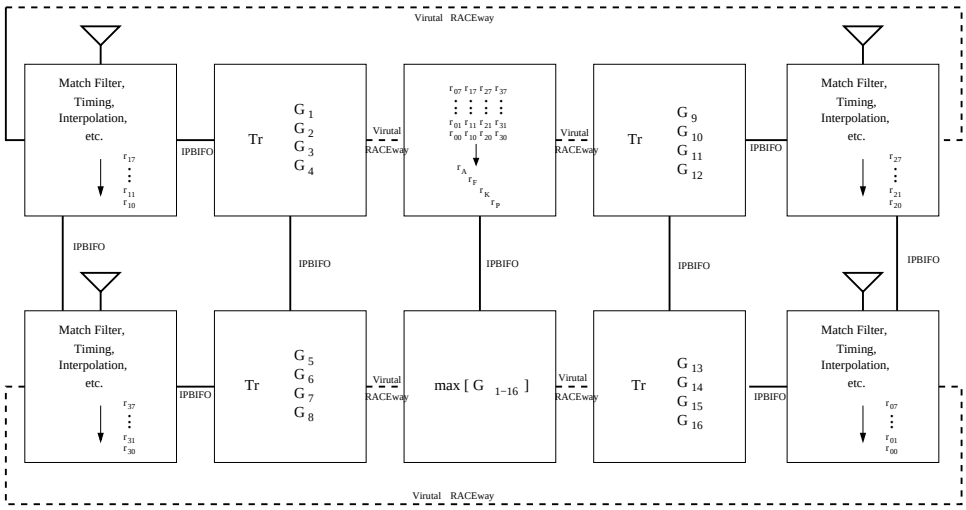

Figure 5.8: Proposed Grid Parallel Processing

IPBIFOs. These processors are on one board. The same holds true for the four rightmost processors. The middle two processors must then be on another board. The virtual connections between boards are labeled virtual RACEway and also use the RACEway bus for board-to-board communication.

5.5.4 Star

Figure 5.9 illustrates the proposed method of implementing the differential algorithm using a star architecture. The data is again partitioned the same as in the previous 4×4 topologies. The central processor is responsible for collecting the data from the communications processors and dividing it among the remaining processors for decoding. The central processor must also coordinate all of the data passing that must take place for the overall decoding algorithm.

Since the star topology has one central processor that all other processors communicate with, almost all of the processor-to-processor communication must be modeled using virtual RACEway connections. There is one processor that is able to communicate with the central processor using an IPBIFO connection and is on the same board. There are three DSP boards used in this topology; however, due to

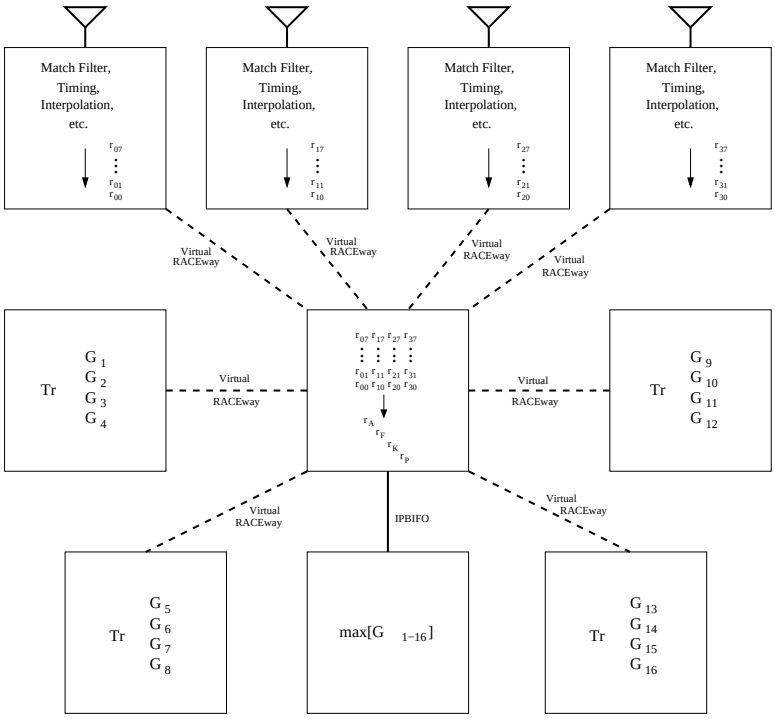

Figure 5.9: Proposed Star Parallel Processing

topology constraints, processors that are right next to each other will not be able to communicate directly.

5.5.5 Hypercube

Figure 5.10 illustrates the proposed method of implementing the differential algorithm using a hypercube architecture. Hypercube topologies only work with a number of processors equal to a power of 2. For this reason, only 8 processors will be used for the hypercube topology in our system. If more processors are needed, then the hypercube topology requires 16 processors. Although we do have enough processors on our system to simulate a 16 processor decoder, there would be too many wasted clock cycles for the narrow band case.

The communications processors are placed on one side of the hypercube. The data is then passed over to the other side of the topology where a four processor ring is formed to compute the decoding algorithm. Since only 4 processors are available to

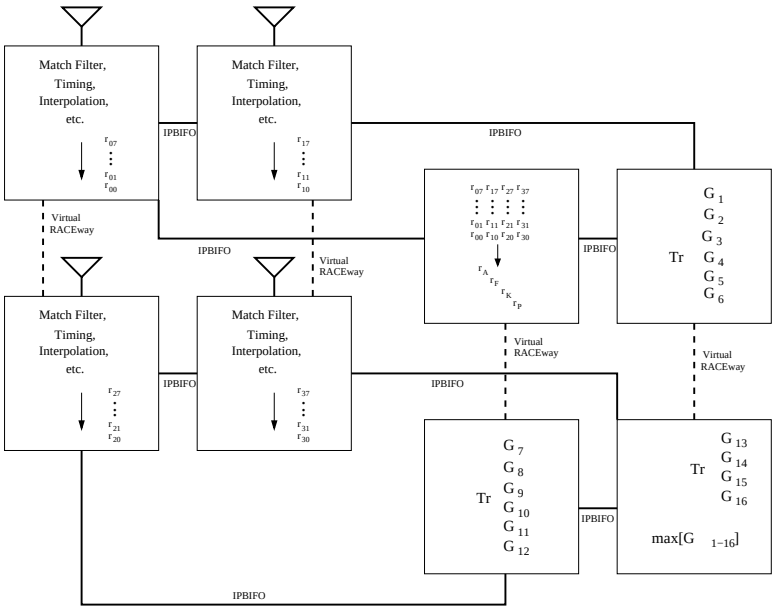

Figure 5.10: Proposed Hypercube Parallel Processing

compute the decoding algorithm (compared to six for the previously mentioned 4×4 topologies), the partitioning of the decoding process must be changed. One processor will still compute all of the diagonal elements but now the remaining three processors will compute the remainder of the decoding algorithm.

Only two boards are needed as the topology requires only eight processors. As constrained by hardware, there are two communications processor per board and two algorithm decoding processors per board. The boards are connected via virtual RACEway connections. This topology and its processor placement can be imagined by placing the two boards on top of one another and connecting each processor to the processor above or below it.

5.5.6 Binary Tree

Figure 5.11 illustrates the proposed method of implementing the differential algorithm using a binary tree architecture. Similar to the hypercube, this topology requires less than the number of processors needed to maintain the desired throughput discussed in the introduction of Section 5.5. A binary tree topology requires that the

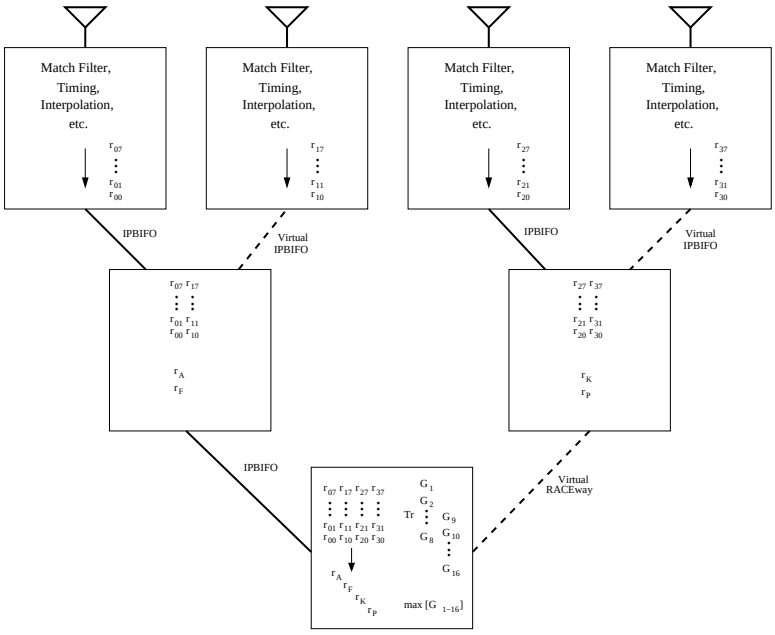

Figure 5.11: Proposed Binary Tree Parallel Processing

number of processors must be a power of 2 minus one. Unless 15 processors are used with wasted clock cycles, 7 processors (instead of 10) must now do the job of the communication and decoding tasks.

Since there are four communications processors, they fit nicely at the top of the tree as shown in Figure 5.11. Data is passed down the tree and does not all come together to be processed until it reaches the root of the tree. Minimal processing is able to be done on the intermediate layer of the tree and the root processor is required to do almost the majority of the processing. Even if 15 processors were used, all of the data would still have to be passed down the tree before it could be processed. With 15 processors, data could be passed up the other half branch of the tree, but with additional inter-processor communication overhead and wasted processor cycles.

