Multipass Communication Systems for Tiled Processor Architectures

by

Nathan Robert Shnidman

Bachelor of Science in Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 1995

Master of Engineering in Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 1996

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2006

Author .... [illegible] ....
Department of Electrical Engineering and Computer Science
February 3, 2006

Certified by .... [illegible] ....
Anant Agarwal
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .... [illegible] ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Multipass Communication Systems for Tiled Processor Architectures

by

Nathan Robert Shnidman

Submitted to the Department of Electrical Engineering and Computer Science
on February 3, 2006, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Multipass communication systems utilize multiple sets of parallel baseband receiver functions to balance communication data rates and available computation capabilities. This is achieved by spatially pipelining baseband functions across parallel resources to perform multiple processing passes on the same set of received values, thus allowing the system to simultaneously convey multiple sequences of data using a single wireless link.

The use of multiple passes mitigates the effects of data rate on receiver processing bottlenecks, making the use of general-purpose processing elements for high data rate communication functions viable. The flexibility of general-purpose processing, in turn, allows the receiver composition to trade-off resource usage and required processing rate. For instance, a communication system could be distributed across 2 passes using $2\times$ the overall area, but reducing the data rate for each pass and the resultant overall required processing rate, and hence clock speed, by 1/2. Lowering the clock speed can also be leveraged to reduce power through voltage scaling and/or the use of higher V_t devices.

The characteristics of general-purpose parallel processors for communications processing are explored, as well as the applicability of specific parallel designs to communications processing. In particular, an in depth look is taken of the Raw processor's tiled architecture as a general-purpose parallel processor particularly well suited to portable communications processing.

An example of a multipass system, based on the 802.11a baseband, implemented on the Raw processor along with the accompanying hardware implementation is presented as both a proof-of-concept, as well as a means to explore some of the advantages and trade-offs of such a system. A bit-error rate study is presented which shows this multipass system to be within a small fraction of 1dB of the performance of an equivalent data rate single pass system, thus demonstrating the viability of the multipass algorithm.

In addition, the capability of tiled processors to maximize processing capabilities at the system block level, as well as the system architectural level, is shown. Parallel implementations of two processing intensive functions: the FFT and the Viterbi decoder are shown. A parallelized assembly language FFT utilizing 16 tiles is shown to have a $1,000\times$ improvement , and a parallelized 48-tile assembly language Viterbi decoder is shown to have a $10,000\times$ improvement over corresponding serial C implementations.

Thesis Supervisor: Anant Agarwal
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

This thesis is the culmination of much hard work and perseverance on my part, but it would not have been possible without the help, support, and good wishes of a large number of people. I would especially like to thank my fiancée Alison for her help and understanding. Her presence has comforted me and been my foundation in the last hectic push to finish. My parents and family have been behind me for years and made it possible for me to pursue my educational goals. My advisor Anant Agarwal has provided me with guidance and the freedom to explore my ideas, a precious gift that I truly appreciate. Charlie Sodini and Anantha Chandrakasan first and foremost for being friends and mentors and helping me find my way, and secondly for being my thesis committee.

I would also like to thank many of the students whom I have met along the way and who have made this graduate experience so much more than just this thesis. The whole Raw group has contributed to my research and understanding and for their friendship and for that I thank them. In particular I would like thank Jason Miller for being a good friend, officemate, and sounding board. Matthew Frank for also sharing an office and for some truly thought provoking and exciting conversations, as well as his help and support. David Wentzlaff for his ideas, insight, and companionship. Hank Hoffman and Theodoros Konstantakopoulos for their friendship and their editing help. Benjamin Walker for his work on the Wireless RF board, as well as for allowing me to share a bit of what I have learned with him.

Dan McMahill, Don Hitko, and Andy Wang have provided camaraderie, teaching, learning, and shared experience and for that I thank them. Charlie Sodini's Group, Harry Lee's Group, and Anantha Chandrakasan's Group have all provided me with friendship, help, and support at various times throughout my career which has been much appreciated.

Thanks to Jennifer Tucker, Cornelia Colyer, Mary McDavitt, and all of the staff who have given so generously of their time and have kept the completion of this thesis and my continued sanity from being mutually exclusive concepts.

I would also like to thank DARPA for the financial support which made this thesis possible, and the Engim Corporation for their generous donation of RF components, schematics, and time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Increased functional demands on portable digital devices such as PDAs, cell phones, handheld gaming consoles, and laptops have led to a corresponding demand for more powerful processing capabilities in such devices [1]. Portable devices have become integrated into everyday life to such an extent that it is often impractical to carry an individual device to perform each desired user task. Thus, the trend and expectation is toward merging the functionality originally distributed among multiple devices into a single device [2]. Wireless connectivity is a capability which is increasingly important to the portability of such devices. The integration of wireless networking into portable devices enables them to perform a much fuller and more fully-functioned set of tasks. Future device designs will continue further along this trend [3].

This growing array of tasks assigned to portable devices is rapidly precluding the use of highly specialized digital processing elements. To satisfy the diversity of applications envisioned, more general-purpose processing capabilities are required. In addition, the functionality of such devices requires (or is supplemented by) incorporating faster and more efficient wireless capabilities. General-purpose parallel digital processors, which are becoming more common [4, 5, 6, 7, 8, 9], can be utilized to meet these coincident demands [10].

1.1 Portable Processing

Ubiquitous wireless access has quickly become an integral part of everyday life [2, 11]. As wireless devices have become more commonplace, the competition to incorporate more functionality and utility into small portable devices has grown [12]. Cell phones are expected to play videos, search the Internet, and act as portable email clients [13]. Game devices are used to watch movies and play Internet-linked games, in addition to their traditional functions [14, 15]. Laptops, tablet PCs, and PDAs come equipped with wireless capabilities such as Wi-Fi and Bluetooth connectivity [16, 17, 18].

The varied and disparate nature of the applications bundled on a single wireless device create a technology and design challenge. Powerful general-purpose processing will be required to provide the myriad set of functions envisioned for portable devices. Given the need for such processors, these processing resources can also be utilized for communications processing.

The advantages of reusing a system's general-purpose processor (GPP) for performing digital communications processing, as opposed to using a separate application-specific implementation, are threefold. First, there is the obvious reduction in cost. Since portable devices will already require powerful processing units, these units can be reused for communications. This will help amortize the costs already associated with providing such processors and preclude further expenditure to provide communications baseband processing.

A second major advantage is the ease of development and use associated with a general-purpose processing solution. By using a GPP, communications functions can be implemented in software. Implementation using a high level language, compiler infrastructure, and debugger can allow quicker and easier implementation of communication algorithms. This translates to a reduction in development costs and time.

Finally, the flexibility afforded by a general-purpose processing solution is extremely valuable. A software implementation allows the communication algorithms used, and even the baseband structure itself, to be modified or replaced as needed. Applications can adapt or change the algorithms applied in response to current conditions. For instance, the type of equalization used can be changed depending on the amount and types of noise observed on the communication channel. Under poor conditions, a more ambitious but complex equalization could be used to try and maximize the received signal. When conditions are more favorable, a simpler but less effective equalization algorithm might be sufficient, with the reduced processing overhead freeing up resources for other tasks or enabling a corresponding reduction in the frequency of operation.

Using a software implementation also allows system changes at a later date. Changing standards, additional functions, improved algorithms, or even bug fixes can be easily implemented in the communications chain simply through a software update. In a hardware system, a whole new revision of the hardware would be required.

1.2 Digital Computation for Communication

The majority of modern portable devices operate on digital information. As such, most wireless systems used by portable devices are digital communication systems. They are systems which have digital bits as both the input to and output from the communication link, but which convert this digital information to an analog waveform for transmission across a physical wireless link [19, 20]

The general properties and makeup of a digital communication system will be discussed in general in Section 1.2.1 and in greater detail in Chapter 2. Section 1.2.2 then addresses the bottleneck problem – the issue of the processing requirements of certain communications functions increasing dramatically as data rates increase. Such bottlenecks are of particular concern in the uniprocessor GPP or DSP case where processing requirements can rapidly outstrip the capabilities of the available resources.

1.2.1 Digital Communications Overview

A communication system is a system which conveys information from a transmitter to a receiver across a medium. The information is input at the transmitter, is conveyed across the channel, and is then output by the receiver. A digital communication system is a communication system in which the system inputs and outputs are digital bits, see Figure 1-1.

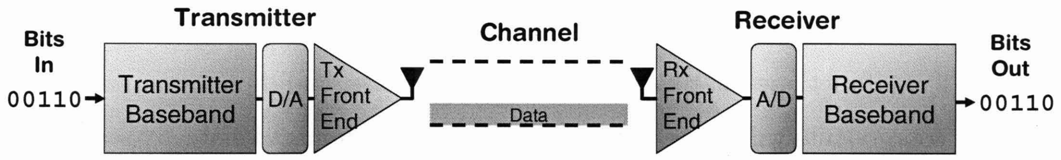


Figure 1-1: A canonical wireless digital communication system block diagram.

The receiver is often the focus of communication system design as its mandate is the most difficult. The data presented to the transmitter is deterministic. The transmitter knows exactly what information is to be sent. The job of the transmitter is to generate a minimal representation of the information to be sent which is, at the same time, robust under the rigors of communication and then to condition and transduce this representation onto the channel.

The process of transmitting the information across the channel inherently distorts the representation of that information and makes reception a probabilistic, instead of a deterministic, enterprise. The transmitter itself introduces non-linearities and component noise to transmitted signals. In addition, the channel adds uncertainty to the system through path loss and propagation effects and obfuscates the information sent by the transmitter. The receiver itself further distorts the transmitter signals through the nonlinearities and noise of its own components.

In this environment, it is the job of the receiver to overcome the distortion and uncertainty of the received signals and attempt to retrieve the information originally sent. The receiver must attempt to recover the signal which was intended to be communicated from the noise which obscures it. Due to the inherent challenges of the job of reception, receivers are necessarily more complex and difficult to design than transmitters in a duplex[1] communication system. The ability to successfully perform reception therefore implies sufficient capabilities to perform transmission. Consequently, this dissertation will focus more heavily on the receiver functions than transmitter functions.

1.2.2 Bottleneck Problem

In an ideal communication system data would flow across the wireless link at the fastest rate which the channel and applications could sustain. The ability of the receiver to process this data would never be an issue. Ideally, the applications producing and consuming the data and the channel are the drivers in setting the communication data rate. There are some receiver functions, however, whose processing requirements grow very large with increasing data rates. These functions can require more temporal reuse, and thus faster operation, than the receiver components will allow. The system then fails to meet its steady-state real-time deadlines. This is especially a problem for single GPP or DSP receiver architectures where a single set of hardware must be reused in time to try and perform all necessary functions. Even if such processors have the capability to implement a receiver and meet the real-time system requirements, there is little ability to increase temporal reuse to accommodate any increased processing required by higher data rates. This issue commonly limits the usage

[1]A *duplex* system is one in which all devices can both transmit and receive information.

of GPPs and DSPs in communications systems.

Single Receiver System

The problem of receiver processing limiting functionality can severely impact the maximum achievable data rate of the system. Depending on the channel conditions, the wireless application may be able to utilize a higher communication data rate. The receiver components, however, may not be able to sustain that higher rate, see Figure 1-2.

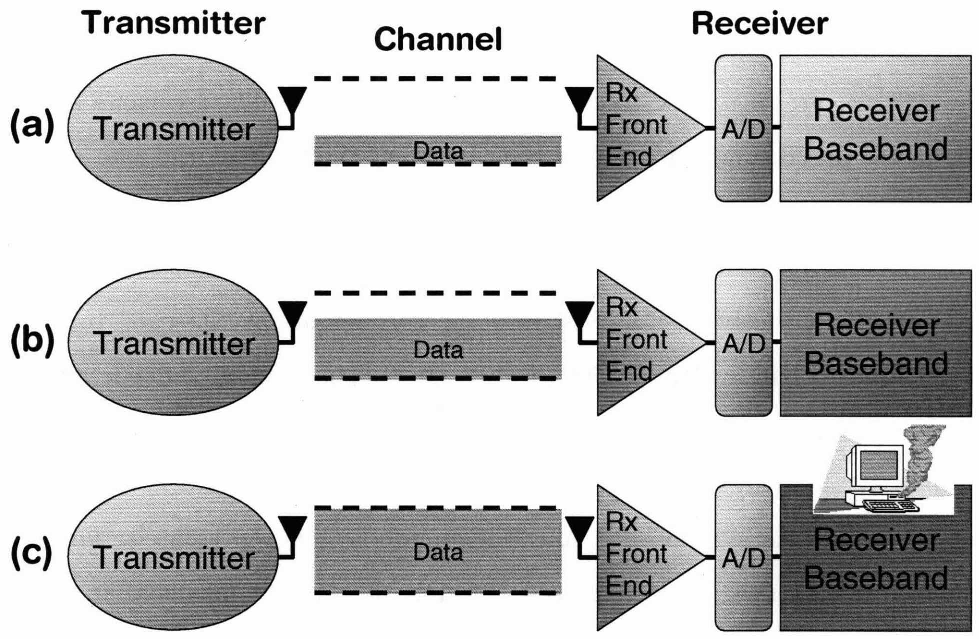


Figure 1-2: Effect of growing data rate on a single receiver system.
(a) For low data rates, the receiver can sufficiently process the received data.
(b) As the data rate increases, the increased reuse of the same receiver begins to push the temporal limits of the hardware.
(c) If the desired data rate is too high, the receiver hardware can no longer keep up.

For case (a), above, at a low data rate, the receiver can successfully process the flow of data. The receiver has been designed such that, with accommodating channel characteristics, the requested data rate is one which is available and sustainable. As the amount of data increases in case (b), however, the receiver starts to have problems keeping up. The high data rate requires more temporal reuse of the receiver resources, as well as a receiver design compatible with the higher data rate. When the desired data rate goes too high, as in case (c), the hardware resources can no longer keep up with the flow of information. Temporal reuse of the same receiver is no longer feasible and the communication link can no longer be sustained.

This problem is a reflection of the bottlenecks encountered at many of the receiver elements. Two types of bottlenecks are the main limiters of the processing data rate in receive chains. These bottleneck types are referred to here as *data expansion* and *data chunking.*

Data Expansion

Data expansion occurs when a change in data representation increases the rate of processing within a receiver functional block. This is similar to a multiplication factor being applied to the processing rate seen prior to the data expansion block. The receiver's ability to deal with this faster processing requirement then becomes the limit to the rate at which data can be sent across the wireless channel.

For instance, a receiver demapping stage converts received symbols, the complex values which represent the received sinusoids, into the corresponding bits used to generate the symbol. Since the bandwidth of the wireless channel is generally fixed, an increase in data rate is achieved by having the symbols represent a larger number of bits. When the received data symbols are converted back into bits, the receiver data flow rate grows from the symbol rate into the symbol rate times the number of bits represented. This potentially large increase in the rate of data passed through the system means that the allowable number of bits per symbol must be limited by the receiver's ability to process this higher rate of bits.

Another example of data expansion can be seen in a Viterbi decoder [21, 22], where each set of bits generates a large amount of processing as they are used to create a new set of nodes in a trellis structure. As the data rate increases, the number of bits to be processed grows and the amount of processing work becomes very difficult to schedule. Viterbi decoders, and especially an optimized tiled Viterbi decoder implementation, will be discussed in Section 7.4.5.

Data Chunking

Data chunking occurs when data must be buffered into blocks before processing can occur. The negative impact of data chunking is that streaming-type processing is stalled by the need to use relatively large memory elements in order to gather sufficient amounts of data for processing to proceed. In essence, data pipelining is interrupted by a function which relies on correlation of a large amount of data or temporally random access. This interruption breaks the streaming model (see Section 4.3).

The deinterleaving function is an example of a data chunking function in a receiver chain. Deinterleaving is a process by which the ordering of bits within a portion of the bit stream is pseudo-randomly shuffled. It is the compliment to the interleaving function which is performed at the transmitter. The shuffling and unshuffling of bits by the interleaver and deinterleaver, respectively, provide a way to mitigate transmission errors. By ordering the bits in an essentially random fashion, the system improves the chances of recovering from a localized error. This is because many channel coding processes work on consecutive bits and can recover from single bit isolated errors much more easily than multi-bit errors. By moving these consecutive bits away from each other, the hope is that the impact of an error which corrupts multiple consecutive bits will be lessened. When the deinterleaving function restores the original bit ordering, it separates these consecutive error bits and turns them into isolated errors. In order to reshuffle a section of bits, however, the intermediate partial ordering of bits must be stored until the reordering can be completed. Once a whole deinterleaving block of bits has been stored, the newly order bits can be streamed out and the deinterleaver can begin on the next set of bits. Thus, the flow of processing bits is interrupted by the need to accumulate a whole section of bits.

A data chunking function like deinterleaving can be mitigated by a sufficient amount of storage. This is a possibility because the deinterleave function does not perform any operations on the bits themselves; it is only focused on the ordering of those bits. A function like the Fast Fourier Transform (FFT) [23], however, is a more complicated example of a data chunking function as it has a profound effect on both the relative ordering of information and the data itself. The FFT processing requires interaction between data values such that multiple data values must be accumulated before operation on the data can proceed. As such, certain wireless systems, for example, the 802.11a specification [24] (see Section 7.2 and appendix A), provide a maximum acceptable latency for the FFT. This maximum latency represents the amount of time the receiver pipeline can be stalled before disrupting the flow of received information. All necessary data must be buffered and all processing must be done within this maximum latency limit. A more careful examination of the FFT and its impact on receiver processing can be found in Section 7.4.5.

Multiple Receiver System

One way to overcome bottlenecks at the receiver and enable higher data rate communication is to use multiple receivers. A system with multiple receive chains that could distinguish between and appropriately direct data destined for each of these receivers would be able to accommodate multiple data streams with higher aggregate data rate than a single receiver system. For a given data rate, each of the multiple receivers could operate at a fraction of the total data rate, thus mitigating the bottlenecked portions of the receive chain, see Figure 1-3.

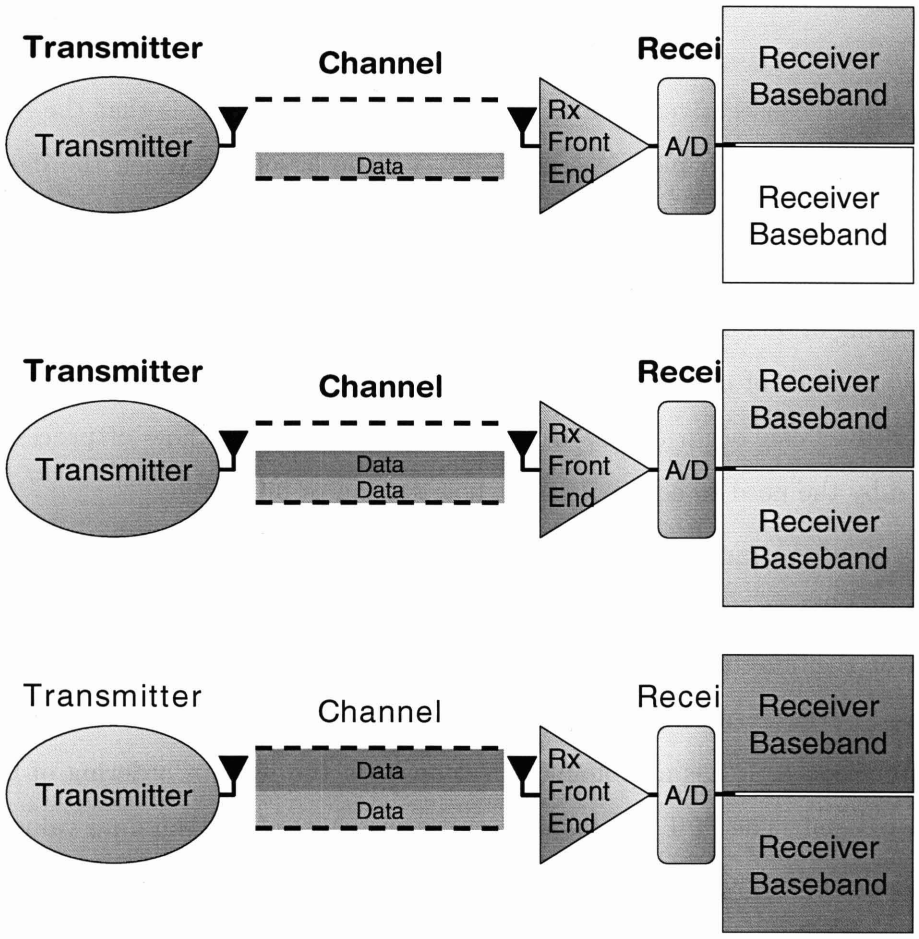


Figure 1-3: Use multiple receivers to try and accommodate higher data rates.
(a) Only a single receiver might be required for a low data rate. Any other receivers would then sit idle with under-utilized resources.
(b) If the data rate increases, however, a system which could spread the load over multiple fixed receivers would allow the system to operate on a higher aggregate data rate.
(c) A yet further increase in data rate, however, could be too much for even multiple fixed receivers.

The key to such a system would be in the combining and separating of data streams. Multiple different data streams or the demultiplexed data from a single data stream could be treated as multiple subchannels of data sent in parallel. Subchannels could be distinguished by frequency, by time, or by any means which would allow the subchannels to be combined and then later distinguished. What would be needed is a combining function which would allow multiple streams of data to be joined at the transmitter, but in an orthogonal way such that they could be disambiguated at the receiver. One specific approach using superposition as the combining function is presented in Chapter 5. Some additional potential combining functions are discussed in Section 11.1.

A multiple fixed receiver system reduces the impact of bottlenecks over a single fixed receiver system by providing both spatial and temporal reuse of the receive chain, as opposed to the temporal-only reuse of the single fixed receiver system. Spatial reuse is the reuse of a design to create multiple hardware instances which can operate in parallel. Functions can then be migrated onto parallel resources to reduce the load on a single resource. Higher data rates may be accommodated by a combination of streaming more data through each receive chain (temporal reuse) and by using multiple receive chains in parallel (spatial reuse).

This type of system does have some drawbacks, however. First of all, the system itself is fixed. While multiple receivers do allow higher data rates, there still exists a data rate threshold above which the system cannot operate. This threshold is higher than in the single receiver case as multiple receive chains must now be saturated to reach the threshold, but it still exists.

In addition, there are much higher costs associated with a multiple receiver system. In terms of characteristics important for portable wireless devices, such as physical size, dollar cost, and power, the multiple fixed receiver system is more expensive than a single receiver system. These costs might preclude building enough receivers to allow the desired maximum data rate. In effect, having multiple fixed receivers incurs the cost of the "worst" case, even in the more common or "lesser" cases. The system must be design around the maximum data rate envisioned. This means the resources required for the maximum data rate must be present even if they are not needed or in use.

From a design point of view, however, there is a significant savings in the use of multiple receivers. The spatial reuse of parallel resources means that a receive chain must only be designed once and can then be reused with little or no modification for each of the multiple receivers. The reuse of a single less ambitious or previously generated design can provide similar functionality to a more difficult, new, higher data rate design. The same problem still exists as in the single receiver case, though, of having to choose *a priori* which data rates the receivers will support. Fixing the number and design of receivers results in the forced choice of a single system architecture design point with a specific maximum data rate. This limits the multiple fixed receiver system both in terms of maximum data rate as well as possible combinations of lower data rates distributed among the multiple receivers.

The multiple receiver system as described above is an application of the multipass algorithm introduced in Section 1.2.2 and explored more fully in Chapter 5. While most of the discussion here focuses on the general-purpose processing space and the flexible receiver system presented below, the multipass algorithm is applicable to the fixed receiver case regardless of implementation method (e.g. GPP, ASIC, FPGA, etc.). The multipass algorithm can be used to achieve high data rate processing which trades increased area for a lower overall clock rate. A lower clock rate eases the temporal constraints on the system which translates to an increased ease of implementation with a corresponding reduction in design time. In addition, a lower minimum clock rate can be parlayed into lower overall power consumption through a reduction in V_{dd} and/or the use of higher V_t devices (see Section 5.4.1.

Flexible Receiver System

A wireless system which focuses on taking advantage of increases in processing speed provided by parallel resources and flexibility of general-purpose processing could do no more than slavishly re-implement elements from a traditional analog wireless systems inrece the digital domain. A better approach would be a system which could "construct" receivers as they are needed. Such a system could retask elements from a group of available resources in order to compose these resources into as many receivers as the desired data rate requires, see Figure 1-4. If later the data rate were to lower, or if other processing tasks were to take priority, these resources could then be freed up and reassigned for use in other tasks.

Such a flexible system offers the opportunity to trade-off temporal and spatial reuse to find a "sweet spot" based on the current conditions. For lower data rates, where temporal reuse is sufficient, a single receive chain can be used. When processing speeds begin to exceed the available or desired temporal reuse, however, spatial reuse can be employed to create

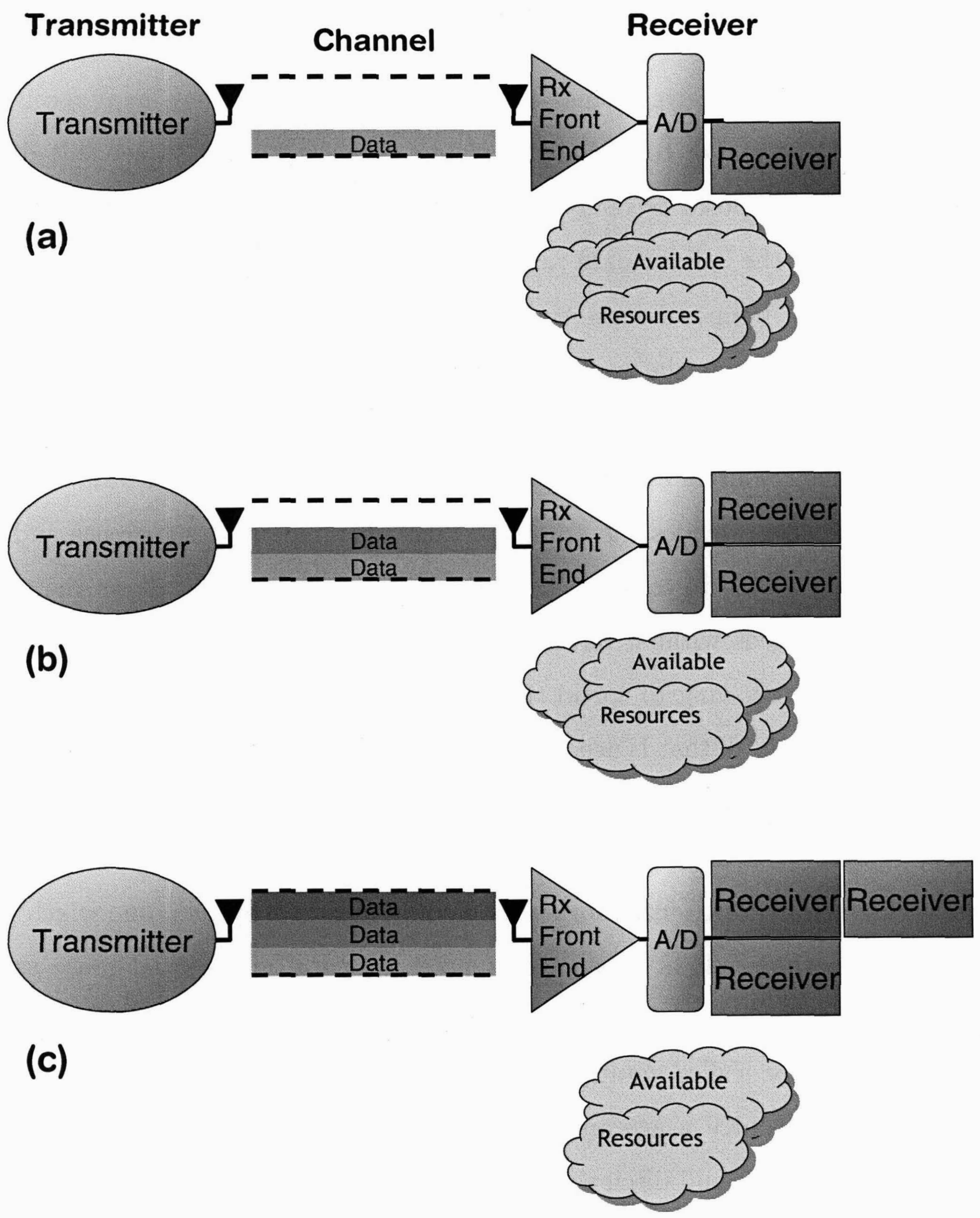


Figure 1-4: A means of flexibly composing receivers out of a set of available resources would allow the system to adjust to current conditions as needed.
(a) For lower data rates using only a small amount of resources to construct a single receiver might be sufficient.
(b) As the amount of data to be communicated grows more resources could be assigned to create additional receivers in order to successfully process the higher rate of data.
(c) A yet further increase in data rate could still be accommodated simply by making use of yet more of the available resources..

multiple parallel receive chains. This reduces the demands of temporal reuse and allows more, but slower and less ambitious, receivers to meet the processing needs. Even higher data rates may be accommodated by simply assigning more resources for communication in order to build additional receive chains. There is no need to know how many receive chains are needed ahead of time, or to design the system to accept every conceivable data rate. The system simply composes as many receivers as necessary in whatever configuration is necessitated by meet the current data rate.