Again, only two boards are needed to accommodate 7 processors. One half of the tree plus the root processor is contained on one board and the other top half of the tree is on the other board. Virtual IPBIFO transfers (inter-processor communication through another processor or through global memory transfers) take care of processors that are not directly connected. One virtual RACEway connection

is used to communicate between boards, between the right top half of the processors in Figure 5.11 and the root processor contained on the other board.

5.5.7 Pyramid

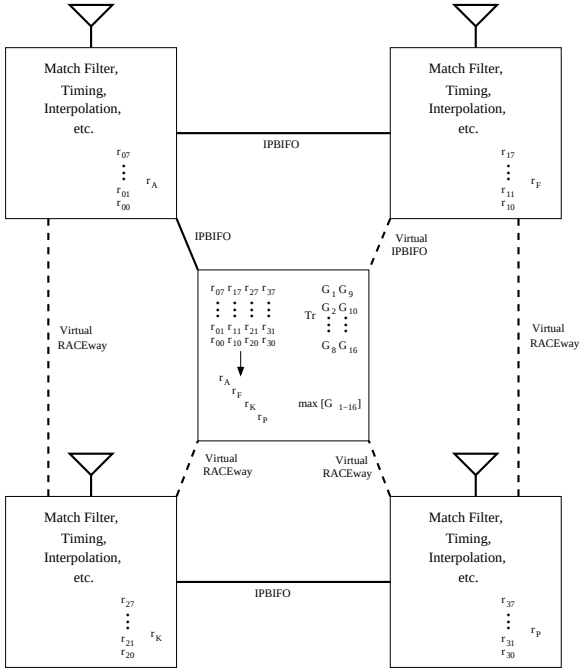

Figure 5.12: Proposed Pyramid Parallel Processing

Figure 5.12 illustrates the proposed method of implementing the differential algorithm using a pyramid architecture. A problem with the pyramid topology, similar to the binary tree and hypercube, is that only 5 processors may be used for the decoding algorithm unless 21 processors are available. Twenty one processors are not available on our system, so 5 must be used.

Data is simply passed from the communications processors, aggregated, and then processed on the top of the pyramid. Three of the processors come from one board and the other two come from the other board. Virtual RACEway connections

are used to communicate across boards and virtual IPBIFO connections are used to communicate onboard when two processors are not directly connected.

Like the binary tree topology, a single processor is left to finish the decoding process. The hypercube doesn't appear to be beneficial to us as the links between processors at the base of the pyramid are not used and the topology simply mimics a five processor star topology with a very powerful primary/central processor that is not physically available.

5.6 Summary

The Alamouti, 2×2 and 4×4 differential space-time decoding algorithms have been mapped out using common parallel processing topologies. The final step is to implement the decoding algorithms on the real-time system and benchmark their performance. Combining the decoding algorithm benchmarks with the inter-processor communication benchmarks will give the total processor cycles required for each decoding algorithm using each parallel processing topology.

Chapter 6

Results

Each space-time decoding algorithm has been implemented on DSP processors. As discussed in Chapter 4, the 4×4 differential space-time decoding algorithm would not work at this point (because we are using QPSK) on the real-time equipment. However, using benchmarks and programming the algorithm on the DSPs will allow us to determine the performance of the different parallel processing topologies and the decoding algorithm in general. This chapter describes the implementation details of each algorithm and the results obtained from real-time benchmarks.

6.1 Methodology

Each space-time decoding algorithm was broken down into small specific roles as discussed and illustrated in Chapters 4 and 5. These roles directly relate to the software tasks running on the real-time operating system (RTOS). These tasks were timed and using the timer mechanism found on the DSP processors. The DSP timing mechanism is restricted on a per software function basis, therefore all timing measurements were performed per RTOS software function (functions are refereed to as tasks by the TI DSP RTOS and we will as well for the remainder of this paper).

A software library was created during system benchmark testing that contains the RTOS tasks to transfer data from processor to processor. The MIMO decoding algorithm timing results supplement the benchmark testing results. The results of the different functional and RTOS tasks are accumulated per processor and then across all processors involved in the MIMO decoding algorithm. This methodology has the benefit of producing results that can be extrapolated to larger systems.

As previously discussed, symbol timing, training data, and other issues have not been completely solved to allow for true MIMO transmission and reception. This means that the data being processed during the decoding algorithms is not actual data being sent from a transmitter and processed, but random data to take its place. Because these tests are concerned with the real-time aspects of the chosen space-time algorithms, and not the actual transmission and reception using these algorithms, these tests fulfil their purpose.

The transfer of data from processor to processor is always assumed to use the maximum block size. This minimizes the impact of data transfer overhead and maximizes throughput. For example, data can be passed from processor to processor via the inter-processor BIFIFO (IPBIFO). Data can be transferred in 4 byte (or one word) increments. Benchmark testing has shown that throughput is optimized by transferring the full IPBIFO (32 kilobytes) at a time. The maximum block size for IPBIFO transfers is thus 32 kilobytes.

On-chip RAM will always be the preferred location for computations. However, since its size is very limited, data must be buffered to and from the DSP processor. Local SDRAM will always be preferred for the buffering of data as it is very large and second in speed only to the on-chip RAM. Global memory will be the last place used for storing results, but it must be used frequently for all board-to-board communication. All of the final results will be placed in global memory to simulate the real life scenario where the results must be passed onto a host processor board for interpretation.

Texas Instruments has optimized assembly code for a variety of common DSP functions. Use will be made of this optimized code for the FFT used in matched filtering and finding the maximum value in a vector for the 4×4 differential decoding.

The number of clock cycles may be converted to human readable form by the equation:

$$t = \frac{N}{\left(\frac{4}{300MHz}\right)} \tag{6.1}$$

where t equals the time in seconds and N equals the measured number of timing ticks. Notice that there are 4 processor cycles for every timing tick. All of the results will be presented in timing ticks and can easily be converted to processor clock cycles by multiplying by four. The timing ticker wraps around at 0xFFFFFFFF, so care was taken to ensure that the timer is not run too long. This is not a problem for these tests, but would be a concern for longer benchmarks.

On each processor, the tasks are divided into essentially two groups: one group of tasks that concern themselves with the actual space-time decoding algorithm and another group of tasks that take care of memory transfers, memory management, and inter-processor communication. The passing of information always makes use of the DMA co-processor to minimize CPU processing.

The implementation of each decoding algorithm makes use of the same tasks and is independent of the topology. The timing differences can only be attributed to the topologies and their method of interconnections between the processors and boards. The exceptions are for the hypercube, binary tree, and pyramid 4×4 differential space-time cases where a different number of processors are used due to topology constraints. These topologies conveniently replace inter-processor communication with a larger processor and are not directly comparable to the bus, line/ring, star, and grid 4×4 differential topologies. Even so, they are able to give insight into the benefits and performance gain that additional hardware (for example, if a specialized FPGA board was acquired for the BYU real-time system) could bring to the system.

A comparison of all of the topologies for each decoding algorithm is found at the end of the discussion of results. All of the results are normalized to the fastest topology (least number of total timing ticks). The topology with the least amount of total timing ticks is divided by itself to normalize to one. The other topologies are also divided by the same value. These results should enable the reader to quickly and accurately compare topologies without the use of a calculator. As mentioned, the exceptions are the hypercube, binary tree, and pyramid 4×4 differential space-time cases. These are normalized to the fastest bus, line/ring, star, or grid topology and

are less than one when the total timing ticks are less than best bus, line/ring, star, or grid topology. This is due to the forced reduction in the number of DSP processors used in those topologies. In some cases, it is not fair to directly compare the cycle counts due to reduced inter-processor communication due to a smaller number of processors. The normalization timing value is rounded to the hundreds or thousands place to enable discussion and comparison, not precision.

6.2 Communications Processors

The communications processors are responsible for setting up and initiating the real-time capture from the 6216, the RF frontend boards. These initialization steps are contained in a task called "main" and are not timed. Each processor also has a task named "taskHeartbeatLED" that is used as the idle task and for debugging purposes, and is also not timed.

The tasks that were timed and are the critical processor intensive tasks are the following.

- *taskWriteDataVME* - Responsible for the transfer of the output of the matched filter data to the space-time decoding processors. This task transfers data from the local SDRAM to the DSP board's global buffer, which is accessible from the VME bus for retrieval by the host computer for analysis. This task is used for the bus topology and any board-to-board transfers. The results from this task are derived from the inter-processor communication benchmarks found in Appendix A.

- *taskWriteDataIPBIFO* - Also responsible for the transfer of the output of the matched filter data to the space-time decoding processors. This task transfers data using a DMA memory transfer from the DSP's SDRAM to the IPBIFOs. This task is used for any processor-to-processor communication on the same board. The result is also derived from the benchmarks performed and can be found in Appendix A.

- *taskComplexFFT* - Responsible for the matched filter output by computing the complex FFT. This must be run twice to fill up the 16K IPBIFO. The number of taps used for the matched filtering process is 64. The number of filter taps was chosen as a compromise between accuracy and speed. The number of taps needed for actual MIMO transmission may differ. The result is also derived from the benchmarks performed and can be found in Appendix A.

Table 6.1 shows the number of processor timing ticks for each task on the communications processors. In Table 6.1 the $*$ refers to the fact that one or the other of the tasks will execute, but not both. This is due to the processor only communicating and passing the data via one communication path at a time depending on whether bus or IPBIFO transfers are required.