This balancing of temporal reuse constraints and spatial reuse constraints allows the system to explicitly manage the trade-off between communication and computation. The system can offer more resources to communication to achieve higher data rates, but at the cost of reducing the resources available for computation of that or other data.

Such a system also provides the ability to take advantage of design reuse and reuse a single receive chain design to compose the additional receivers. This allows the system to adjust to data rate needs without having to *a priori* design in the ability to handle every data rate and without having to spend time on multiple receiver designs. A single design and verification effort is all that is needed. The system can simply reuse an already existing design to gain the use of additional cumulative data rates through the composition of receive chains.

A computer architecture suited for such a communications system, one which provides flexible hardware which allows a balance of both temporal and spatial reuse as well as concurrent independent computation, will be presented in Chapter 4. Furthermore, a communication technology, called multipass, which allows a flexible system to dynamically trade-off increased resource usage and area for lower processing rates and required clock speeds is introduced next and discussed in detail in Chapter 5.

Multipass

The *Multipass* algorithm is a method of communicating multiple streams of data in parallel across a single communication link. Multiple streams of data are processed individually at the transmitter and then combined for transmission across the channel. At the receiver, the combined data is decomposed into separate data streams, each of which is processed by its own baseband receiver processing pass.

The multipass algorithm allows communication systems to utilize multiple receiver passes in parallel to recover the data sent across a communication link. Multiple sets of resources distribute the receiver processing load for the "Hard" bottleneck functions, see Figure 1-5. For the system discussed in Chapter 7 the Viterbi decoder is this "Hard" bottleneck function. The multipass algorithm allows the system to achieve a high cumulative data rate through the use multiple baseband processing passes, and hence multiple Viterbi decoder instantiations, each running at a slower data rate. This enables the system to trade increased resource usage and area for a lower required processing rate and system clock speed. A slower clock speed can then be translated into a reduction in system power through the use of a lower supply voltage, or the use of lower V_t devices, as explored in Section 5.4.1.

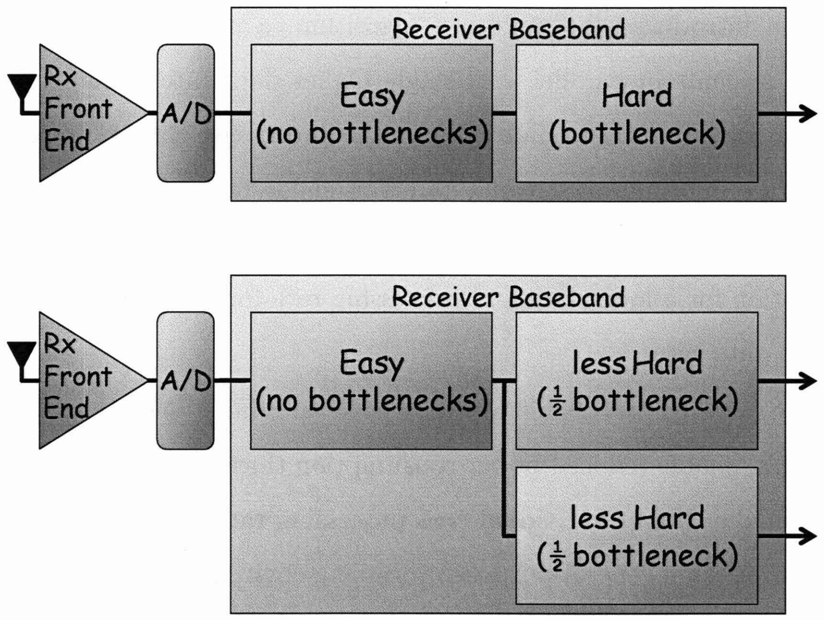


Figure 1-5: Utilizing a single receiver processing pass forces a single set of hardware to attempt to overcome bottlenecks. While the "Easy" processing functions do not require a high processing rate, a multipass system can distribute the load of the "Hard" bottlenecked processing functions across multiple sets of hardware, requiring only half the processing rate for each. This reduces the impact of bottlenecks on the system and provides the opportunity for either higher data rate processing or lower system clock speeds. In the system discussed in Chapter 7, the "Hard" bottlenecked function is the Viterbi decoder. Multiple instances of the Viterbi decoder are utilized to mitigate the requirements of this function.

In a fixed receiver system, this trade-off between area and processing rate must be decided *a priori*, but the multipass algorithm allows increased flexibility in the initial system design. In a flexible receiver system built with general-purpose processing elements, however, the multipass algorithm allows the system to dynamically adjust the number and type of passes used to meet communication and computation constraints. The ability to decompose a high data rate communication stream into multiple lower rate streams makes feasible the use of general-purpose processors for high data rate communication processing. A lower required processing rate enables system viability by reducing the data rate requirements to a level compatible with the computational overhead incurred by general-purpose processing systems. A flexible multiple receiver system which takes advantage of the multipass algorithm is discussed in Chapter 5.

1.3 Thesis Contributions

This dissertation introduces the multipass algorithm, a means of communicating multiple information streams in parallel as a single higher data rate stream over the channel. The multipass algorithm utilizes multiple receiver processing chains, or passes, to extract the lower data rate information streams from this single higher data rate received stream. Adding additional passes allows the system to trade-off increased area and processing resource consumption for a lower required processing rate for each pass, while maintaining a high cumulative data rate.

Reducing the required processing rate allows a lower minimum clock speed. A lower clock speed can, in turn, enable reduced power consumption through the use of voltage scaling or lower leakage devices. Lower individual pass processing rates also make possible high data rate communications in a general-purpose processing environment and allow a direct trade-off between communications and computation in that environment. While the multipass algorithm itself is implementation agnostic, it is as appropriate for specialized hardware or FPGA implementations as a software implementation, multipass will be discussed here in the context of general-purpose processing tiled architectures.

Multipass communications systems offer a means to utilize tiled architectures in the communications realm. Such systems take advantage of the multiple parallel general-purpose processing elements of a tiled architecture to dynamically compose receive passes. The number and rate of these passes can then be flexibly adjusted to balance the overall required communication rate with the available computation resources, while maintaining a clock rate compatible with a general-purpose processing tiled architecture.

As a first step in the proof-of-concept of a multipass system, an effort was taken to implement an 802.11a software receiver on a tiled architecture. The Raw processor was used as a tiled architecture platform to implement this baseband design. A hardware radio front-end board, the Raw Wireless Board, was designed and built to interface with the Raw handheld motherboard in order to complement and complete the receiver system, see Figure 1-6.

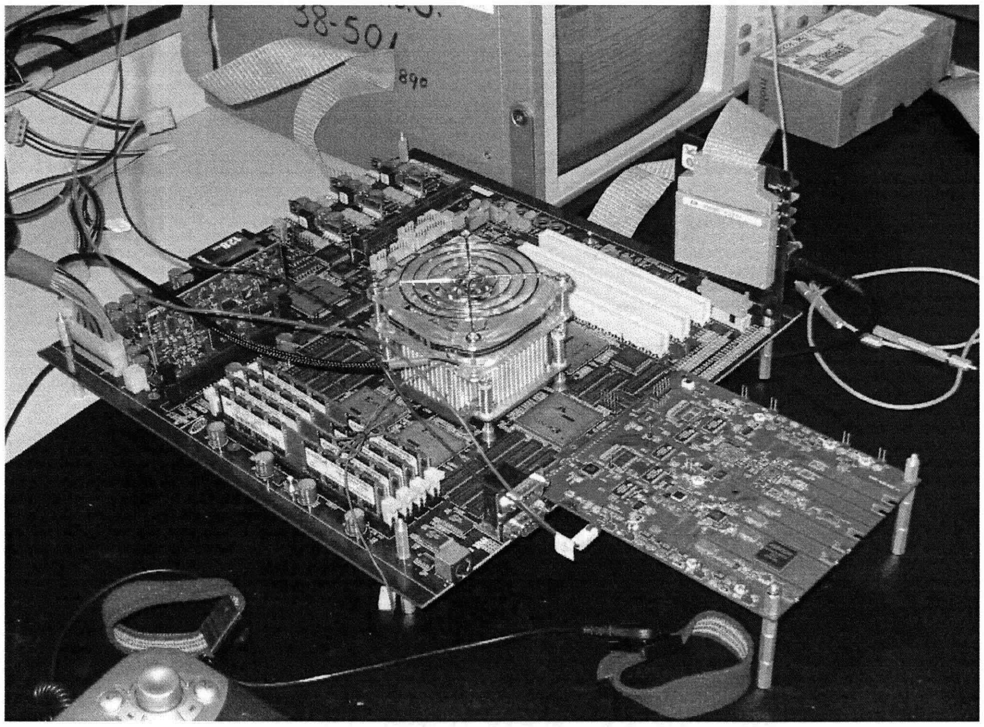


Figure 1-6: Photo of the Raw Wireless System.

The 802.11a Raw baseband design was then improved and expanded to become the first multipass receiver implementation. A multipass system, including a software multipass transmitter and a cycle-accurate Raw simulation of a multipass receiver was designed and tested. Versions of the multipass system using C and Matlab were also implemented and verified with the cycle-accurate results in order to improve simulation speeds.

As part of the optimization effort to make the 802.11a baseband compatible with real-time operation in a general-purpose processing environment, critical path receiver functions were parallelized and optimized. Chief among these bottlenecked functions were the FFT and Viterbi decoder functions. Assembly versions of both functions consistent with 54Mbps operation on a Raw digital processor were designed. This resulted in a 70 cycle 48-tile parallelized version of the Viterbi decoder, a $10,000\times$ improvement, as well as a 16-tile parallelized FFT which resulted in a $1,000\times$ improvement in performance.

Given that the multipass approach enables high data rate communication system implementation on a general-purpose processing substrate, an evaluation of the performance of a multipass communication system relative to an application specific single pass system was undertaken. The feasibility of the multipass communication approach is shown, with a multipass system using a superposition combining function achieving performance reasonably close to the single pass case. System improvements which enhance this performance to within a small fraction of 1dB of that of the single pass case are discussed. In addition, future potential compositions of multipass systems using better-performing or application-specific combining functions are discussed.

1.4 Dissertation Structure

The rest of this dissertation is structured as follows. Chapter 3 provides motivation for digital radios. It discusses the unique aspects of the digital domain which provide opportunities for new radio processing techniques. Chapter 2 provides a very brief overview of digital communication systems. Readers familiar which such systems can feel free to skip this chapter. Chapter 4 explores the characteristics of multiple types of parallel architectures. The impact of these various designs are discussed in relation to implementation of a digital radio communication system.

Chapter 5 presents the definition and a description of a multipass system, a means of utilizing the aspects of parallel architectures to implement a flexible digital radio. Chapter 6 provides details on the Raw processor, the tiled architecture used as the implementation target for the multipass system design.

Chapter 7 explores the implemented systems. A quick overview of the 802.11a specification, as well as the elements of a general 802.11a system are discussed. The details of the various system functions are explored. The hardware elements of the system and their interaction is also discussed. The software effort, including the simulation infrastructure is covered. Finally, the expansion of an 802.11a system into a multipass design, as well as the implementation details of a multipass system are presented. The optimization and parallelization of critical functions is overviewed.

Chapter 8 offers the multipass system performance results. The testing methodology is presented. Comparison with the single pass case, as well as an analysis of the results and methods of improvement, are given.

Chapter 9 is a summary of the conclusions derived from this work.

The similarity of the multipass approach to other system solutions is considered in Chapter 10, including the commonality between multipass system and multiple user access schemes.

Finally, Chapter 11 envisions future multipass system configurations and applications. The use of additional and more complex combining functions is discussed. Also contemplated is the utilization of the general-purpose flexibility enabled by multipass systems to implement adaptable algorithms and other flexible receiver architectures. The application of multipass designs to other parallel platforms is also briefly considered.

Chapter 2

Digital Communications

This chapter provides a slighly more detailed overview of digital communication systems than in Section 1.2.1. Section 2.1 discusses the general aspects of digital communication systems, including the design elements of the transmitter and receiver, as well as a model for the wireless channel. Section 2.2 provides a more in-depth description of the various modulation techniques employed by such systems. These techniques play an important role in transforming digital bits into continuous waveforms. Readers who are familiar with digital communications can feel free to skip this chapter.

2.1 Digital Communication Systems

A digital communication system is comprised of a transmitter and a receiver which communicate digital information across a channel, see Figure 2-1. The transmitter converts the digital information, the *Bits In*, into values for transmission across the channel. These values are then sent across the wireless channel to the receiver. The receiver converts these values back into bits, the *Bits Out*. The receiver does this in a manner which attempts to maximize the likelihood that the *Bits Out* output from the receiver matches the *Bits In* input to the transmitter.

The transmitter typically begins by performing digital processing on the *Bits In* to increase the robustness of the data communication link, as well as its usage efficiency. This results in data values which are used to represent the processed *Bits In*. This processing which takes place prior to up-conversion to higher frequencies is commonly referred to as baseband processing. These processed digital values then undergo digital-to-analog (D/A) conversion to generate analog values from the digital information. The radio front end of the wireless transmitter prepares these values for transmission by mixing the analog values with sinusoids to generate waveforms at the carrier frequency. The antenna then transduces these carrier frequency sinusoids to electromagnetic waves which radiate out from the antenna and across the channel.

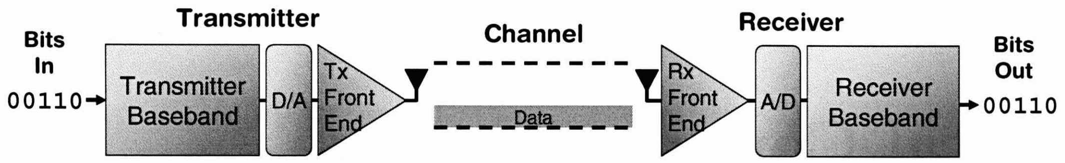


Figure 2-1: A canonical digital communication system block diagram.

The process of transmitting the information signal across the channel reduces its fidelity. The channel conditions, such as fades, path loss, and other propagation effects, as well as interference and channel noise contribute to a degradation in signal quality. The main factor which distinguishes a wireless channel from a wired channel is the changing nature of the channel characteristics and noise.

A common model used for channel noise is the Additive White Gaussian Noise (AWGN) model. While insufficient as an exact channel model, the AWGN model is often used for case studies. It provides a tractable and easily computed noise model which provides reasonable insight into the functionality of a communications system [19, 25, 26]. The AWGN model, see Figure 2-2, consists of a white Gaussian source which provides a statistically independent, identically distributed sequence of Gaussian random variables which are added to the signal as noise. Gaussian white noise is used because it provides a good approximation to the thermal noise of system components [26, 27]. The signal, as it travels through the communications system, is impacted by a large number of small, statistically independent noise sources which cumulatively, as per the Central Limit Theorem, look Gaussian in aggregate [25, 28, 29, 30]. This noise is referred to as white because it looks flat over the frequency range of interest and all of the individual noise sources are independent, like in true white noise. However, it is not true white noise in the sense that the flat spectrum does not extend over all frequencies between $\pm\infty$ (this would require the AWGN source to have an infinite amount of power [31]).

Channel effects also play a large role in the quality of the signal received. Propagation effects such as path loss, slow and fast fades, delay spread and multipath, radiation resistance, and antenna gains can have a large impact on receiver signal strength. For instance, path loss results in the signal power falling off quickly as the distance between the transmitter and receiver is increased (thus increasing the relative impact of many channel noise sources) [32, 27, 33, 34, 35] and is proportional to the inverse of the distance (typically between $\frac{1}{d^2}$ and $\frac{1}{d^4}$). Other effects, such as fades in reception as the receiver moves into areas where the transmitted signal can't reach, or multipath effects where multiple reflected copies of the incoming signal are received, can distort and negatively impact the received signal. Direct interference from other radio sources, such as other transmitters, can also add significant non-signal power in the frequency range of interest, thus degrading the communication link. Channel effects are often expressed as a channel transfer function which is applied to the transmitted signal when it travels across the channel. Receiver functions, such as equalization, attempt to quantify and reverse the channel effects on the received signal, see Equation 2.1. The presence of noise on the received signal, however, makes this reversal of channel effects a non-linear process, and, hence, much more difficult to accomplish.

$$r(t) = s(t) * h(t) + n(t) \tag{2.1}$$

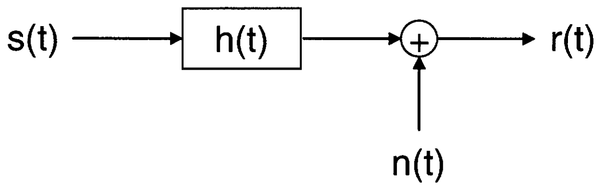


Figure 2-2: Additive White Gaussian Noise (AWGN) channel model.

The Signal-to-Noise Ratio (SNR) is often used as a metric of signal integrity. It relates the power or signal strength of the desired transmission as seen at the receiver to the power of all other extraneous sources observed with that signal. This is sometimes expressed as E_b/N_0, or the energy per bit divided by the noise power spectral density [36, 37]. In general, the higher the SNR, the more likely the information from the transmitter will be correctly communicated to the receiver.

At the receiver, another antenna transduces the resulting electromagnetic waves plus the accompanying noise into analog waveforms. The receiver front-end then down-converts these analog waveforms from the carrier frequency to a much lower frequency or DC. These low frequency signals are again referred to as baseband. The baseband signals are then commonly transitioned from the analog to the digital domain by an Analog-to-digital (A/D) converter. The resulting digital values are processed by the receiver's baseband receive chain in an attempt to recover the information sent and to recreate the transmitted bits, the *Bits In*. These recreated bits are the system output, the *Bits Out*.

The purpose of the digital communication system is to maximize the likelihood of correct and timely reception of the digital information from the transmitter at the receiver. Multiple signal processing techniques are applied to the information to achieve this end. These communication signal processing techniques generally fall into three categories: source coding, channel coding, and channel correction.

Source coding maximizes the usefulness of each bit to be transmitted in an information theoretic sense [19]. This consists of compressing the data such that the most information can be represented in the fewest bits possible. This compression has the effect of minimizing the correlation between bits and subsequently minimizing the number of bits in total. Reducing the total number of bits helps to contribute to the communication system's efficiency, as it results in more information being transmitted per bit sent while not affecting the amount of power required to send each bit. For the sake of the systems addressed in this dissertation, source coding is assumed to have been done prior to bits being presented as input to the digital communication system. The Separation Theorem guarantees that this assumption of a source coder separate from the channel coder does not impact the optimality of communication [38, 28].

Channel coding refers to modulation and error correction and mitigation techniques. It is a means by which to improve the likelihood of successful transmission of data across the channel. There are multiple possible techniques employed to achieve this goal [39]. Modulation techniques map data into waveforms for transmission across the channel. Modulation is discussed in greater detail in Section 2.2. Among the error correction and mitigation techniques are state-based encoding and interleaving. State-based encoding, such as convolutional encoding, adds redundancy and locally correlates the bits through the use of state information in the encoding. This results in a larger number of overall bits, but with a set of bits which are redundant enough, and correlated enough, that errors in bit transmission can often be overcome and the correct bits inferred at the receiver. Interleaving, on the other hand, is an error mitigation technique. It involves changing the ordering in a set of bits such that correlated bits, which have been generated by other channel coding techniques, are not transmitted consecutively. This is done to reduce the impact of any multi-bit errors which might occur during transmission. It is not unusual for interference or propagation errors to effect multiple consecutive bits. Interleaving is a means by which these corrupted bits can be spread out throughout a data block thereby isolating errors and giving other channel coding error recovery techniques, especially those which rely on mutual information between consecutive bits, a higher likelihood of success.

Channel coding is often at odds with source coding. Source coding focuses on efficient representation of bits in order to reduce the number of bits sent, while channel coding inflates the number of bits to maximize the tolerance of the communication link. Therefore, it is necessary to balance the transmission-level requirements of robustness and functionality in the face of error-inducing conditions with the system-level requirements of power efficiency and minimized latency.

Channel correction techniques attempt to undo the effects of the physical wireless channel on the transmitted waveforms. They are mainly concerned with compensating for error sources which are relatively constant over the course of a transmission. Error sources such as current channel effects, hardware offsets, and fading can often be addressed without regard to the specific data being sent. This data independence allows the use of meta-information, such as headers or pilot tones, which can be standardized *a priori* and known by both the transmitter and receiver to empirically determine the compensation parameters for channel correction techniques. Channel correction functions include techniques such as: channel equalization, phase correction, and frequency correction. [26, 36, 30]

An example of a digital communication system which implements a set of these signal processing techniques is presented in Section 7.2.

2.2 Modulation Techniques

Modulation is the mapping of data onto time-varying signals for use by the analog radio front-end. [25, 26, 27, 32] Transduction of signals by the antenna into electromagnetic waves necessarily requires these time-varying signals. Modulation commonly consists of generating a sinusoid whose specific aspects convey a piece of data. Sinusoids are typically used as the modulating waveform for numerous reasons, among them are: the prevalence of Fourier analysis and the ability of sinusoids to act as an orthonormal basis which spans the reals [40]; the convenient mixing properties of sinusoids when multiplied together [41, 23]; the use of frequency as a means of partitioning and dictating the electromagnetic spectrum; and the ease of creating sinusoidal sources in analog electronics.

There are three aspects of sinusoids which are used to encode information: amplitude, frequency, and phase. The type of modulation scheme determines which sinusoid aspect or aspects are used to distinguish data values. For instance, in terrestrial radio, two different modulation schemes are used: amplitude modulation (AM) and frequency modulation (FM). In amplitude modulation, the amplitude of a radio frequency sinusoid, or carrier wave, is directly adjusted so that it matches the data signal. The data signal can then be directly recovered from the envelope of the transmitted carrier wave. In frequency modulation, the frequency of the carrier wave is adjusted based on the data values. The deviation of the carrier wave frequency from its nominal frequency is detected at the receiver and used to recreate the transmitted data.

Complex exponentials are often used to represent the modulated sinusoids. The relationship between sinusoids and complex exponentials is expressed in Euler's formula, see Equation 2.2, and for a negative angle, see Equation 2.3. The value of a sinusoid at a constant phase, such as $cos(\theta)$, can thus be represented as the Real part of $e^{\imath\theta}$. A time-varying sinusoid, such as $cos(\theta t)$, has a phase component, θ, which changes with time, t, where $\theta/2\pi$ represents the frequency of the time-varying sinusoid. A change in amplitude, A, or constant offset in phase, ϕ, would appear as $Acos(\theta t+\phi)$. Euler's formula can be restated to explicitly take into account the sinusoid aspects changed by modulation (phase, frequency, and amplitude), as seen in Equation 2.4.

$$e^{\imath(\theta)} = cos(\theta) + \imath * sin(\theta) \tag{2.2}$$

$$e^{-\imath(\theta)} = cos(\theta) - \imath * sin(\theta) \tag{2.3}$$

$$Ae^{\imath(\theta t+\phi)} = A[cos(\theta t + \phi) + \imath * sin(\theta t + \phi)] \tag{2.4}$$

Euler's formula states that, a single complex exponential can be represented as the sum of two sinusoids: a real, or cosine, sinusoid and an imaginary, or sine, sinusoid. Since both the real and imaginary sinusoids are at the same frequency, θ, the sum of real and imaginary sinusoids combine to form a single sinusoid with a particular phase offset, ϕ. The phase offset desired determines if the complex sum contains only a real cosine part, or only an imaginary sine part, or some combination of the two. Regardless of the relative size of the real and imaginary parts of the complex sum, the norm of the complex sum equals the complex exponential's amplitude, A, as required by Euler's formula. Where, in this case, the norm used is the complex norm.

$$\|x + \imath y\| = \sqrt{x^2 + y^2} \tag{2.5}$$

The complex norm in Equation 2.5, above, is an application of the Pythagorean theorem, Equation 2.6, to signal space where x, the Real value, and y, the Imaginary value, correspond to $cos(\theta)$ and $sin(\theta)$ respectively.

$$cos^2(\theta) + sin^2(\theta) = 1 \tag{2.6}$$

Euler's formula provides a way of representing the modulation output as a complex exponential. Since each of these outputs contains a real and imaginary part, it is common to plot the modulator output graphically as a complex vector in what is called signal space. Signal space is simply a graphical representation of the complex plane with the x-axis representing the real part of a complex vector, and the y-axis representing the imaginary part. Since the real part of a complex exponential corresponds to the cosine term in Euler's formula, it is referred to as In-phase, and the real axis, the x-axis, is labeled I in signal space. The imaginary part of the complex exponential corresponds to the sine term, which is $\pi/4$ shifted in phase from the real term. Thus, the imaginary term is called in Quadrature (for the quarter shift), and the imaginary axis, the y-axis, is labeled Q in signal space.

Signal space is a convenient way of representing constant frequency complex exponentials and modulation schemes. In signal space, the phase and amplitude of an exponential easily translate into the angle and magnitude of a complex vector respectively. A variable frequency modulation scheme would correspond to a rotating complex vector in signal space with the angle of the vector changing in time at a rate relative to the frequency value. As this situation is not conducive to either easy representation or interpretation, signal space modulation diagrams are usually restricted to modulation types that modify phase and/or amplitude only. The modulation schemes used by the systems in this dissertation are of the constant frequency type and can therefore be readily represented in signal space.

The modulation diagrams in Figure 2-3 represent three possible types of modulation using phase and/or amplitude. Equation 2.7 is the canonical equation used to represent these modulation schemes. It shows a complex exponential and its corresponding real and imaginary parts. The grouping of terms highlights the phase, ϕ, and amplitude, A, values which are varied by the modulation schemes, while pulling out the frequency term, θ, which is held constant. The specific modulation types demonstrated in Figure 2-3 are discussed below.

$$(Ae^{\imath\phi})e^{\imath(\theta t)} = A[cos(\phi) + \imath * sin(\phi)]e^{\imath(\theta t)} \tag{2.7}$$

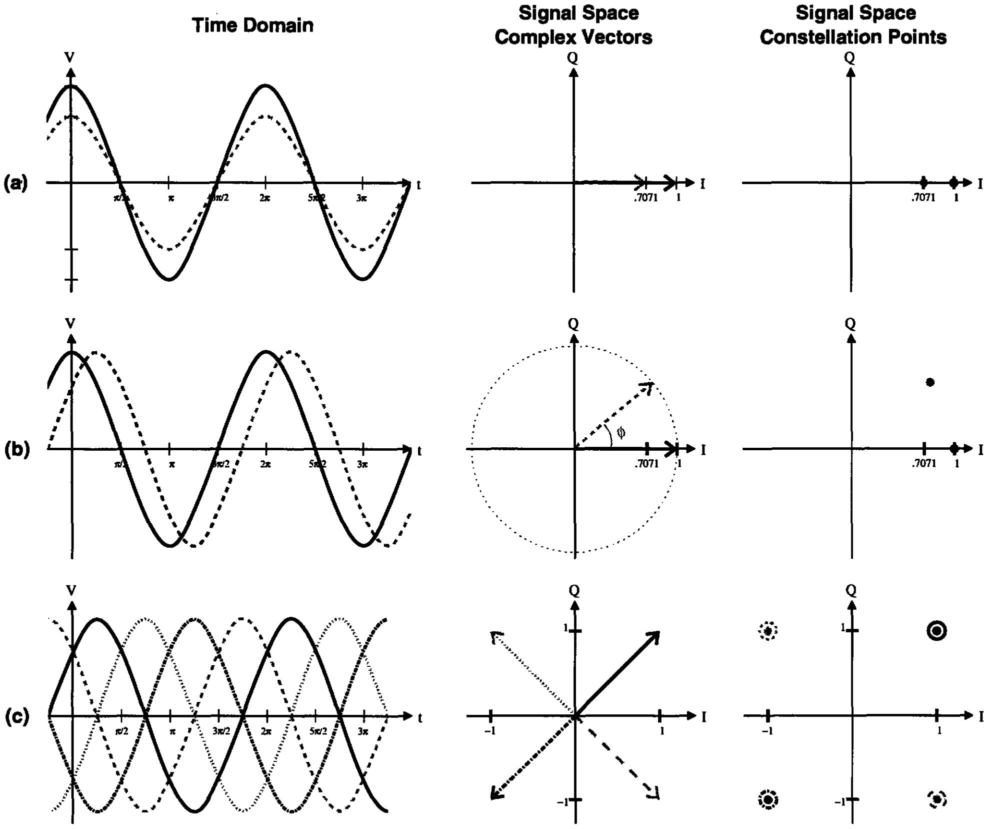

Time Domain
Signal Space Complex Vectors
Signal Space Constellation Points
(a)
(b)
(c)
V
t
Q
I
.7071
1
ϕ
-1

Figure 2-3: Examples of different types of modulation. Each example is shown in three forms: time domain signal, complex vectors, and constellation points. There are three modulation types shown here: (a) Amplitude Modulation (AM); (b) Phase Modulation (PM); (c) Quadrature Amplitude Modulation (QAM)

Figure 2-3 Part (a) shows an amplitude modulation, or AM, modulation scheme. The

two modulation sinusoids can each be represented by an expression of the form of Equation 2.7, and are characterized by a different amplitude values, A. In signal space, the length of the complex vector corresponds to the norm of this expression. In this case, the vector length is shorter for the sinusoid with the smaller value of A and longer for the larger value of A. The signal space vectors lie solely on the I or real axis as both sinusoids here are pure cosines. This corresponds to the phase of the sinusoids, ϕ, being zero and the complex exponential containing no imaginary part. The constellation points in Figure 2-3 are simply signal space points which correspond to the terminal locations of the complex vectors. The (x, y), or in this case (I, Q), coordinates of the constellation points correspond to the real and imaginary parts of the complex exponential and provide a more convenient notational format.