Table 6.1: Communications Processor Tasks & Timing Results

Task Name	Number of Ticks	Number of Times Used	Total
taskWriteDataVME	47,621	1*	47,621
taskWriteDataIPBIFO	20,211	1*	20,211
taskComplexFFT	537,508	2	1,075,016

6.3 Alamouti Real-Time Processing

The Alamouti algorithm has been implemented on one DSP processor as discussed in Section 5.3. The space-time decoding algorithm is the same for each case implemented with the exception of the inter-processor communication. Each processor has a task called "main" that is responsible for processor initialization and that is not timed. Each processor also has a task named "taskHeartbeatLED" that is used as the idle task and for debugging purposes and not timed.

The critical processor-intensive tasks that we timed are the following:

- *taskGetDataVME* - Responsible for the retrieval of the output of the matched filter data from the communications processors. This task transfers data from the VME bus to the DSP board's global buffer and then to the DSP's SDRAM. This task is used for the bus topology. The result is derived from the benchmarks performed and can be found in Appendix A.

- *taskGetDataIPBIFO* - Responsible for the retrieval of the output of the matched filter data from the communications processors. This task transfers data using a DMA memory transfer from the IPBIFOs to the DSP's SDRAM. This task is used for the line/ring topology. The result is also derived from the benchmarks performed and can be found in Appendix A.

- *taskAlaDecode* - Responsible for the Alamouti decoding algorithm as discussed in Section 5.3. This task was implemented on one processor and its results are recorded here.

- *taskWriteDataGBL* - Responsible for the transfer of data to the DSP board's global memory where it can then be retrieved via the VME bus by the host computer for post processing. The result is also derived from the benchmarks performed and can be found in Appendix A.

6.3.1 Bus

Alamouti bus processing is simulated as follows: A buffer full of data is collected from the matched filtering process, the DSP board's global memory is filled and an interrupt is sent to the decoding processor to let it know that the data is available for processing. A software semaphore is used for the "ping-pong" buffer that controls the data input and output from the Alamouti decoding process. When the decoding process is finished, the results are sent to the DSP board's global memory. By placing the results in global memory, it is then available to the host computer for post processing if desired. See Figure 5.1. Table 6.2 shows the number of processor timing ticks for each task using DSP-to-DSP communication via the VME bus.

Table 6.2: Alamouti Bus Tasks & Timing Results

Task Name	Number of Ticks	Number of Times Used	Total
taskGetDataVME	231,508	2	463,016
taskWriteDataGBL	47,621	2	95,242
taskAlaDecode	983,860	1	983,860
Grand Total			1,542,118

6.3.2 Line/Ring

Data is simulated and processed as explained in Section 5.3.2 (see Figure 5.2) and using the same methodology as described in Section 6.3.1. Table 6.3 shows the number of processor timing ticks for each task using inter-processor BIFIFO communications.

Table 6.3: Alamouti Line/Ring Tasks & Timing Results

Task Name	Number of Ticks	Number of Times Used	Total
taskGetDataIPBIFO	20,211	2	40,422
taskWriteDataGBL	47,621	1	47,621
taskAlaDecode	983,860	1	983,860
Grand Total			1,071,903

6.3.3 Comparison

Table 6.4 shows the total number of processor cycles simulated for each topology for the Alamouti decoding algorithm on our real-time system. A clear difference can be seen between the bus and line/ring topologies. One third of the total processor cycles are used for bus data transfers compared with about ten percent for the line/ring topology (see Tables 6.2 and 6.3). The line/ring topology is the clear

winner with a 44 percent speed up due to quick IPBIFO interconnects between DSP processors.

The Alamouti algorithm implementation has proven to be very simple to implement using only two processors. Since only one board is needed to compute this MIMO algorithm, IPBIFO inter-processor communication should always be used. Future work could be dedicated to higher order Alamouti systems; however, the cost versus benefit is questionable. A more diverse system with additional receive and/or transmit antennas may not require much additional processing power, but its benefits are limited to diversity gain and not a fundamental increase in channel capacity or throughput.

Table 6.4: Alamouti Real-Time Timing Results Comparison

Topology	Total Timing Ticks	Normalized Timing
Bus	1,542,118	1.44
Line/Ring	1,071,903	1.00

6.4 2 x 2 Differential Space-Time Real-Time Processing

The 2×2 differential space-time algorithm decoding has been implemented on two DSP processors as discussed in Section 5.4. The space-time decoding algorithm is the same for each case implemented with the exception of inter-processor communication. Each processor also has a task called "main" that is responsible for processor initialization and is not timed and a task named "taskHeartbeatLED" that is used as the idle task and for debugging purposes and is also not timed.

The critical processor-intensive tasks that are timed are the following:

- *taskGetDataVME* - Responsible for the retrieval of the output of the matched filter data from the communications processors. This task transfers data from the VME bus to the DSP board's global buffer and then to the DSP's SDRAM.

The result is derived from the benchmarks performed and can be found in Appendix A.

- *taskGetDataIPBIFO* - Responsible for the retrieval of the output of the matched filter data from the communications processors. This task transfers data using a DMA memory transfer from the IPBIFO's to the DSP's SDRAM. The result is also derived from the benchmarks performed and can be found in Appendix A.

- *taskDiffDecode* - Responsible for the 2×2 differential space-time decoding algorithm as discussed in Section 5.4. The differential decoding task (taskDiffDecode) is split into two processors as discussed in Section 5.4.1. There will be two different subtasks created for this task, one that encompasses finding all of the $Y_k Y_{k-1}$ and the first three resultant G product matrices (referred to as "taskDiffDecode$_A$") and the other that processes the last 5 resultant G product matrices and finds the maximum value of the results (referred to as "taskDiffDecode$_B$"). This task was implemented on the wireless system and each subtask's results are recorded here.

- *taskWriteDataGBL* - Responsible for the writing of data to the DSP board's global memory where it can then be retrieved through the VME bus by the host computer for post processing. The result is also derived from the benchmarks performed and can be found in Appendix A.

6.4.1 Bus

The 2×2 differential bus processing is simulated as follows: when a buffer full of data is collected from the matched filtering process, the DSP board's global memory is filled and an interrupt is sent to the decoding processors to let them know that the data is now available for processing. Data in global memory is partitioned so that all data is available to all processors for both read/write access. Software semaphores are used for the "ping-pong" buffers that control the data input and output from the 2×2 differential decoding process to ensure that data is not overwritten.

The differential decoding algorithm is split across two processors and divided into two different software task. The processor in-charge of software task taskDiffDecode$_A$ begins computing the Y_kY_{k-1} and passes the results to the processor in-charge of the other software task, taskDiffDecode$_B$. The processor in-charge of taskDiffDecode$_A$ also computes the first three results of GY_kY_{k-1} and passes them to the second processor which computes the last five GY_kY_{k-1} and finds the maximum of all eight results. When finished processing, data is sent to another memory location in the DSP board's global memory, making it available to the host computer board for post processing. See Figure 5.3. Table 6.5 shows the number of processor timing ticks for each task using VME bus DSP-to-DSP communications.

Table 6.5: 2×2 Differential Space-Time Bus Tasks & Timing Results

Task Name	Number of Ticks	Number of Times Used	Total
taskGetDataVME	231,508	5	1,157,540
taskDiffDecode$_A$	4,018,715	1	4,018,715
taskDiffDecode$_B$	4,952,453	1	4,952,453
taskWriteDataGBL	47,621	2	95,242
Grand Total			10,223,950

6.4.2 Line/Ring

The 2×2 differential line/ring processing is simulated as follows: when a buffer full of data is collected from the matched filtering process, both of the DSP's IPBIFOs are filled with the same information and interrupts are sent to the decoding processors to let them know that the data is available for processing. The data is received by both decoding processors and is partitioned in the processor's local SDRAM for storage. Data is processed using the same methodology as described in Section 6.4.1 and when finished, the resultant data is sent to a memory location in

the DSP board's global memory making it available to the host computer board for post processing. See Figure 5.4. Table 6.6 shows the number of processor timing ticks for each task using IPBIFO DSP-to-DSP processor communications.

Table 6.6: 2×2 Differential Space-Time Line/Ring Tasks & Timing Results

Task Name	Number of Ticks	Number of Times Used	Total
taskGetDataIPBIFO	20,211	6	121,266
taskWriteDataIPBIFO	20,211	4	80,844
taskDiffDecode$_A$	4,018,715	1	4,018,715
taskDiffDecode$_B$	4,952,453	1	4,952,453
taskWriteDataGBL	47,621	1	47,621
Grand Total			9,220,899

6.4.3 Star

The 2×2 start topology results are the same as the 2×2 ring case. This is due to the virtual IPBIFO connection (as discussed in Section 5.4.4) essentially making the star topology a ring topology with our equipment. See Figure 5.5. Table 6.7 shows the number of processor timing ticks for each task using IPBIFO DSP-to-DSP processor communications, including the one virtual IPBIFO connection.

6.4.4 Comparison

Table 6.8 shows the total number of processor cycles simulated for each topology for the 2×2 differential space-time decoding algorithm on our real-time system along with the normalized timing value. A difference can be seen between the bus and line/ring topologies, but not as pronounced as the Alamouti algorithm. About one tenth of the total processor cycles are used for bus data transfers compared with about three percent for the line/ring and star topology (see Tables 6.5, 6.6, and 6.7).

Table 6.7: 2×2 Differential Space-Time Star Tasks & Timing Results

Task Name	Number of Ticks	Number of Times Used	Total
taskGetDataIPBIFO	20,211	6	121,266
taskWriteDataIPBIFO	20,211	4	80,844
taskDiffDecode$_A$	4,018,715	1	4,018,715
taskDiffDecode$_B$	4,952,453	1	4,952,453
taskWriteDataGBL	47,621	1	47,621
Grand Total			9,220,899

The line/ring and star topologies are the winners with an eleven percent speed up due once again to quick IPBIFO interconnects between DSP processors. As discussed in Chapter 5, the line/ring and start topology are essentially the same on our real-time system and thus the timing results are also the same. The narrower difference between bus-based communications and IPBIFO communications can easily be attributed to the virtual connections that were used. These virtual connections essentially constitute a hybrid bus/IPBIFO-based communication and contribute to the slow down.

However, not all difference in the results can be attributed to the virtual connections. The lowered percentage difference between the total timing ticks for the fastest and slowest Alamouti topology and the fastest and slowest 2×2 differential space-time algorithm is attributed to the decreasing role that the inter-processor communication plays within a processor intensive algorithm. As the need for more processing increases (in a system with limited processing power and a limited number of processors), the role that inter-processor communication plays is reduced. In this case, the bottleneck appears to be processing power and not inter-processor communication.

Table 6.8: 2×2 Differential Real-Time Timing Results Comparison

Topology	Total Timing Ticks	Normalized Timing
Bus	10,223,950	1.11
Line/Ring	9,220,899	1.00
Star	9,220,899	1.00

6.5 4 x 4 Differential Space-Time Real-Time Processing

The 4×4 differential space-time algorithm decoding has been implemented using six DSP processors as discussed in Section 5.5. The space-time decoding algorithm is the same for each case implemented with the exception of the inter-processor communication. Each processor has a task called "main" that is responsible for processor initialization and that is not timed. Each processor also has a task named "taskHeartbeatLED" that is used as the idle task and for debugging purposes and not timed.

The critical processor intensive tasks that are timed are the following:

- *taskGetDataRACE* - Responsible for the retrieval of any board-to-board data, excluding the host computer. This task transfers data from the VME bus using the RACEway board-to-board module, then to the DSP board's global buffer, and finally to the DSP's SDRAM. The result is derived from the benchmarks performed and can be found in Appendix A.

- *taskGetDataIPBIFO* - Responsible for the transfer of data using a DMA memory transfer from the IPBIFO's to the DSP's SDRAM. This task will be used for all DSP-to-DSP memory transfers (excluding the bus case). The result is also derived from the benchmarks performed and can be found in Appendix A.

- *taskDiffDecode* - Responsible for the 4×4 differential space-time decoding algorithm as discussed in Section 5.5. This task is different depending on the processor it runs on due to the partitioning of the function onto multiple DSPs. One processor is responsible for computing the $Y_k Y_{k-1}$, four for finding four

instances of GY_kY_{k-1}, and the fifth DSP processor is responsible for finding the maximum value of each 16 length vector and for posting the results to global memory. Extra processing is reserved for the fifth DSP to allow any additional processing as discussed in Chapter 5.

- *taskWriteDataGBL* - Responsible for the writing of data to the DSP board's global memory where it can then be retrieved through the VME bus by the host computer for post processing. The result is also derived from the benchmarks performed and can be found in Appendix A.

There will be three different subtasks created for the differential decoding task (taskDiffDecode): one that encompasses finding all of the Y_kY_{k-1} diagonal elements (referred to as "taskDiffDecode$_A$"), one that computes four resultant G product matrices (referred to as "taskDiffDecode$_B$"), and one that finds the maximum value of the results (referred to as "taskDiffDecode$_C$"). This task was implemented on the wireless system and each subtask's results are benchmarked here.

6.5.1 Bus

The 4×4 differential bus processing is simulated as follows: data is transferred through global memory to simulate RACEway bus transfers. When a buffer full of data is collected from the matched filtering process, the DSP board's global memory is filled and interrupts are sent to the decoding processors to let them know that the data are available for processing. Multiple boards are used, so data are transferred to the board's global memory that hosts the processor that runs "taskDiffDecode$_A$." The output of "taskDiffDecode$_A$" is transferred to the global memory of the other two boards and "taskDiffDecode$_B$" starts running on the four available processors. Their results are transferred to the last remaining processor to compute "taskDiffDecode$_C$." When finished processing, data are sent to another memory location in the DSP board's global memory making it available to the host computer board for post processing. See Figure 5.6. Table 6.9 shows the number of processor timing ticks for each task using RACEway bus DSP-to-DSP communications.

Table 6.9: 4×4 Differential Space-Time Bus Tasks & Timing Results

Task Name	Number of Ticks	Number of Times Used	Total
taskGetDataRACE	165,517	12	1,986,204
taskWriteDataRACE	37,500	5	187,500
taskDiffDecode$_A$	3,080,106	1	3,080,106
taskDiffDecode$_B$	5,859,096	4	23,436,384
taskDiffDecode$_C$	8	1	8
taskWriteDataGBL	47,621	1	47,621
Grand Total			28,737,823

6.5.2 Line/Ring

The 4×4 differential line/ring processing is simulated using the same methodology found in the 4×4 bus case but using the connections illustrated in Figure 5.7 and discussed in Section 5.5.2. Table 6.10 shows the number of processor timing ticks for each task using IPBIFO transfers, and virtual connections comprised of RACEway bus DSP-to-DSP communication.

6.5.3 Grid/Mesh

The 4×4 differential grid/mesh processing is simulated using the same methodology found in the 4×4 bus case but using the connections illustrated in Figure 5.8 and discussed in Section 5.5.3. Table 6.11 shows the number of processor timing ticks for each task.

6.5.4 Star

The 4×4 differential star processing is simulated using the same methodology found in the 4×4 bus case but using the connections illustrated in Figure 5.9 and discussed in Section 5.5.4. Table 6.12 shows the number of processor timing ticks for each task.

Table 6.10: 4×4 Differential Space-Time Line/Ring Tasks & Timing Results

Task Name	Number of Ticks	Number of Times Used	Total
taskGetDataIPBIFO	20,211	12	242,532
taskWriteDataIPBIFO	20,211	10	202,110
taskGetDataRACE	165,517	8	1,324,136
taskWriteDataRACE	37,500	12	450,000
taskDiffDecode$_A$	3,080,106	1	3,080,106
taskDiffDecode$_B$	5,859,096	4	23,436,384
taskDiffDecode$_C$	8	1	8
taskWriteDataGBL	47,621	1	47,621
Grand Total			30,582,897

Table 6.11: 4×4 Differential Space-Time Grid Tasks & Timing Results

Task Name	Number of Ticks	Number of Times Used	Total
taskGetDataIPBIFO	20,211	10	202,110
taskWriteDataIPBIFO	20,211	6	121,266
taskGetDataRACE	165,517	10	1,655,170
taskWriteDataRACE	37,500	10	375,000
taskDiffDecode$_A$	3,080,106	1	3,080,106
taskDiffDecode$_B$	5,859,096	4	23,436,384
taskDiffDecode$_C$	8	1	8
taskWriteDataGBL	47,621	1	47,621
Grand Total			28,917,665

Table 6.12: 4×4 Differential Space-Time Star Tasks & Timing Results

Task Name	Number of Ticks	Number of Times Used	Total
taskGetDataIPBIFO	20,211	1	20,211
taskWriteDataIPBIFO	20,211	1	20,211
taskGetDataRACE	165,517	12	1,986,204
taskWriteDataRACE	37,500	12	450,000
taskDiffDecode$_A$	3,080,106	1	3,080,106
taskDiffDecode$_B$	5,859,096	4	23,436,384
taskDiffDecode$_C$	8	1	8
taskWriteDataGBL	47,621	1	47,621
Grand Total			29,040,745

6.5.5 Hypercube

The 4×4 differential hypercube processing is simulated using the same methodology found in the 4×4 bus case but using the connections illustrated in Figure 5.10 and discussed in Section 5.5.5. Table 6.13 shows the number of processor timing ticks for each task. The $*$ makes reference to the fact that there are only four processors performing these tasks instead of six. Due to the reduced number of processors a lower bit rate is expected.

6.5.6 Binary Tree

The 4×4 differential binary tree processing is simulated using the same methodology found in the 4×4 bus case but using the connections illustrated in Figure 5.11 and discussed in Section 5.5.6. Table 6.14 shows the number of processor timing ticks for each task. The $*$ again makes reference to the fact that there are fewer processors (in this case 3 instead of 6) performing these tasks. A lower bit rate is also expected.

Table 6.13: 4×4 Differential Space-Time Hypercube Tasks & Timing Results

Task Name	Number of Ticks	Number of Times Used	Total
taskGetDataIPBIFO	20,211	9	181,899
taskWriteDataIPBIFO	20,211	6	121,266
taskGetDataRACE	165,517	4	662,068
taskWriteDataRACE	37,500	4	150,000
taskDiffDecode$_A$	3,080,106	1*	3,080,106
taskDiffDecode$_B$	5,859,096	4*	23,436,384
taskDiffDecode$_C$	8	1*	8
taskWriteDataGBL	47,621	1	47,621
Grand Total			27,679,352

Table 6.14: 4×4 Differential Space-Time Binary Tree Tasks & Timing Results

Task Name	Number of Ticks	Number of Times Used	Total
taskGetDataIPBIFO	20,211	7	141,477
taskWriteDataIPBIFO	20,211	3	60,633
taskGetDataRACE	165,517	1	165,517
taskWriteDataRACE	37,500	1	37,500
taskDiffDecode$_A$	3,080,106	1*	3,080,106
taskDiffDecode$_B$	5,859,096	4*	23,436,384
taskDiffDecode$_C$	8	1*	8
taskWriteDataGBL	47,621	1	47,621
Grand Total			26,969,246

6.5.7 Pyramid

The 4×4 differential pyramid processing is simulated using the same methodology found in the 4×4 bus case but using the connections illustrated in Figure 5.12 and discussed in Section 5.5.7. Table 6.15 shows the number of processor timing ticks for each task. The $*$ again makes reference to the fact that there are fewer processors (in this case 1 instead of 6) performing these tasks. A lower bit rate is also expected.