Part (b) of Figure 2-3 displays a phase modulation, or PM, modulation scheme. The phase, ϕ, is modified according to the incoming data and the time domain sinusoid shifts accordingly. In signal space, the complex vector rotates in response to the change in phase dictated by the data value. The relative magnitude of the real and imaginary parts change to reflect the shift in phase and that sinusoid's constellation point is at the corresponding location. Since the amplitude remains constant, however, the length of the vector, and the distance of the constellation point from the origin does not change. This is equivalent to the norm staying constant.

Part (c) of Figure 2-3 shows a modulation scheme in which four different sinusoids are possible. These four sinusoids can be thought of (and implemented) as either a phase modulation with four different phases (a rotation of the complex vector to four different points): $\pi/4$, $3\pi/4$, $-3\pi/4$, $-\pi/4$, or as a combination of phase and amplitude modulation: $\pm\pi/4$, and $\pm A$.

If implemented as a four level phase modulation such a scheme is called Quadrature Phase Shift Keying or QPSK. The term quadrature is used because the final modulated sinusoid is generated by combining two sinusoids in quadrature (sine and cosine), and Phase Shift Keying because there are four keys, or constellation points, which are reached by shifting the phase of the complex exponential. Larger PSK modulations are generated by adding more constellation points. These points are evenly spaced around the circle described by rotating the phase of the constant amplitude complex exponential through a full 2π radians.

As a combination of phase and amplitude modulation, the modulation scheme in Part (c) of Figure 2-3 would be called 4-Quadrature Amplitude Modulation, or 4-QAM. The Quadrature term again denotes that a combination of both a cosine and sine is used to generate the modulated sinusoid. This implies that a phase shift is present (for pure amplitude modulation only a single sinusoid would be needed). The Amplitude Modulation term is used because a change in amplitude value is also used to convey information. The value 4 is used because there are four constellation points. These constellation points are typically arranged in an evenly spaced symmetric square pattern for N-QAM as it is easily generated and energy efficient. It is easily generated because symmetric variations of phase and amplitude are used. It is energy efficient because the constellation points are relatively far apart, which helps reduce errors, while keeping the average of the constellation points relatively close to the origin, which corresponds to lower transmission power. In a digital communication system, the constellation points represent sets of bits and N is commonly a power of 2 so that there is a point representing every potential set of bit values in the fully populated square constellation. Therefore, 16-QAM, 64-QAM, and 256-QAM are the likely larger constellation sizes.

Chapter 3

Digital Radios

In recent years, advances in semiconductor fabrication process technology and micro-architectural design have continued to increase the computational capabilities of digital processors. At the same time, architectural approaches have allowed for increased flexibility and low-level parallelism in digital processors. These two aspects combine with a renewed interest in parallel processing to create an opportunity to use such digital processors to address an improved class of communication problems.

The use of digital processors in radio systems provides the opportunity to exploit specific properties of the digital domain to create a new type of more flexible communication system [10]. Digital processing in wireless systems is commonly used to slavishly re-implement analog techniques in the digital domain. This type of approach may make use of some of the flexibility of digital processing, but it commonly fails to take advantage of many of the inherent strengths of operating in the digital domain. Among these are: infinitely cascadable functions, inherent data synchronization and ordering, relaxed temporal constraints, and the ability to adapt and change the system structure. To profitably operate in the digital domain and optimize the use of digital processing for radios, there is a need to design systems which maximize these benefits while minimizing the associated negatives of digital radio processing: A/D conversion errors, rounding errors, less efficient implementation of individual functions, and the need for explicit control mechanisms.

3.1 Digital Implementation Approach

The need to convert data from the analog to the digital domain is necessitated by the digital nature of the information being sent. There is a design decision, however, in choosing at which point in the baseband receive chain to implement this conversion. Most digital wireless systems perform this conversion at the earliest feasible point in the receive chain. This is commonly just after down-mixing the analog waveforms to a baseband frequency where the A/D converter can operate with minimal aliasing. There are multiple advantages to performing this conversion early in the reception process, both from the point of view of using digital hardware in general, and from the point of view of designing a system with the specific strengths of the digital domain in mind, see Sections 3.3 and 3.4.

The A/D conversion process adds noise to the received signal in the form of quantization noise, sample timing error, and aliasing [42]. Since A/D conversion is a required function, these noise sources are unavoidable in the receive chain. Once the data has been moved to the digital domain, the noise associated with further processing is minimized, see Section 3.3.1. Thus, there is an advantage to performing this conversion as early as possible.

Additionally, once operation has moved to the digital domain, digital design techniques can be used to implement the baseband functions. This means that the baseband implementation has the advantages of digital designs: the portability of design which allows it to be reused between implementation technologies; the modularity of design which allows it to be built, verified, and tested once and then reused between system implementations; the relative ease of simulation; and the ability to use compilation to implement efficient functions in high level languages.

The technologies used to implement digital baseband processing are varied, with full custom VLSI, ASIC designs, FPGA designs, and software implementations on both DSPs and GPPs all used. These different technologies provide a trade-off between fast fixed-hardware implementations and flexible software-based implementations.

The fixed-hardware based approaches, such as full custom VLSI or ASIC chips can be highly efficient and offer the fastest processing capabilities and correspondingly high receiver data rates. These approaches allow the hardware to perform a small set of functions extremely well, but at the cost of the flexibility available in other implementation types. The fixed-hardware approach relies on the spatial reuse of a large amount of fixed, specialized hardware to perform baseband functions.

Firmware-based approaches try to balance the benefits of the fixed-hardware and software based approaches. Firmware is reprogrammable, which offers more flexibility than a fixed-hardware approach. However, the reprogramming of firmware is typically slow and infrequently undertaken. In addition, the inability of FPGAs to access or modify their instruction stream or functionality means that flexibility is limited as compared to a software-based approach. At the same time, however, FPGAs are capable of higher data rates than software-based approaches, although they do not achieve the highest rates of fixed-hardware approaches. The firmware-based approach relies on the spatial reuse of a large amount of partially fixed, partially specialized hardware.

Software-based approaches provide flexibility of design and implementation. They have the added ability to be relatively easily upgraded or subsequently modified, which makes them more cost-effective. This is extremely convenient for fixing errors, adding functionality, or improving implementation methods. The need to reuse the exact same hardware to perform all of the functions, however, restricts the data rates at which the system can operate. These data rates are lower than those of a fixed-hardware solution, depending on the capabilities of the processor used. The software-based approach relies on the temporal reuse of a small amount of flexible, general hardware.

Digital wireless systems which use a software-based approach, however, provide the ability to collocate digital processing for applications and communication in a single device. This provides an opportunity for savings on a number of different fronts. In terms of space, cost, power, and adaptability, the integration of data communications and processing into a single processor could provide an improved efficiency. For example, space savings could be realized reusing already present resources through implementing a single general chip capable of providing all functions, as opposed to providing components for multiple distinct functions but then only partially utilizing those resources. The reduced component count can also translate into reduced system cost. There is an additional improvement from the design cost perspective as software is generally cheaper and easier to design than hardware and is certainly more forgiving of errors, changes, or later improvements.

Additionally, the reduction in the number of components, combined with an increased usage of a single component, namely the digital processor, could also allow power savings to be achieved. The need to power the ancillary specific components is replaced by increased usage of the digital processor. Power consumption is limited to that of the processor, which is already assumed to be required by the system. On the other hand, the relative amounts of power, the ability and time required to shut off the power to various components, coupled with the increased demands on the digital processor may mitigate or reverse these savings.

Finally, by using a general digital processor to implement both communications and computation, it is possible to achieve a reduction in implementation complexity and cost, as well as improved system control. The two functions can be merged into a single device and their usage requirements and capabilities explicitly balanced. This provides an opportunity to exploit the trend towards having more powerful and general digital processing available at both ends of the wireless link to improve the communication system as a whole.

3.2 Software Radios

With the advent of cell phones, pagers, and other wireless devices, the use of digital processors for communication-type processing has grown dramatically. Many wireless systems use some form of digital processing to provide, at a minimum, baseband functionality. Continuing in this trend, software radio systems have been proposed which rely on digital processors to implement an even larger range of radio functions [43, 44, 45]. These software radio systems involve various permutations of minimal analog components (filters, mixers, A/D converters, and amplifiers), along with digital signal processors (DSP) [46, 47, 48, 49, 50] or general-purpose processors (GPP) [51, 52, 53, 54]. The general theme of these systems is to minimize the analog portions of the radio by performing as much processing as possible in the digital domain.

Traditionally, the purpose of software radio systems has been to address the issue of inter-standard compatibility [44, 53, 45]. This dissertation will take a different view of software radios. Instead of attempting to use the flexibility of software radios to provide interoperability, this dissertation will examine some of the novel approaches to communication enabled by a software-based system. Specifically, it will take advantage of freedom from strict temporal constraints imposed by analog circuits and the freedom from static system design imposed by a hardware-only implementation.

There are multiple types of systems associated with the term "software radio". These systems range from those which try to perform all functions beyond the transduction of the antenna in the digital realm, to those which focus on providing a subset of radio functionality in a common programming language on a stock general-purpose processor [55], to those implemented in carefully optimized software on a substrate of DSPs [47].

For the purpose of this dissertation, a precise definition of software radio is not needed. It is sufficient that demodulation and all further processing be performed in the digital domain. Down-conversion to an intermediate frequency (IF), or even baseband, prior to conversion to the digital domain is assumed to be acceptable and may very well be required to build this type of system using present technology. Hence, this type of system will be referred to as a *digital radio* in this dissertation in order to distinguish it from other software radio system approaches.

3.3 Digital Radio Strengths

Digital radios leverage unique aspects of digital domain to improve communication systems. Three digital domain properties in particular lend themselves to new digital communication system architectures. These properties, which will be described and discussed in the following sections, are referred to therein as *infinite cascade*, *data synchronization*, and *temporal shift*.

3.3.1 Infinite Cascade

Moving to the digital domain allows communication systems to trade the compounding noise and loss of analog components for regenerative components in the digital domain, at the cost of quantization noise, rounding errors, and possible conversion range and rate issues. The rectification provided by digital gates allows these regenerating digital components to be chained together without additional signal noise or data degradation. This allows a digital system to string together an arbitrarily long chain of processing elements without concern for noise. Since the noise of the system is no longer dependent on its structure in the digital domain, infinite cascading allows the system to avoid noise considerations in determining the length of processing chains, the number of parallel chains possible, or the elements which make up these chains. The composition of digital processing can even be dynamically chained without impacting the signal noise.

The ability to infinitely cascade functions without a noise penalty is a powerful tool.

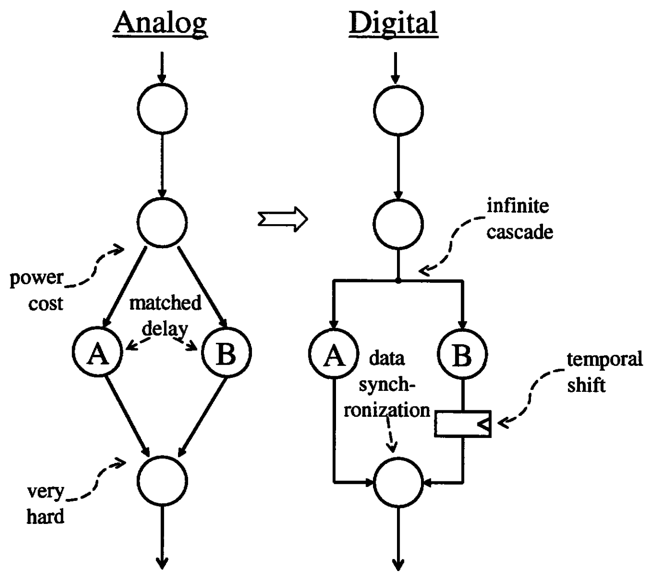


Figure 3-1: The digital domain is especially well suited for functions convenient for novel digital radio systems.

Infinite cascade:	Move, copy, and access data without loss.
Data synchronization:	Timing by sample count allows splits and merges.
Temporal shifting:	Store and retrieve data.

As the complexity and functionality of the system grows, the number of components and functions through which the data must flow grows quickly. For a long enough chain of components, the one-time cost of conversion to the digital domain is dwarfed by the noise cost of an analog implementation. Digital operation allows the design to trade-off very high, but only instantaneous, precision for a fixed small level of error. While rounding error introduced by processing blocks can increase the error in a digital system beyond this fixed level, use of a digital representation format sufficient for the processing required can often mitigate and trivialize this error. Thus, a system which benefits from multiple processing steps is very well-suited to implementation in the digital domain.

The regenerative nature of digital circuits also plays an important role in the flow of data through a system. Since data is restored after each processing element in a digital system implementation, features which can cause degradation of the information signal in the analog domain are trivially overcome in the digital domain. For instance, copying information to multiple destinations can be a lossy or difficult process in a discrete analog implementation. Impedance matching is required for the same signal to be simultaneously delivered to two destinations in parallel, and a careful application-specific circuit design is required for each copying or forwarding of the data. In the digital domain, these interconnections are effortlessly composed since the regenerating digital circuits compensate for any such losses. Therefore, a digital system can split a received data stream into multiple simultaneous streams without affecting the information stored therein.

3.3.2 Data Synchronization

Data synchronization and ordering is an inherent property of the A/D conversion process. The quantization of continuous time data into samples allows data streams to synchronize on these quantized units regardless of time delays. Data flows can then be split and resynchronized merely by processing samples in lock-step. As long as, for a given throughput, samples are processed in order, then their outputs are naturally aligned. This allows the system to perform different functions on copies of the same data stream without a need for explicit synchronization of the output. Matching the heads of the resulting matching data rate streams, regardless of the relative delays introduced by differing processing chains, is equivalent to matching the values in time. These streams are then naturally positioned for merging, like the two sides of a closing zipper coming together. In an analog system, meticulously matched delays would be required to approach the same effect.

Also, since continuous time is broken up into quanta by the conversion process, any characteristic of the data stream may be used as a relative time origin (e.g., a special signal which corresponds to the beginning of a data packet) and the relative position of any subsequent data value can be determined by its distance in quanta from that time origin. Thus, counting data values effectively substitutes for determining time delays since the number of data values can be used as a surrogate for the time base. Hence, data streams which undergo different processing can still be aligned and cycles within the data can be accessed simply by including delays of a fixed number of samples corresponding to the period of the cycle. The actual time delay experienced by any such streams becomes irrelevant; ordering, causality, delays, and synchronization can all be accessed and used, but without the constraints of literal timing.