Table 6.15: 4×4 Differential Space-Time Pyramid Tasks & Timing Results

Task Name	Number of Ticks	Number of Times Used	Total
taskGetDataIPBIFO	20,211	3	60,633
taskWriteDataIPBIFO	20,211	1	20,211
taskGetDataRACE	165,517	2	331,034
taskWriteDataRACE	37,500	2	75,000
taskDiffDecode$_A$	3,080,106	1*	3,080,106
taskDiffDecode$_B$	5,859,096	4*	23,436,384
taskDiffDecode$_C$	8	1*	8
taskWriteDataGBL	47,621	1	47,621
Grand Total			27,050,997

6.5.8 Comparison

Table 6.16 shows the total number of processor cycles for each topology for the 4×4 differential space-time decoding algorithm on our real-time system along with the normalized timing values. The normalized timing values are shown to the thousandths place to differentiate between the topologies. Only a small difference can be seen between the bus, line/ring, star, and grid topologies. Due to the high number of processor cycles required to decode the 4×4 differential algorithm, the processor-to-processor communication did not play a significant role in the total timing values.

The topology with the highest number of total timing ticks (line/ring) saw an inter-processor communication timing value versus total decoding algorithm timing value ratio of ten percent (See Table 6.10).

Table 6.16: 4×4 Differential Real-Time Timing Results Comparison

Topology	Total Timing Ticks	Normalized Timing
Bus	28,737,823	1.000
Line/Ring	30,582,897	1.060
Grid	28,917,665	1.006
Star	29,040,745	1.010
Hypercube*	27,679,352	0.963
Binary Tree*	26,969,246	0.938
Pyramid*	27,050,997	0.941

The $*$'s contained in Table 6.16 indicate topologies that are not directly comparable with the bus, line/ring, star, and grid topologies. This is due to the reduced number of processors for these space-time algorithms and the reduction in interconnection delays between the DSPs. As discussed in Section 6.1, these results help show the cost versus benefit of obtaining higher performance processors to replace multiple lesser performing processors. The bus, line/ring, star, and grid were normalized to one around the best performer, the bus topology. The remainder of the topologies were also normalized around the bus topology for comparison.

A surprising result is that the bus topology is the most efficient of the four common topologies. This is attributed to the virtual connections. The inter-processor communication overhead associated with "fitting" the other topologies within our real-time system using virtual topology connections outweighed the benefits of a simple bus-based topology. The virtual connections, as illustrated in Section 5.5 for the 4×4 topologies, also make use of RACEway bus-based communications to commu-

nicate from board to board. Even though there are fewer bus transfers using virtual connections on the line/ring, grid, and star topologies compared to the bus topology, there are significantly more IPBIFO transfers. In the end, the additional IPBIFO transfers didn't out-weigh the slower RACEway bus-based data transfers due to the number of times they had to used to move data to the appropriate location for processing.

The Binary Tree algorithm was the best performer out of the non-conforming parallel processing topologies. This is primarily due to the fact that all communication is done via the IPBIFOs except for one RACEway transfer. Since board-to-board communication is the costliest, the topology requiring the least such communication won. These results may be misleading considering that we are taking into account the decoding of only one block of data. For example, the way the binary tree topology was proposed, the inter-processor communication would have to go up and down the tree. This would complicate and slow down any pipelining of data.

As previously discussed in Section 6.1, the advanced topologies are not directly realizable on our system. However, Table 6.16 does show over a six percent increase in performance between the bus topology and the binary tree topology. A cost versus benefit analysis would have to be performed to determine if a six percent increase would justify the purchase of an additional board. The board would have to replace the processing power of four DSP processors and be able to transfer data of the RACEway bus to be comparable to this discussion.

6.6 Interpretation

These results suggest that for higher order MIMO space-time decoding algorithms, the multi-processor topology may be less important than the skill of the programmer efficiently implementing the decoding algorithm. Even with reduced inter-processor communication, the hypercube, binary tree, and pyramid do not cause a significant speed up in processing capability and show only marginal gains. It appears that the majority of development time should be spent optimizing the space-time decoding implementation. The logical first steps would be to ensure that optimized

assembly libraries and functions are used whenever possible. More time should be spent optimizing DSP code, most likely assembly code, than optimizing data transfers.

However, when the total number of processor timing ticks is compared to the percentage of total timing ticks, then it appears that a larger percentage of processor cycles are used for inter-processor communication. If the rate at which the processor timing ticks increases is constant and linear (from the 2×2 to the 4×4 differential space-time decoding algorithms), the rate at which additional processors are added is constant and linear (taken from the 2×2 to the 4×4 case again), and the rate at which inter-processor communication increases is constant and linear (again, from the 2×2 to the 4×4 case), then additional insight can be gained for higher order MIMO systems.

Table 6.17 shows the rate of increase in the number of processors needed to compute the space-time algorithm, the amount of processing, and the inter-processor communication overhead that was required to go from a 2×2 differential system to a 4×4 differential system. Figure 6.1 shows what would happen if the amount of processing and inter-processor communication continued at the same rate as what was observed in Table 6.17. Notice that around a 16×16 differential space-time decoder, the amount of processing required to compute the algorithm would be the same as the amount of processing required to transfer data back and forth between 70 processors (assuming the same rate of increase of processors)! This would clearly contradict the idea that most work should be spent optimizing the decoding algorithm and not the inter-processor communication.

Table 6.17: Differential Space-Time Coding Comparison and Rate of Increase

Space-Time Code	Number DSPs	Algorithm Timing	Inter-Processor Timing
2×2 Differential	2	8,971,168	249,731
4×4 Differential	6	26,516,498	2,221,325
Rate of Increase	$3\times$	$\approx 3\times$	$\approx 10\times$

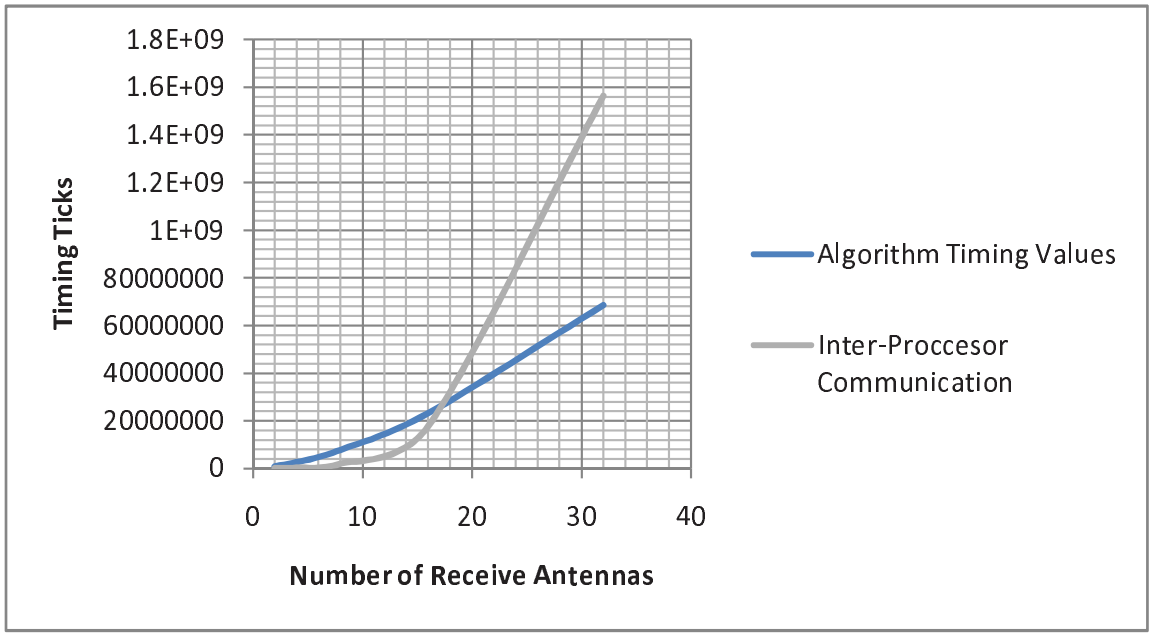

Figure 6.1: Algorithm Timing Values Versus Inter-Processor Communication

To implement an 8×8 differential space-time decoding algorithm on our system, the primary task should be to optimize the differential decoding algorithm and spend less time on the inter-processor communication. To implement a 16×16 differential space-time decoding algorithm, the work of optimizing the decoding algorithm should already be accomplished and most of the time should then be spent to improve the inter-processor communication. This appears to be the logical development path.

Chapter 7

Conclusion

There is scarce literature on the real-time implementation of MIMO systems and almost none for higher order antennas systems (greater than the 4×4 antenna case). We have examined the real-time parallel processing system that the BYU wireless lab has built and have laid a foundation for higher order antenna MIMO research. Common parallel processing topologies have been discussed in Chapter 3. The Alamouti, the 2×2 and the 4×4 differential space-time algorithms have been discussed in detail and their decoding algorithms have been implemented and benchmarked. All inter-processor communication methods have also been benchmarked. These benchmarks, in the form of the processor's own timing capability, have been used to piece together decoding algorithm plus parallel processing topology results. These results are used to discuss each topology's performance for each MIMO decoding algorithm presented.

7.1 Discussion and Recommendations

For all of the calculations performed, each space-time decoding algorithm was calculated for only one block of data through the parallel processing topology. Thus, these results only account for one pass of data through the system. These conditions allow for the best case scenario where processors are communicating without interruptions from other processors. Bus based communications, including board to board communications, are especially vulnerable to considerable slow down due to processors waiting for the bus to be free. For this reason, for the 4×4 differential decoding algorithm, the bus based topology may not be the top performer when total throughput is considered.

For the simple narrow band 2×2 Alamouti algorithm, the line/ring topology is clearly the best bet. All data is received and processed on one DSP board. When data needs to be transferred to diagonal processors (where a direct IPBIFO connection is not available) there are two paths available – transfer data through another DSP processor or transfer data through the global memory. Transferring data through another processor is faster; however, it ties up another processor's CPU cycles and a DMA coprocessor. Transferring data through global memory is slower, but does not require the help of another DSP processor. However, only one DSP processor is able to access global memory at a time and may not be available when needed. In our system, when implementing the Alamouti algorithm, there is plenty of processing power to choose either method.

For the 2×2 differential decoding algorithm, the line/ring or star topology should be used. Because there is not a dedicated central processor available on our DSP boards, the line/ring topology makes more sense than the star. The line/ring topology takes advantage of the inherent ring architecture of the DSP boards when using IPBIFOs for data transfers. When multiple passes of data are required, the ring architecture would be able to maintain a constant throughput by continually passing data to the next DSP for processing. The only slow down may be when the final processor needs to pass data off board through the global memory. However, this may be accomplished using a separate DMA coprocessor and should not interfere with IPBIFO transfers.

The 4×4 differential decoding algorithm is able to stress the system more and is thus more interesting. Although a bus-based topology is simple and effective according to the results from Chapter 6, as discussed, there are throughput concerns when multiple blocks of data are received. The grid topology also looks promising since it was only a step behind the bus topology. With much of the processor interconnection through IPBIFOs, it should also hold up better to multiple passes of data.

The recommended topology for the space-time decoding algorithms discussed and implemented on our system is the one that takes advantage of IPBIFO commu-

nication whenever possible on each processor board and uses the RACEway interconnect for board-to-board communication. However, care must be taken when trading topology complexity for faster inter-processor communication, a lesson that is learned from the 4×4 differential bus topology results.

Due to an exponential increase in required processing power, see Figure 6.1, simply doubling the processing requirements obtained from the 4×4 case is not an adequate solution to implement real-time 8×8 differential decoding (eight communications processors are needed leaving just 12 available for the decoding algorithm). The BYU wireless lab should concentrate on implementing the 4×4 case and increasing the bandwidth. There is plenty of additional processing power to accommodate a wider band signal. However, a burst mode of transmitting data where the time between data bursts is greater than the time needed to decode at the receiver is possible. Although not technically real-time, this method of transmission could accommodate the 8×8 case.

7.2 Future Work

As mentioned, most of the future work should be concentrated on optimizing the Alamouti, 2×2 and 4×4 space-time coding and decoding algorithms. Much care should be taken to use TI's assembly routines whenever applicable. Optimized assembly libraries should be included with the custom libraries already created and should contain the common algorithms and functions that are used throughout the differential space-time encoding and decoding process. With a solid basis of optimized space-time functions, along with inter-processor functions that have already been created, faster prototyping and programming would be possible to accelerate the research in real-time MIMO processing. Other modulation techniques must be implemented on the system to allow for real-time differential space-time encoding and decoding.

Future work should include researching hybrid parallel processing topologies that could fit well within our real-time system. A FPGA board could be purchased to allow greater flexibility and an increase in specialized processing power. Pentek has

many options for incorporating a FPGA into our existing system [36]. The drawback would be an increased learning curve due to the new architecture and programming language. Pentek also offers an eight processor VME board [37]. This board could open up new possibilities and allow even greater flexibility in researching real-time MIMO systems. The architecture would be familiar and the development environment would be the same.

Another possible effort would be to create a digital signal processing framework using the test system. Every system task could be standardized to be able to accept input from any other task and give output to any other task. Inter-processor communication would be the same. A graphical interface would need to be created that could allow quick parallel digital signal processing design (similar to Matlab's visual design product Simulink). Tasks and processors could be logically tied together to allow the creation of specialized functions. Matlab could possibly even be the front-end to the system. This would allow students to piece together wireless systems capable of transmitting and receiving actual data, experiment with DSP ideas and concepts, and not get bogged down in computer programming.

Thus far we have concentrated on a fairly narrow band. As the software matures and bugs are ironed out, wide band tests would be appropriate. Tests could include determining the maximum real-time bandwidth afforded by the hardware, wide band simulations in a "burst" mode that take advantage of extensive post processing, and exploring different space-time coding and decoding algorithms with different parallel topologies. First priority should be given to a completed trasmit/receive solution using Alamouti and differential space-time codes.

As 802.11n becomes the wireless local area network standard, MIMO space-time algorithms will become commonplace. Research into ultra-high bandwidth applications that use a high order of antennas, in the 10's to 100's range, would be an exciting way to further this research. Parallel processing would be essential for research as well as the proper use of topologies, both classical and hybrid.

Appendix A

Benchmarks

Benchmarks were obtained by using the TI DSP processor timers available on each DSP. For each process, algorithm, or function, a timer was used to count the number of clock cycles the DSP processor used to finish the process, algorithm, or function. Multiple cycles of the process, algorithm, or function were performed and the results were obtained by averaging the total number of cycles from each result. Calculations were performed on blocks of data that would probably be computed on the system, for example, the largest block of data that we bench-marked against was 16K words, or one BIFO full. By running the benchmarks with different sizes of data blocks, insight was gained into the overhead of each function.

For each of the benchmark tables contained in this paper, the table is formatted with the following conventions: The DSP process (for example, addition) is compared with the memory location (either IDRAM or SDRAM), the buffer size used (usually ranging from 256 words to 16K words), and processor speed (250 MHz or 300 MHz). Unless otherwise noted, a processor speed of 300 MHz is assumed. Unless otherwise noted, all benchmarks will be in megabytes per second (MB/s) to show the throughput capacity of each benchmark.

A.1 Global Memory Benchmarks

Data transfers from a single board's global memory to DSP memory were performed. Data transfers from one board's global memory to another board's global memory (via the VME bus) and then to a DSP were also performed.

Table A.1: Reading from Global Memory (MB/s)

Buffer Size	250 MHz Board		300 MHz Board	
	IDRAM	SDRAM	IDRAM	SDRAM
16K words	48.31	48.06	48.56	47.95
8K words	44.85	44.52	44.61	44.59
1K words	21.37	20.92	21.71	21.40
256 words	7.66	7.66	7.90	7.80

Table A.2: Writing to Global Memory (MB/s)

Buffer Size	250 MHz Board		300 MHz Board	
	IDRAM	SDRAM	IDRAM	SDRAM
16K words	98.08	51.38	100.79	60.30
8K words	83.26	46.78	85.78	53.98
1K words	26.97	21.77	28.46	23.77
256 words	8.23	7.65	8.61	8.14

Table A.3: Writing VME - DSP to Another Board's Global Memory (MB/s)

Buffer Size	250 MHz Board	
	IDRAM	SDRAM
16K words	17.33	15.06
8K words	16.03	14.17
1K words	7.85	7.46
256 words	2.82	2.82

Table A.4: Reading VME - Another Board's Global Memory to DSP (MB/s)

Buffer Size	250 MHz Board	
	IDRAM	SDRAM
16K words	34.56	34.52
8K words	26.11	26.43
1K words	6.01	6.37
256 words	0.92	1.78

The following was learned from these benchmarks:

- Data transfers are most effective when the largest block of data possible is transferred, to reduce the overhead of setting up the data transfer.
- In most cases, there is not a large difference between transferring data between global memory and a DSP's SDRAM and a DSP's IDRAM. However, it is still beneficial to transfer to/from IDRAM if possible.

A.2 Flash Memory Benchmarks

Data transfers from a single board's FLASH memory to DSP memory were performed.

Table A.5: Reading from Flash Memory (MB/s)

Buffer Size	250 MHz Board		300 MHz Board	
	IDRAM	SDRAM	IDRAM	SDRAM
16K words	2.51	1.96	2.98	2.32
8K words	2.51	1.96	2.98	2.32
1K words	2.51	1.96	2.98	2.32
256 words	2.50	1.96	2.98	2.32

As expected, the FLASH memory transfers were very slow. However, FLASH memory transfers are permanent and non-volatile and serves its purpose well.

A.3 Arithmetic Benchmarks

Arithmetic operations: addition, subtraction, multiplication, and division were performed in data blocks using DSP's local memory.

Table A.6: Addition (MB/s)

	250 MHz Board		300 MHz Board	
Buffer Size	IDRAM	SDRAM	IDRAM	SDRAM
16K words	27.91	7.39	33.52	8.66
8K words	27.91	7.39	33.52	8.66
1K words	27.88	7.39	33.48	8.66
256 words	27.79	7.37	33.38	8.65

Table A.7: Subtraction (MB/s)

	250 MHz Board		300 MHz Board	
Buffer Size	IDRAM	SDRAM	IDRAM	SDRAM
16K words	27.91	7.37	33.52	8.65
8K words	27.91	7.37	33.52	8.65
1K words	27.88	7.37	33.48	8.65
256 words	27.79	7.37	33.38	8.65

Table A.8: Multiplication (MB/s)

Buffer Size	250 MHz Board		300 MHz Board	
	IDRAM	SDRAM	IDRAM	SDRAM
16K words	25.65	7.09	29.99	8.51
8K words	25.64	7.09	29.99	8.51
1K words	25.62	7.08	29.96	8.51
256 words	25.55	7.08	29.88	8.50

Table A.9: Division (MB/s)

Buffer Size	250 MHz Board		300 MHz Board	
	IDRAM	SDRAM	IDRAM	SDRAM
16K words	16.65	4.60	19.65	7.40
8K words	16.65	4.60	19.65	7.40
1K words	16.64	4.60	19.64	7.40
256 words	16.60	4.60	19.60	7.40

It can be deduced that:

- Calculations performed from IDRAM are significantly faster than those same calculations performed from SDRAM. IDRAM should always be used for these arithmetic functions.
- Division is significantly slower than the other operations and should be avoided whenever possible.