3.3.3 Temporal Shift

Together, Infinite cascadability and data synchronization help to contribute major benefits to operating in the digital domain: the ability to copy or operate on data without necessarily degrading it, and the ability to merge or compare and temporally correlate data from multiple streams. The use of state elements in conjunction with these properties allows digital communication systems to take advantage of a powerful third property – temporal shifting.

Persistent state allows a digital implementation to store and/or copy data such that it can be used and reused as often as necessary within the real-time constraints on the output. Data no longer needs be tied to the real and continuous time flow of the analog components. This relaxed temporal constraint creates the opportunity to have multiple processing passes on the same set of data. Each pass can then be specialized for a specific task, such as maximizing the detection of a single orthogonal component. Since multiple passes are available, the deleterious effects on other data components of such specialization do not have an impact on the performance of the system.

The temporal shift ability allows data to be stored or buffered and shifted in time, both in an absolute sense and relative to a sample count time base. This is enabled by the previously discussed properties of infinite cascade and data synchronization. The infinite cascade property ensures the act of copying and storing the data does not degrade the signal resolution, while the data synchronization property allows temporally shifted data to be used in conjunction with other relevant data streams (since the relative time bases can be realigned to allow merging, comparison, correlation, or any other multi-stream function). This allows data streams to be stopped and then restarted, subject to latency constraints, to allow separate data streams to align even under uneven delays.

Temporal shifting also allows functions to operate across delays within the same data stream. Many correlation-type functions require that data values be compared or operated on with regard to previously processed samples. The ability to copy and temporally shift these previous data values means that the values remain available for such operations.

3.4 Software Implementation Benefits

Implementing radio systems in software on a digital processor confers many advantages beyond those gained by simply moving to the digital domain. The ability to reuse hardware to implement the maximum functionality with the minimum resources provides a large cost benefit. The general-purpose nature of digital processors allows for arbitrary changes in the composition and function of processing blocks, as well as the data flow in general, to best suit the current needs of the system.

A software implementation on a digital processor allows the same processor hardware to be reused to perform multiple processing passes. In constrast, a fixed digital domain implementation would require specialized hardware for each such pass, with each of these hardware passes incurring a large cost in terms of design time and functional mismatches. Additionally, this effort must be repeated for each potential modification to the receiver. By reusing already present hardware in the software implementation, however, the resource cost for each system modification is minimized because underutilized or superfluous hardware (e.g., for a function which is no longer needed) is avoided. Consequently, a change in the functionality of the system no longer requires costly changes to the hardware.

System flexibility is also significantly increased by a software implementation. Passes can be added, removed, or modified dynamically; *a priori* knowledge of system's extents or parameters is not required. The system can dynamically adjust to the current channel conditions or data rate requirements. For instance, the length and number of orthogonalizing codewords (see Chapter 5) might be increased for a relatively quiet channel. This would allow more data to be sent in parallel over the same channel but would require extra processing passes on reception. A fixed system would not be able to adjust to take advantage of the available channel characteristics.

Chapter 4

Communication Systems on Parallel Processing Architectures

In recent years, attention in the computer architecture community has increasingly refocused on multiple processor systems[4, 56, 5, 6, 7]. A shift to increasing concurrency has occurred in both the embedded and general purpose domains. This provides the opportunity to design communications systems which take advantage of the multiple programmable processors available in a parallel processing system. These types of systems have the potential to move beyond simple reimplemention of analog designs in the digital domain by making use of parallel but independently controlled processors to provide flexible and dynamic wireless functionality.

The use of a multiple processing core architecture in a communication system presents many design challenges and opportunities. The specific processing system design elements have a large impact on the compatibility of the system with communication processing. Design decisions such as the processor interconnection network, I/O capabilities, and overall system topology have a profound impact on the communication systems which can be successfully built on an architecture. The level and types of parallelism available in a parallel processing architecture control what efficiencies, and thus what critical path latencies, can be achieved. These architectural elements all play a role in determining whether a given architecture is a suitable implementation environment for a communication system.

4.1 Parallel Processing Trends

Traditionally, the incessant need for more powerful digital processors has been satisfied primarily through *temporal reuse*. That is, by designing processing elements which can perform functions in an increasingly shorter amount of time so as to free up these same elements for use in the next task. The dramatic increase in the clock frequencies of processors has been the hallmark of this approach [57].

A shift toward partial *spatial reuse*, the duplication of processing elements to perform multiple similar functions concurrently, has been a common element of processor architectures for decades. Super-scalar, vector, and VLIW processors take advantage of multiple parallel functional units to supplement and multiply the performance improvement of temporal reuse. These types of processors are not fully parallel, however, as there are resources which are shared in common by the parallel functional paths. Some subset of issue logic, decode logic, caches, register file, and program counter (PC) are typically shared among all datapaths in all of the above processor types. The shared resources act as bottlenecks and prevent truly independent operation of the multiple datapaths. Attempts to overcome these bottlenecks include compiler instruction-level parallelism (ILP) extraction [58], multi-threading [59], and trace caches [60, 61].

Many limiting factors have made continuing in this trend unattractive. For instance, the increased switching frequency and smaller transistors needed for continued improvement of temporal reuse leads to problems such as increased power density and excessive leakage current. Also, attempts to keep all of the parallel elements of these systems productively occupied, while heroic, have begun to yield diminishing returns, while the design complexity required to overcome the bottlenecks of partial spatial reuse architectures have become prohibitive [62, 63, 64].

Recently, focus has shifted to full spatial reuse to supplement and continue improvements in processing capabilities. Full spatial reuse consists of using multiple independent processing elements in conjunction. Each processing element may utilize some amount of temporal and partial spatial reuse internally, but the sharing of resources does not extend beyond the processing element other than interconnect between processing elements and external interconnect.

While research and scientific designs have explored fully parallel processing for years [65, 66, 67, 68], commercial designs such as Intel's Pentium D series, AMD's Athlon 64 X2 and dual-core Opteron series, IBM's Cell architecture, and ADI's[1] and TI's[2] multi-core DSPs have begun to follow this trend towards parallel architectures. These commercial designs primarily consist of instantiating a small number of previously designed cores on the same substrate with bus-based communication between them. This is a logical step from traditional uniprocessor designs, but does not attempt to redesign the processor architecture specifically with parallel computing in mind. A new class of research designs, however, have focused on more tightly coupled chip level parallel architectures [69, 70, 71, 9, 72]. An overview of one such class of architectures, tiled architectures, is presented in Section 4.5.1. A more in depth look at a specific tiled architecture, the Raw processor [69, 73, 74], is presented in Chapter 6.

4.2 Effects of Interconnect on Communication System Design

The topology, latency, and bandwidth of the interconnect between processing cores can have a profound impact on the overall system design and on the parallelization of individual design elements in a communication system. There is a tension between the latency of transmitting data over the interconnect and of computation on each processing core. This results in a trade-off between the processing load and amount of temporal reuse on any individual processor, on the one hand, and the cost of transferring data over the interconnect, on the other.

In communication systems, the size of data to be transmitted between functional blocks is typically known at design time. The fixed width of the data allows bandwidth constraints to be translated into latency constraints for design purposes. For instance, if the bandwidth of an interconnection link is one byte and the width of the data to be transmitted is one word, then the system incurs an extra latency penalty equal to the time it takes to communicate the additional three words across the link. Since latency, and the associated meeting of time constraints, is the true concern for the communication system, considering bandwidth in terms of latency is appropriate here.

[1]e.g., the ADSP-BF561 Blackfin Symmetric Multi-Processor for Consumer Multimedia
[2]e.g., the quad-core TMS320VC5441 Fixed-Point Digital Signal Processor

Assuming that a sufficient number of processing cores are available for the spatial pipelining required, the time required to communicate between processing cores becomes critical. To maximize performance and keep the data pipeline in constant operation, the system needs to move data quickly from one unit to the next. In most cases, the specific functional latency (the time each function takes to produce its output) is not as vital as avoiding data pipeline stalls. Buffering in the network can be used to absorb this functional latency and keep the pipeline filled. There are, however, functions for which the amount of buffering available may be insufficient and, consequently, these functions have a maximum tolerable network latency to avoid stalls. Examples of this include control functions like header processing, or functions whose latency and data rate combine to require excessive buffering, such as the bottleneck functions described in Section 1.2.2. The cumulative latency of the full system, however, is always a primary concern in a communication system.

The interconnect latency plays a pivotal role in determining which types of spatial parallelism can be profitably exploited by the system. In general, parallelization efforts must attempt to balance computation and interconnection costs. Large computational jobs partitioned at the inter-block level can often absorb the latency of slow interconnect with sufficient buffering. Tightly coupling short and fast computation across processing elements, e.g., at the intra-block level, however, requires similarly short and fast interconnect to avoid stalls. The amount of spatial parallelization available on an architecture is the finest sustainable level of spatial pipelining for which the amount of computation is on the order of the interconnect latency. At that level, further parallelization across processing elements would result in interconnect latencies disrupting the computation and waste functional resources.

The amount of data to be processed can also affect the spatial partitioning of functions. The goal is to balance computation and interconnection, but the accessibility of data can significantly impact the length of computation time and therefore affect functional partitioning. If the live data set is small enough to be stored in the register file of a processing core, then this data is immediately available and will not slow computation. If the partitioning requires too much data to be present on a single processing core, then register spills and cache misses could seriously impact the execution time on that core. Spreading these data values, along with their functions, over multiple cores may take advantage of the storage of multiple processors, as well as buffering of data in the network, to reduce computational overheads. This keeps data values live and out of main memory, and reduces or eliminates cache accesses, so that values are readily available as needed. This solution is only appropriate, however, if the interconnection latency is less than the memory latency. Otherwise such a solution may result in slower processing than just using the memory system.

The topology of the interconnections between and within functions can have a profound impact on the cost of communication [75, 76]. How well each problem or function maps onto the type of interconnect available can profoundly affect the interconnect latency between processing elements, thereby, impacting parallelization efforts. For instance, a function like FFT might map well onto an Ω-network which has log_2 type connections [77, 68]. Each consecutive section of the trellis would only require a single hop in such a topology. However, implementing the same FFT on a mesh network might require many more hops to achieve all the cross routes of the trellis, thereby increasing the cumulative interconnection latency. The longer wires used in hops in an Ω-network, though, may limit the achievable system clock speed and reduce overall performance.

This creates a trade-off between localization of computation and interconnect latency. If a processing function requires feedback and/or data reuse, then performing all of the processing on a single core may be very efficient. It avoids concerns about delay and scheduling of using the network to move around the appropriate data for the next computation. Performing all of the computation on a single processor, however, may result in an unacceptable increase in computation time, or in a live data set which is too large for the register file or cache. Hence, the cumulative interconnect latency of spatially pipelining must be carefully balanced against the computational requirements of temporal reuse.

4.3 Effects of I/O on Communication System Design

To take advantage of increased data processing capabilities, it is necessary to have sufficient data available on which to operate. The ability to move data into and out of processing cores is vital to keeping spatial pipelines and parallel resources profitably occupied. The system I/O resources, whether pins or network connections, must provide enough bandwidth to allow data to move as needed.

Efficient operation is predicated on the ability to keep data flowing at a predictable and constant average rate compatible with the sink and source rates of processing elements. An inability to feed data into the processing elements, even if those elements are sufficiently configured to meet all processing and latency requirements, will result in stalls from data starvation. If data starvation stalls occur, but the processing elements are still capable of meeting latency requirements, however, then that implies the desired communication system may still be possible. Consequently, the system could potentially be achieved at a slower processing rate, or by using fewer resources with more temporal reuse. Conversely, if the system is not capable of moving data out to consumers quickly enough, then data values can back up and cause stalls through backward pressure on processing elements as they attempt to output values.

The concept of a stream of data is a useful abstraction for the data sets consumed and generated by communication systems. Streams are especially useful for signal processing and communications, as signals are inherently streams with an infinitesimal time step between values. The process of quantizing these signals into discrete time steps does not change the fundamental aspects which qualify the now discrete data as a stream. There are four main properties which qualify a set of data as a stream:

Serial ordering Data streams are made up of serial in-order data values. The order of arrival of data corresponds to the correct temporal ordering according to the stream's time base. The first value presented, the head of the stream, represents the oldest temporal value, and the last value seen corresponds to the temporally most recent value. This allows temporal relationships to be expressed through relative position in the stream. This position can be combined with knowledge of the time interval between data values, the stream data rate, to determine exact timings.

Constant rate Streams require a constant steady-state or average data rate. If the consuming system can operate at this rate, there is no concern for data being lost or for data stalls. The possible stream data rates are typically know *a priori* and directly inform the system design. Knowledge of the stream data rate and the amount of time available between data values directly influences parallelization and partitioning.

Liveness Only data in a window around the head of a stream is live. This helps reduce the need for expensive memory, as only a small subset of the data is relevant, or live, at any given time. The smaller the window, the less storage needed. Data may also be output or discarded after its liveness expires and it is not needed for any further computation. The amount of time data values are live, and correspondingly, the width of the window, is a function of the system latency.

Semi-infinite The data length of a stream must be much longer than the window of liveness. This aspect is important to ensure that steady-state assumptions dominate. Transition costs or anomalies, such as stream startup or ending therefore do not dominate and can be mitigated.

These properties are very similar to and encompass some of the digital radio aspects discussed in Section 3.3. This is not accidental – stream processing is a natural approach to communication processing and digital radios take advantage of this model.

Streams are a very useful abstraction in communication systems as they map very well to the system elements. The constant discrete output of an A/D converter looks like a stream. Therefore, any sort of monitoring of continuous signals looks like a stream. In a similar manner, streams can also describe the movement of large amounts of data. While not semi-infinite, a large enough block of data moved as serial values will look, for all practical purposes like a stream. As long as the data length is much larger than the window of liveness, then the streaming model is applicable. The producer/consumer relationship between spatial pipeline elements, or function blocks, also acts as a stream of data, with the consumer receiving a stream of values from the producer of the network and outputting a stream of the processed results to the next consumer in the pipeline.

In order to satisfy the streaming data model, processing elements need adequate external access to move data in the amounts and at the rates required. The system I/O resources must allow sufficient bandwidth for all necessary streams to flow and coexist. In addition, a system could have more than one communication stream operating at the same time and must be able to meet the aggregate system requirements. For instance, a wireless camera could be receiving an image stream while wirelessly transmitting a processed data stream.

4.4 Effects of Reuse on Communication System Design

One of the primary benefits of implementing a communication system in a general-purpose processing environment is the ability to take advantage of multiple different types of reuse. The various types and amounts of reuse must provide sufficient benefit to make operating in this environment profitable. Designs must be fast enough to meet the latency requirements of the communications system and to successfully achieve real-time operation. At the same time, the system must retain enough flexibility to justify implementation using general-purpose processing elements.

The use of general-purpose processing allows access to multiple complementary types of reuse. These reuse types fall into two broad non-exclusive categories: reuse which improves performance to make communication processing feasible, and reuse which enables improved functionality or flexibility in the communication system.

The main barrier to the use of general-purpose processors for high data rate communications processing is the limit of temporal reuse. Uniprocessor GPPs are constrained by their use of a single program counter and set of fetch and decode logic. This inherently serializes the program flow and requires that the same hardware be used repetitively to implement all functions. The primary means of improving the performance of such a system is to decrease the amount of time required to perform each instruction on the same hardware. In order to meet the stream data rate latency constraints of a high data rate communication system, however, the time available per instruction becomes unrealistically small.

The cumulative time required to schedule all of a communications system's instructions precludes the use of solely temporal reuse to meet real-time operational constraints. What is needed is to reduce the amount of work required to a level consistent with the timing constraints. This can be achieved through parallel processing. Communication processing can commonly be broken down into multiple independent functional blocks connected by streams of data in a producer/consumer fashion. Each of these blocks can then be assigned to its own set of hardware, assuming sufficient interconnection is available between hardware elements. Many communications functions are equally capable of being internally pipelined, creating the opportunity for significant improvement in processing latencies through the use of spatial reuse for parallelization. By reusing the same design to create multiple instances of hardware which can operate in parallel, the workload on any given hardware instance can be reduced.

Spatial parallelization could, and traditionally has been, implemented by application-specific hardware design. Multiple instances of functionally specific designs are used to minimize the processing time of any given operation. While such designs are the most time efficient means of implementing functions, they suffer from a lack of flexibility. A separate time-consuming design effort must be undertaken for each operation to be performed, including all testing and verification. These disparate static designs are then fabricated into silicon and cannot be changed or modified without fabricating a new revision of all of the hardware on a die. There is no potential to change the functionality of any block or set of blocks based on conditions, system parameters, algorithm modification or improvement, or system upgrades. The best which can be achieved is to decide *a priori* which conditions, parameters, and functions are most likely to occur and explicitly design to meet that limited set of possibilities. This results in additional design time and complexity, as well as additional hardware which often goes unused except under limited and specific circumstances.

To achieve the processing power of spatial parallelization while retaining the flexibility and adaptability of general-purpose processing requires combining and balancing both spatial and temporal reuse. This can be achieved by using multiple processing elements in conjunction to meet system timing constraints while retaining the flexibility and benefits of general-purpose processing on each element. Spatial pipelining of processing can reduce the temporal load on any single general processing core. This would allow these processing elements to operate within the functional latency constraints, thus enabling their use in a communications system.

One of the primary benefits of operation in the general-purpose processing domain is the ability to make use of software implementations and software reuse. Implementing functions in software allows the use of abstraction layers to reduce design times and difficulties. High-level languages, compilers, and debuggers can be used to more quickly and easily generate designs with a reasonable level of optimization. Designers need not be conversant with the underlying low level implementation issues and can instead focus on system and algorithm development and optimization. Such a high-level approach might not be sufficient to meet the latency requirements for all parts of a system (critical path functions may need to be tweaked by hand or done in assembly) but it allows the majority of the system to be implemented in a sufficient manner and much more easily.

In addition, software can be written once and then reused on multiple processors. A major benefit of design reuse is in the reduced time and effort in building the larger systems. A single design can be used multiple times as part of a larger system. Additionally, the system can be easily expanded through the reuse of previously generated designs.

Additional software instances can be implemented trivially as long as the same type of processing element, or one with the same instruction set architecture (ISA), is used. The ability to migrate designs is now based on the ISA, not on the hardware. This flexibility is a result of software making use of the ISA abstraction layer which is itself more flexible. The result is that in order to modify a system to replicate a function multiple times using multiple processing elements, say to build another receiver chain, no new design effort would be required. The same software implementation can be seamlessly implemented on additional elements.

Another benefit of a parallel general-purpose processing approach is that the system functionality and applications are not limited by the hardware design. The system is able to accommodate disparate uses and it is not necessary to know all future uses at design time. For instance, if a system is composed of multiple general-purpose processing elements, some number of these elements could be used for communication processing when the system is transferring data. Those same elements could later be retasked to perform completely non-communication related functions when no communication is occurring. The freed hardware resources could be used in whatever manner is most useful at the time, but the system would still retain the ability to switch resources back to communication, if necessary.

A GPP implementation allows system functions to be predicated on current conditions. For instance, the complexity and precision of the channel correction techniques used could be based on the current channel conditions. In less-noisy environments, simple channel correction techniques could be employed using a minimal number of processing elements. Thus, as the noise increased, more ambitious channel correction software could be employed using additional processing elements.

Another benefit is the ability to adjust the level of spatial pipelining employed by the system to accommodate the current communication data rate. For a lower data rate with longer latency requirements, the system could rely on the temporal reuse of fewer processing elements. An increase in data rate, and corresponding reduction in acceptable latency, could be addressed by spreading out the work over more processing elements with fewer instructions per element.

A further benefit of operating in a general-purpose processing environment is that system changes can be translated into software changes. Changes can be easily enacted through software updates, as opposed to requiring a complete refabrication of the hardware. Upgrades, improvements, bug fixes, and even migration to new communication specifications can be realized through a single set of software modifications and then distributed to all instances of the hardware. New physical devices are not required for each slight, or major, modification.

4.5 Parallel Architectures for Communication

The main function of a communication system is the transfer of information in a timely manner. The major difficulty in designing communications systems using general-purpose processors lies in achieving sufficient performance to guarantee real-time operation. The efficiency and type of implementation can easily limit the communication data rate and impede communications.

The key to achieving the performance necessary for high data rate communication using general-purpose processing is parallelism. For most systems, it is not possible to run all necessary functions serially because to achieve suitable performance, an unreasonable amount of temporal reuse, and a correspondingly high clock speed, would be required. The solution is to spatially pipeline functions across multiple processing units. Functions can be spread out among processing cores and the data registered between cores, creating a pipeline of functions across multiple cores. Performing multiple functions contemporaneously by distributing them across multiple processors reduces the amount of temporal reuse required for any single core and allows higher data rate processing to be achieved.

For sufficiently low latency interconnect between cores, the same design philosophy can be applied on an intra- as well as inter-function scale. Some receiver functions require so much computation that a single core is insufficient to complete the function in a reasonable time frame. For instance, the Viterbi decoder and many of the other bottlenecks discussed in Section 1.2.2 require large amounts of processing. Such functions can themselves be spatially pipelined and multiple processing cores may be used in conjunction to achieve acceptable latencies.

Latency between processing units has a significant impact on the types of parallelization which can be used and the level of parallelization employed. A comparison of the various types of multi-processor core systems and some of their relevant parameters is presented in Table 4.1. An overview of each of these system types is presented below, followed by an examination of relevant system aspects for communications.

Multi-Core System Comparison			
	Min. Communication Latency	*Number of Cores*	*Interconnection Scale*
Tiled	1's of cycles	Tens to hundreds	Tile
CMP	10's of cycles	A few	Chip
Multi-chip	100's of cycles	Up to thousands	Physical System
Networked Cluster	>1000's of cycles	Arbitrarily large	Network

Table 4.1: Comparison of parallel architecture systems.

4.5.1 Tiled Architectures

Tiled architectures are characterized by multiple replicated general-purpose processing cores connected by low latency interconnect. Such architectures are designed to improve system performance by taking advantage of their high bandwidth, low latency interconnection network to harness multiple lower complexity general processing cores in parallel, see Figure 4-1. There has been significant recent research interest in tiled architectures, with many different design approaches currently being investigated [4, 70, 71, 9].

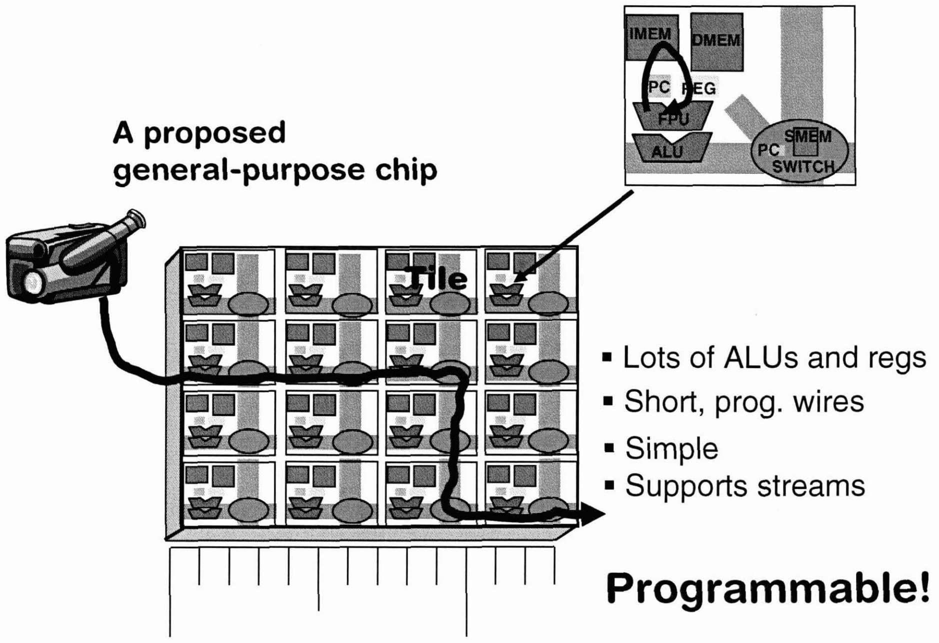


Figure 4-1: An example of a Tiled Architecture [78].

Each of a tiled architecture's processing elements can operate independently, with coarse-grained parallelism, or tiles can operate cooperatively, with fine-grained parallelism. When using coarse-grain parallelism, multiple loosely related functions operate independently on different processing elements with little communication between them. When using fine-grained parallelism, elements operate in conjunction with large amounts of inter-element communication and data value interdependence between elements. This allows tiled architectures to offer a significant level of flexibility. Fine-grain parallelization allows processing elements to work cooperatively with each element performing a few very small specific software functions in order to achieve performance closer to that of a hardware or firmware based solution. But coarse grain parallelism allows these same elements to perform in a manner similar to other multi-processor systems, with independent processing tasks assigned to each element. This mix of fine-grain performance and coarse-grain functionality is very powerful.

There are several features which are common to tiled architectures. These are:

Uniform replication Having multiple processing elements on a single die is achieved by replicating a single core element design. All processing elements are identical, thereby reducing design time and effort. Having uniform elements also allows application implementations to be arbitrarily migrated on-chip.

General purpose While the level of complexity present in each processing core is a matter of research, a common aspect of tiled architectures is that each element is general purpose in nature. Each core must be able to perform arbitrary computation. This allows a tiled architecture extreme flexibility without compromising the uniform replication requirement. Some designs implement a superset of this requirement by including specialized structures within a core to improve performance on specialized functions, yet each core is still capable of arbitrary computation.

Individual program counter (PC) Each core must be able to have its own control flow. The capability to have disparate instruction streams on cores allows coarse-grain parallelism by enabling unrelated functions or applications to run on different cores. The availability of both coarse and fine-grain parallelism is imperative to providing enough work to keep as many cores productively occupied as possible.

Low latency, high bandwidth network The interconnection between processing cores must be on the order of a few cycles. This is necessary in order to make use of multiple cores cooperatively to address fine-grain parallelism. Low level parallelism, especially containing data dependencies, requires low latencies between parallel units to avoid stalls which reduce or eliminate the performance benefits of parallelization. The network must also have high enough bandwidth to minimize communication bottlenecks.

Scalable Tile architectures are designed to be scalable. Scalable designs are insensitive to the sizing of the processor, both in terms of the number of cores and the size of each core. This means that arbitrarily large tiled structures can be generated through uniform replication of cores. Any available die area can be profitably employed simply by adding more cores. Additionally, scalability results in a core design which can be easily migrated across process generations, as the relative timing of each core must be self-consistent.

Scalability also requires that cores, and their interconnect, contain no global structures, such as chip-length wires. Global structures would require significant design modifications if the number or size of the tiles were changed, as any change in process sizing would change the timing of structures which fail to shrink relative to the new process sizing.

Local communication Short wires are critical to reducing the latency of the interconnect. Connecting each core to all its local neighbors results in fast inter-core interconnect with high bandwidth between all neighboring cores. Wire distances, which are sized relative to the size of a core, provide a logically consistent interconnect with scaling. Interconnect which is predicated on distances in number of cores traversed as opposed to absolute distance automatically adjust appropriately as cores sizes are decreased.

Local communication also promotes fine-grain parallelism. Consistent and fast timings are necessary for the exacting synchronization necessary for many fine-grain parallelization tasks. Programmable interconnect allows communication patterns to be determined and implemented statically to tightly schedule the cooperative utilization of cores.

Support streams Stream support is vital to tiled architectures. The ability to stream data allows fine-grain parallelized applications to avoid stalls associated with being starved for data. High bandwidth I/O is necessary to allow sets of ordered data values, or streams, to be transferred directly onto the interconnection network of a tiled architecture. This allows designs which are able to keep consecutive data live to allow operation on large amounts of data with minimal memory accesses. The interconnection network provides a form of storage as "in flight" values avoid the need to process data through memory.

Tiled architectures are appealing for use in communication systems because they allow enough spatial reuse to achieve high processing rates, but enough temporal reuse to retain flexibility. Such architectures provide the ability to take advantage of spatial reuse but through interconnect which is low enough latency to allow for real-time design. They provide the generality and flexibility to allow resources to be accumulated or released as needs dictate. They also provide a uniform substrate on which software can run, allowing each core to use predefined software to seamlessly change its function and to dynamically build new system topologies.

4.5.2 Chip Multi-Processor (CMP)

Chip Multi-Processors (CMP) typically consist of multiple previously designed processing cores implemented on a single die, typically connected by a bus. These designs usually consist of a few small number of cores (e.g. 2, 4, 8) on a single chip. CMP's often represent an attempt to more profitably make use of newly available die area than simply increasing cache sizes by improving performance through coarse-grain parallelism instead.

CMP's are currently used most often as replacements for uniprocessor systems, and, consequently, are commonly used to run software originally designed for the uniprocessor domain. As a result, CMP's typically use high level parallelism, such as thread level parallelism, to try and utilize the additional processors available without significantly restructuring the software being used.

However, the flexibility of CMP's may be limited by the latency of the interconnect between processing elements which is typically on the order of tens of cycles. The opportunity for fine-grain intra-function parallelism is reduced by the long latency of the global scale interconnect. Fine-grain parallelism is often precluded by the low bandwidth imposed by the interconnection bus and the limited number of processing elements available.