A.4 Memory To Memory Transfers

Benchmarks were performed to determine processor performance when using IDRAM or SDRAM. Other benchmarks were performed to verify whether DMA memory transfers were more beneficial than processor-managed memory transfers and whether it is faster to communicate and move large chunks of data from processor to processor compared to small segments of data.

Table A.10: Moving Data from IDRAM to IDRAM (MB/s)

	250 MHz Board		300 MHz Board	
Buffer Size	DMA Co-Proc	DSP	DMA Co-Proc	DSP
16K words	193.83	30.61	232.64	36.76
8K words	190.37	30.61	228.40	36.76
1K words	153.36	30.57	183.59	36.72
256 words	92.77	30.46	110.79	36.58

Table A.11: Moving Data from SDRAM to SDRAM (MB/s)

	250 MHz Board		300 MHz Board	
Buffer Size	DMA Co-Proc	DSP	DMA Co-Proc	DSP
16K words	59.61	8.94	71.54	10.55
8K words	59.31	8.94	71.17	10.55
1K words	55.33	8.94	66.36	10.54
256 words	44.92	8.93	53.71	10.53

Table A.12: Moving Data from IDRAM to SDRAM (MB/s)

Buffer Size	250 MHz Board		300 MHz Board	
	DMA Co-Proc	DSP	DMA Co-Proc	DSP
16K words	415.04	23.96	498.01	28.77
8K words	400.58	23.96	480.75	28.77
1K words	265.62	23.94	318.35	28.75
256 words	124.99	23.95	148.43	28.77

Table A.13: Moving Data from SDRAM to IDRAM (MB/s)

Buffer Size	250 MHz Board		300 MHz Board	
	DMA Co-Proc	DSP	DMA Co-Proc	DSP
16K words	413.05	23.96	495.53	28.77
8K words	399.03	23.96	477.89	28.77
1K words	263.75	23.94	316.00	28.75
256 words	124.02	23.96	147.48	28.77

Table A.14: Library Function memcpy() - IDRAM to IDRAM (MB/s)

Buffer Size	250 MHz Board	300 MHz Board
16K words	473.59	568.65
8K words	472.43	567.36
1K words	458.02	550.06
256 words	414.63	497.96

Table A.15: Library Function memcpy() - SDRAM to SDRAM (MB/s)

Buffer Size	250 MHz Board	300 MHz Board
16K words	12.53	15.04
8K words	12.52	15.03
1K words	12.52	15.03
256 words	12.62	15.14

Table A.16: Library Function memcpy() - IDRAM to SDRAM (MB/s)

Buffer Size	250 MHz Board	300 MHz Board
16K words	50.05	53.33
8K words	49.99	53.27
1K words	49.79	53.11
256 words	49.32	52.80

Table A.17: Library Function memcpy() - SDRAM to IDRAM (MB/s)

Buffer Size	250 MHz Board	300 MHz Board
16K words	111.65	134.08
8K words	111.61	134.02
1K words	111.03	133.31
256 words	109.92	132.11

It was verified that:

- Using the DSP to manage and transfer memory is slow and eats away precious processor cycles. This should be avoided.
- Depending on where you want memory to be transferred or moved from, the DMA co-processor should be used in almost all cases. However, the memcpy() function may be useful and should be analyzed carefully to determine the ideal memory transfer technique. These benchmarks may be useful as a guide.

A.5 IPBIFO Transfers Benchmarks

Benchmarks were taken to determine the speed at which data could be transferred using the IPBIFOs under different circumstances.

Table A.18: DMA - Writing to IPXX (MB/s)

	250 MHz Board		300 MHz Board	
Buffer Size	IDRAM	SDRAM	IDRAM	SDRAM
16K words	237.50	224.95	285.15	266.73
8K words	236.84	224.04	284.22	265.62
1K words	226.56	213.80	271.48	254.77
256 words	196.92	186.52	236.30	223.63

Table A.19: DMA - Reading from IPXX (MB/s)

	250 MHz Board		300 MHz Board	
Buffer Size	IDRAM	SDRAM	IDRAM	SDRAM
16K words	237.50	227.45	285.15	269.34
8K words	236.84	226.56	284.17	268.74
1K words	226.56	216.79	271.48	256.81
256 words	196.87	188.60	235.54	223.63

Table A.20: DMA - Writing to IPYY (MB/s)

Buffer Size	250 MHz Board		300 MHz Board	
	IDRAM	SDRAM	IDRAM	SDRAM
16K words	237.50	224.95	285.15	266.73
8K words	236.91	224.04	284.22	265.62
1K words	226.56	213.80	271.48	254.77
256 words	197.26	186.52	236.32	223.63

Table A.21: DMA - Reading from IPYY (MB/s)

Buffer Size	250 MHz Board		300 MHz Board	
	IDRAM	SDRAM	IDRAM	SDRAM
16K words	237.50	227.45	285.15	269.34
8K words	236.84	226.56	284.17	268.74
1K words	226.56	216.79	271.48	256.80
256 words	196.87	188.60	235.54	223.63

Table A.22: DMA - Writing to an empty FIFO (MB/s)

Buffer Size	250 MHz Board		300 MHz Board	
	IDRAM	SDRAM	IDRAM	SDRAM
16K words	237.50	224.95	285.15	266.73
8K words	236.91	224.04	284.22	265.62
1K words	226.56	213.80	271.48	254.77
256 words	197.26	186.52	236.32	223.63

Table A.23: DMA - Reading from a full FIFO (MB/s)

Buffer Size	250 MHz Board		300 MHz Board	
	IDRAM	SDRAM	IDRAM	SDRAM
16K words	237.50	227.45	285.15	269.34
8K words	236.84	226.56	284.17	268.74
1K words	226.56	216.79	271.48	256.80
256 words	196.87	188.60	235.53	223.63

Table A.24: DSP - Writing to IPXX (MB/s)

Buffer Size	250 MHz Board		300 MHz Board	
	IDRAM	SDRAM	IDRAM	SDRAM
16K words	24.04	8.94	28.88	10.55
8K words	24.04	8.94	28.87	10.55
1K words	24.02	8.94	28.85	10.54
256 words	23.95	8.93	28.77	10.53

Table A.25: DSP - Reading from IPXX (MB/s)

Buffer Size	250 MHz Board		300 MHz Board	
	IDRAM	SDRAM	IDRAM	SDRAM
16K words	19.88	9.25	23.88	11.00
8K words	19.88	9.25	23.88	11.00
1K words	19.87	9.24	23.86	11.00
256 words	19.81	9.24	23.79	10.98

From the data, we determine that:

- Reading from or writing to the IPBIFO took the same amount of time.
- Transferring one BIFO fully maximizes throughput. However, transferring even small blocks of data at a time does not decrease throughput substantially.
- Using the DSP to transfer data to and from the IPBIFO is a very poor choice and should not be used when DMA is available.

A.6 RACEway®

The Race++ functionality of the RACEway backplane never worked. The older and slower Raceway standard had to be used.

Table A.26: RACEway® (MB/s)

Reads	Writes
29.00	128.00

The large difference between Raceway reads and writes can be explained after examining their functionality. A Raceway write starts only after the completion of the data transfer from global memory. The Raceway reads include the latency from transferring data from the other board's global memory through the Raceway interconnect.

A.7 FIR Filter Benchmarks

There exists a fundamental trade off when computing FIR filters: speed versus accuracy. The higher the number of filter taps, the slower the FIR filter but the more accurate the result. Conversely, the lower the number of filter taps, the faster the filter but the less accurate. Many benchmarks were taken to document the differences between the buffer size, the number of filter taps (numH), and the speed of the FIR

filter. Since the real and imaginary components of the received signal were needed for our tests, the TI optimized assembly function, Fir_cplx was used.

Table A.27: Fir_cplx - NumH=128 (MB/s)

	250 MHz Board	
Buffer Size	IDRAM	SDRAM
8K words	3.72	0.34
1K words	3.70	0.34
256 words	3.70	0.34
128 words	3.70	0.34

Table A.28: Fir_cplx - NumH=64 (MB/s)

	250 MHz Board	
Buffer Size	IDRAM	SDRAM
8K words	7.44	0.66
1K words	7.41	0.66
256 words	7.40	0.66
128 words	7.39	0.67

Table A.29: Fir_cplx - NumH=32 (MB/s)

Buffer Size	250 MHz Board	
	IDRAM	SDRAM
8K words	14.86	1.28
1K words	14.81	1.28
256 words	14.78	1.27
128 words	14.74	1.33

Table A.30: Fir_cplx - NumH=16 (MB/s)

Buffer Size	250 MHz Board	
	IDRAM	SDRAM
8K words	29.66	2.35
1K words	29.61	2.35
256 words	29.48	2.34
128 words	29.31	2.59

Table A.31: Fir_cplx, NumH=8 (MB/s)

Buffer Size	250 MHz Board	
	IDRAM	SDRAM
8K words	59.30	4.06
1K words	59.14	4.04
256 words	58.62	4.03
128 words	57.62	4.95

It can be seen that:

- The buffer size was not a determining factor in the speed of the FIR filter process. In fact, it appears that a smaller buffer size may be advantageous in some circumstances.
- A careful study between performance and accuracy must be made to determine the optimum FIR filter settings.

A.8 Table of Benchmarks used

Table A.32 shows an overview of all of the tasks that were timed and presented throughout this paper and extensively used throughout Chapter 6. To compute the number of timing ticks from the benchmark values (recorded in MB/s), the following formula was used:

$$\frac{4 \times 300MHz}{Benchmark \quad in \quad MB/s} \times 16K \div 4 = \frac{Total \quad number \quad of \quad timing}{ticks \quad per \quad 16K \quad buffer} \tag{A.1}$$

where processor cycles are converted from bytes to words, multiplied by the maximum buffer size and then divided by four to obtain the timing ticks (where there is 1 timing tick for each 4 processor cycles).

Multiple tests were run for each benchmark with varying results. To determined the benchmark value used throughout this thesis, an average or median was taken of the available data. For all of the benchmarks recorded in this appendix, the average was used to determine the final benchmark value. However, for the space-time decoding algorithm benchmarks the median value was chosen due to the significant difference between results. The "Comments" column of Table A.32 shows the method determining the final benchmark value.

Table A.32: Master Table for Task Timing Values

Task Name	Timing Ticks	Origin of Data	Comment
taskGetDataIPBIFO	20,211	Benchmarks	Average
taskWriteDataIPBIFO	20,211	Benchmarks	Average
taskReadDataGBL	98,845	Benchmarks	Average
taskWriteDataGBL	47,621	Benchmarks	Average
taskGetDataRACE	165,517	Benchmarks	Average
taskWriteDataRACE	37,500	Benchmarks	Average
ComplexFFT(numH=64)	537,508	Benchmarks	Average
taskAlaDecode	983,860	Code Timing	Median
taskDiffDecode$_A$(2x2)	4,018,715	Code Timing	Median
taskDiffDecode$_B$(2x2)	4,952,453	Code Timing	Median
taskDiffDecode$_A$(4x4)	3,080,106	Code Timing	Median
taskDiffDecode$_B$(4x4)	5,859,096	Code Timing	Median
taskDiffDecode$_C$(4x4)	8	TI assembly specs	TMS320C62x DSP Library - SPRC091

Bibliography

[1] A. Lozano, F. R. Farrokhi, and R. A. Valenzuela, "Lifting the Limits on High-Speed Wireless Data Access," *IEEE Communications Magazine*, vol. 39, no. 9, pp. 156–162, September 2001. 1

[2] www.3GNewsroom.com, "US 3G Auction Tops 15 Billion," http://www.3gnewsroom.com/3g_news/jan_01/news_0181.shtml, January 2001 (accessed 2002). 1

[3] G. Foschini and M. J. Gans, "On the Limits of Wireless Communications in a Fading Environment When Using Multiple Antennas," *Wireless Personal Communication*, pp. 315–335, 1998. 2

[4] C. Peel and A. Swindlehurst, "Performance of Unitary Space-Time Modulation in Rayleigh Fading," *IEEE International Conference on Communications*, vol. 9, pp. 2805–2808, 2001. 5

[5] ——, "Performance of Unitary Space-Time Modulation in a Continuously Changing Channel," *IEEE Proceedings on Acoustics, Speech, and Signal Processing*, vol. 4, pp. 2433–2436, 2001. 5

[6] Q. Spencer and A. Swindlehurst, "Some Results on Channel Capacity When Using Multiple Antennas," in *IEEE Vehicular Technology Conference*, Fall, 2000. 5

[7] J. W. Wallace and M. A. Jensen, "Experimental Characterization of the MIMO Wireless Channel," *IEEE Antennas and Propagation Society International Symposium*, vol. 3, pp. 92–95, 2001. 5

[8] ——, "Measured Characteristics of the MIMO Wireless Channel," *IEEE Vehicular Technology Conference*, vol. 4, pp. 2038–2042, Fall, 2001. 5

[9] ——, "Characteristics of Measured 4x4 and 10x10 MIMO Wireless Channel Data at 2.4-GHz," *IEEE Antennas and Propagation Society International Symposium*, vol. 3, pp. 96–99, 2001. 5

[10] ——, "Modeling the Indoor MIMO Wireless Channel," *IEEE Transactions on Antennas and Propagation*, vol. 50, no. 5, pp. 591–599, May 2002. 5

[11] Q. Spencer, B. Jeffs, M. Jensen, and A. Swindlehurst, "Modeling the Statistical Time and Angle of Arrival Characteristics of an Indoor Multipath Channel," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 3, pp. 347–360, March 2000. 5

[12] ——, "Experiments in Modeling the Space-Time Indoor Wireless Communication Channel," *IEEE Workshop on Signal Processing Advances in Wireless Communications*, 1997. 5

[13] J. W. Wallace, M. A. Jensen, A. L. Swindlehurst, and B. D. Jeffs, "Experimental Characterization of the MIMO Wireless Channel: Data Acquisition and Analysis," *IEEE Transactions on Wireless Communications*, vol. 2, no. 2, pp. 335–343, March 2003. 5, 13, 18

[14] Jon W. Wallace and Brian D. Jeffs and Michael A. Jensen, "A Real-Time Multiple Antenna Element Testbed for MIMO Algorithm Development and Assessment." Antennas and Propagation Society, June 2004. 5

[15] G. D. Golden, G. J. Foschini, R. A. Valenzuela, and P. W. Wolniansky, "Detection Algorithm and Initial Laboratory Results Using V-BLAST Space-Time Communication Architecture," *Electronics Letters*, vol. 35, no. 1, pp. 14–16, January 1999. 7

[16] G. J. Foschini, "Layered Space-Time Architecture for Wireless Communication in a Fading Environment When Using Multi-Element Antennas," *Bell Labs Technical Journal*, pp. 41–59, Autumn 1996. 7

[17] Concurrent Technologies, "VP 100 Embedded PC (Discontinued)," http://www.gocct.com, 2003 (accessed 2007). xxv, 15

[18] Pentek Inc., *Pentek Model 4292/4292 Operating Manual*, A.2 ed., One Park Way, Upper Saddle River, NJ 07458, February 20 2003. xxv, 16

[19] ——, *Pentek Raceway Handbook*, One Park Way, Upper Saddle River, NJ 07458, 2002. xxv, 17

[20] ——, *Pentek Model 6229 Operating Manual*, B.1 ed., One Park Way, Upper Saddle River, NJ 07458, February 01 2001. 18

[21] J. W. Wallace, "Modeling Electromagnetic Wave Propagation in Electrically Large Structures," Ph.D. dissertation, Brigham Young University, December 2001. 18

[22] Pentek Inc., *Pentek Model 6216 Operating Manual*, B ed., One Park Way, Upper Saddle River, NJ 07458, April 09 2001. 20, 21

[23] M. Morisio, "Commercial-Off-The-Shelf (COTS): A Survey," University of Maryland, Tech. Rep., December 2000. 25

[24] COTS-Based System (CBS) Initiative - Carnegie Mellon University, "COTS and Open Systems - An Overview," http://www.sei.cmu.edu/cbs/cbs_description.html, September 2000 (accessed 2002). 25

[25] www.cputech.com, "Commercial Off The Shelf (COTS) Technology - Application to the Defense Sector," http://www.cputech.com/tech-cots-rec.html, 2002 (accessed 2002). 26

[26] Y. Jain, "Parallel Processing With the TMS320C40 Parallel Digital Signal Processor," Sonitech International Inc., Tech. Rep. SPRA053, February 1994. 27, 28

[27] M. J. Flynn, "Very High-Speed Computing Systems," *Proceedings of IEEE*, vol. 54, pp. 1901–1909, 1966. 27

[28] S. H. Roosta, *Parallel Processing and Parallel Algorithms*. Springer, 2000. 28, 31, 33, 36, 38, 39

[29] G. Kotsis, "Interconnection Topologies and Routing for Parallel Processing Systems," Ph.D. dissertation, Institute for Applied Information and Information Systems, University of Vienna, 1992. 29

[30] A. L. DeCegama, *Parallel Processing Architectures and VLSI Hardware*. Prentice Hall, 1989. 30, 53

[31] A. Paulraj, R. Nabar, and D. Gore, *Introduction to Space-Time Wireless Communication*. Cambridge University Press, 2003. 42, 44

[32] S. Alamouti, "A Simple Transmit Diversity Technique for Wireless Communications," *IEEE Journal on Selected Areas of Communications*, vol. 16, no. 8, pp. 1451–1458, October 1998. 43

[33] B. L. Hughes, "Differential Space-Time Modulation," *IEEE Transactions on Information Theory*, vol. 46, pp. 2567–2578, November 2000. 43, 48, 49, 50

[34] D. Reynolds, X. Wang, and H. V. Poor, "Blind Adaptive Space-Time Multiuser Detection for Fading Multipath Channels," *Military Communications Conference*, vol. 2, pp. 1055–1059, 2001. 43

[35] B. M. Hochwald and W. Sweldens, "Differential Unitray Space-Time Modulation," *IEEE Transactions on Communications*, vol. 48, pp. 2041–2052, December 2000. 50, 51

[36] Pentek Inc., "Model 6250 General Information," http://www.pentek.com/products/detail.cfm?Model=6250, 2007 (accessed March 3, 2007). 102

[37] ——, "Model 4293 General Information," http://www.pentek.com/products/detail.cfm?Model=4293, 2007 (accessed March 3, 2007). 102