CMP designs typically take a relatively short-term view of technology scaling. These designs incorporate the ability to increase the number of processing elements and/or amount of interconnect by a small constant factor in order to utilize near term processing technology or die size improvements. Such designs typically do not consider the long term impacts of technology scaling, however. Design characteristics, such as long wires and very complex cores, preclude CMP architecture scalability across many future technology generations. In addition, the interface network between processing elements, memory, and I/O are not designed to scale as the number of cores grows larger (as would be the case in future technology generations).

Limitations on the fine-grained parallelism available on CMP's may preclude the parallelization of some communication system functions, while limited interconnect bandwidth may cause data stalls. Such constraints typically restrict the data rates attainable by CMP-based communication systems.

4.5.3 Multi-chip Systems

Multi-chip systems, as the name implies, consist of multiple distinct processor chips. Each chip occupies a full die and has its own package. The chips are connected together either on a single board or on multiple boards in a single machine through a macroscopic interconnect system such as a back-plane. Multi-chip systems have been designed using many types of interconnect and network topologies but in all cases each processor is independent and interconnections are all external to the processor. The number of processors in such systems is often variable and can be modified by adding additional boards to a back-plane.

In multi-chip systems, the interconnections between elements occur in the fastest case at the pin level, and possibly at the board level. This results in relatively long latencies for communication between elements, on the order of hundreds of cycles in modern multi-chip systems. Again, the latency of the interconnection network limits the types of parallelism available.

Many more processing elements are potentially available in the multi-chip system than in the CMP case. Although these elements are less tightly coupled than on a CMP, the parallelism utilized by a multi-chip system may occur at a lower level in order to make use of these extra parallel resources. This is a reflection of the different intended operating regimes of the current designs. Multi-chip systems have traditionally been designed primarily for scientific computing. Algorithms and applications which make use of such machines are painstakingly and specifically designed to partition the required processing and data and to hide interconnect latencies. Multi-chip systems are not capable of supporting parallelism at the intra-function latency sensitive level, however.

Historically, there have been multi-chip systems which made use of very fine-grain parallelism. These systems typically operated in a regime where the clocking of the system was so slow that the interconnect latencies were overshadowed by the computational latencies. However, processing speeds have increased much faster than the speed of interconnect and this assumption no longer holds.

The relatively long latencies of the interconnect between processing elements make multi-chip systems unsuitable for high data rate communications systems which have any data flow that is not strictly feed-forward. Even in the feed-forward case the cumulative interconnect and processing latencies must be low enough to meet the maximum latency requirements of a communication system.

4.5.4 Networked Clusters

The logical extension of a multiple processor system is a cluster of independent computers connected through an external network, e.g., Ethernet. In this case, communication between computers occurs at the application level, as the interconnection network requires such niceties as the use a protocol stack. This results in network latencies of at least thousands of cycles.

Networked clusters can be very useful for solving computationally demanding problems, but communication is so expensive for these systems that they are typically only used for latency insensitive problems where the computation time required is orders of magnitude larger than any interconnection time. The long latencies of such systems are mitigated by the fact that the individual computers are designed to operate independently and usually do not share resources. Networked cluster computing focuses on partitioning problems and data across individual computers and then reassembling the results. Therefore, networked clusters operate at the coarsest grain of parallelism.

Networked clusters are typically unsuitable for implementing a high data rate communication system. The extremely long interconnection latencies make it impractical to harness a cluster of computers to cooperatively implement a high data rate communication system. However a networked cluster solution might be appropriate to implement multiple loosely related communication systems. This might be useful, for example, in building a base-station system which could simultaneously communicate with multiple users.

Chapter 5

Multipass

Performing wireless radio functions in the digital domain on a digital processor allows multiple iterative or parallel sets of operations, or passes, to be performed on an input data stream. A system which utilizes multiple processing passes will be termed a *multipass* system and be shown to enable more sophisticated and flexible demodulation, detection, and signal processing schemes to be used in recovering information. Multipass systems are better suited to take advantage of the trade-offs and benefits of operating in a general-purpose processing digital domain. This, in turn, allows multipass systems to provide functionality which is more adaptable, cost effective, and convenient than traditional digital communication systems.

The passes of a multipass system can operate independently, giving rise to multiple, possibly unrelated, outputs from the same input stream; or passes can be progressive, with the output from each pass informing successive passes; or some combination of the two. One approach is to use individual passes to maximize the distinctions between different information elements. This improved symbol differentiation, or orthogonality, enables better communication systems with higher data rates and increased capacity.

Multipass receivers, like all receivers, precondition the incoming waveform and then perform detection to attempt to reproduce the desired data stream. The key to multipass systems is that each pass can perform a different type of preconditioning and detection. The preconditioning of each pass can be optimized to complement the detection of a specific piece of information contained in the data. These data pieces can together form a more accurate representation of the transmitted data or form a representation of a larger set of transmitted data.

Multipass systems can also be used to overcome the bottlenecks of baseband communication. Multipass provides a means to compose or make use of multiple receivers to meet data rate constraints at lower processing rates. Each pass can perform a receive chain function and the operation of multiple passes in parallel can reduce the temporal reuse load required for any given pass. Multipass allows spatial reuse of functions to act as an alternative to temporal reuse and a means to load balance between the two, see Figure 5-1.

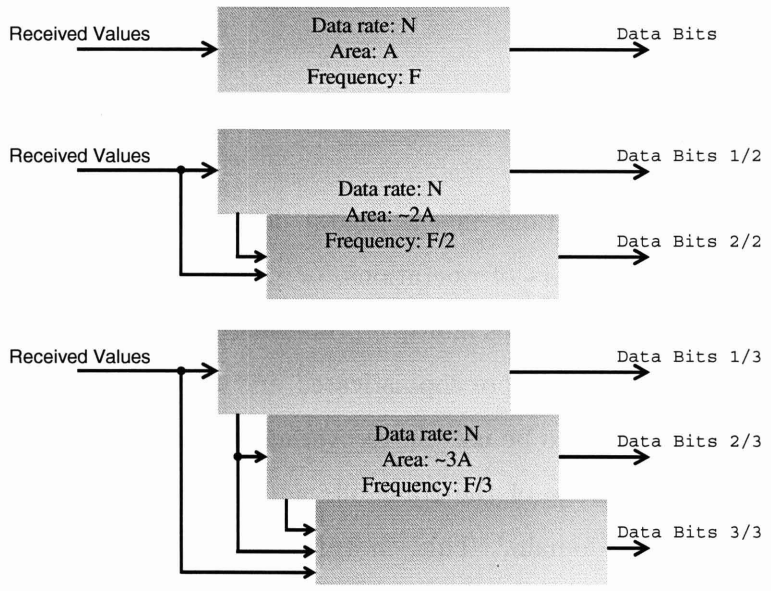


Figure 5-1: A multipass system has the ability to use multiple receive passes to trade temporal reuse (frequency) for spatial reuse (area). An increase in the amount of area used translates to a lower minimum clock frequency. A 2-pass multipass system can use twice the area of a single pass system to operate at half the speed. The reduction in clock frequency could make an otherwise unimplementable system feasible. A lower clock speed would also enable other benefits, like power savings through voltage scaling or the use of higher V_t devices.

This trade-off between area (in the form of additional processing passes) and required processing rate (each pass operates at a fraction of the overall processing rate) allows the system to operate at a lower required clock rate. A reduction in required system clock rate has multiple benefits. One benefit is an increased scope of viable systems, as the decrease in the necessary processing rate allows sufficient computational slack to enable the use of less application-specific implementation methods (e.g., FPGAs, DSPs, GPPs, etc.). Another is an increased ease of implementation due to relaxed circuit timing constraints. Additionally, a lower system clock rate can be leveraged to reduce the power consumption of the system (see Section 5.4.1).

Multipass systems allow the layering of data streams over a channel. A single channel may be used to send multiple streams of data which are then separated and individually decoded by the processing passes. These layers could be designed to interact in an iterative-type approach with each pass providing information for the next (as in the example below in Section 5.2); or passes could be distinct in a parallel-type approach, as if multiple separate wireless links were in operation which just happened to make use of the same analog front-end and channel (see Section 11.1).

5.1 Concept

The multipass systems explored here are designed to overcome the bottlenecks of communication processing on general-purpose digital processors. The use of a multipass design allows the system to explicitly manage the trade-offs between interconnection and computation and spatial and temporal reuse in order to find the balance that best suits the given implementation and conditions. The multipass design provides a means of utilizing a multiple processing core substrate to allow the communication subsystem design to adapt to system-level constraints such as current channel capacity, processing resource availability, and temporal processing limitations. By utilizing spatial pipelining, multipass systems can achieve real-time latencies, as required by communication processing, which would be unachievable using any single general-purpose processor. This becomes especially apparent for the "hard" functions which comprise communication system bottlenecks, such as in symbol demapping, where there is a potential to generate a large number of bits from a single pair of symbol values, and in channel decoding, where a large amount of processing may be required for each of those bits.

Multipass acts as a type of multiplexing and demultiplexing for communication of information across a channel by breaking up the communication workload into multiple, parallel sets of work. Multiple streams of communication are layered onto a single radio front-end and wireless channel, see Figure 5-2. At the transmitter, multiple streams of data are merged into a single stream of multi-layered data. This single stream of data is then transmitted across the wireless channel and received as a single stream by the receiver front-end. The receiver provides a copy of this received stream to each of the receive passes which then isolate a specific layer of information. These individual layers are decoded by their respective receive pass to retrieve the transmitted data.

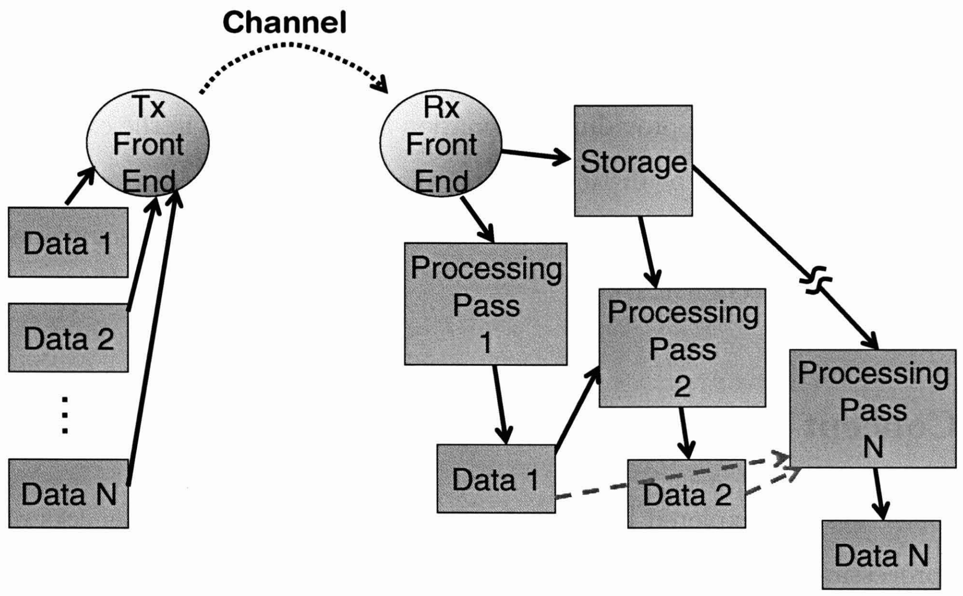


Figure 5-2: Multipass general concept.

The merging of multiple input streams at the transmitter is performed by a combining function. The choice of combining function has a large impact on the system. The physical system constraints, type of orthogonalization, available data rates, impact of various types of noise, system latency, and general complexity of the system are all dependent on the combining function chosen. The type of benefit derived from using a multipass system is also tightly coupled to the combining function used. For example, a combining function that merges passes which remain completely independent might be better suited for systems where all data is equally relevant and the goal is to achieve higher data rates or lower processing rates through parallel data transmission. A combining function which relies on multiple progressive processing of passes might be better suited to situations where there is a hierarchy of data relevance and reception of one data stream at the potential expense of another is desired. The degree of correlation between processing passes is related to the combining function used and defines the amount of interdependence of data layers.

The example multipass system implemented here, as proposed in Section 5.2 and implemented in Chapter 7, uses superposition as a combining function. Superposition was chosen as a simple proof-of-concept combining function. It is by no means the best combining function in all cases, but it is sufficient to highlight the feasibility and capabilities of multipass systems. Other choices of combining function could have a profound impact on the useful operating regime and characteristics of a multipass system. Other combining functions are left as future work and are discussed in Section 11.1.

There are numerous benefits to employing a multipass approach to digital communication system design. The benefit of primary interest here is the ability to make use of spatial resources to relax temporal constraints. Many of the other benefits of multipass systems are discussed in Section 5.4. In this case, the multipass system is a means of utilizing any available processing resources to augment the communications system. This can be done either by dividing a current transmission link into multiple passes and reducing the overall processing rate required while holding the communication data rate constant, or by using additional passes to transfer new data streams and increase the overall data rate.

The decision of which scenario to adopt, higher data rate or lower processing rate, is very dependent on the current system conditions. Fortunately, the flexibility of the system allows it to readily move between either scenario as needed. Many aspects are factored into the decision of how to adjust the multipass system.

First and foremost, the channel, and more specifically the Signal-to-Noise Ratio (SNR), must be sufficient to deliver the required data rate. Depending on the combining function used, multipass can have a negative impact on the SNR. Each pass will look like noise to any other pass. Therefore, there must be sufficient signal present at the receiver to overcome this apparent increase in the noise level (see Section 5.5). If the SNR is not sufficiently high, then use of a higher data rate multipass will significantly degrade signal quality and result in an insufficient communication system. In the lower processing rate scenario, however, it is already assumed that the channel is sufficient for the data rate in current use and moving to a multipass system should be feasible.

In addition, the higher data rate scenario might require more processing to consume the larger amount of data arriving at the receiver. This additional processing may require those processing resources which were just allocated to increase the communication rate. The system will require a careful trade-off between communication and computation in this case, but multipass systems have the capability to explicitly and flexibily manage this trade-off. In the lower processing rate scenario, there is still a tension between communication and computation. While using multiple passes might allow the communication portion of the system to reduce its processing rate, if the computation portion cannot follow suit then there is no real benefit unless a multi-rate system is used. In fact, depending on the functions implemented, computation at the lower processing rate might be feasible but may again require those resources recently allocated to communication. Thus, the communication and computation trade-off must be closely managed.

5.1.1 Digital Domain Enablers

Multipass systems take direct advantage of the strengths of the digital radio discussed in Section 3.3, namely, infinite cascade, data synchronization, and temporal shifting. Figure 5-3 expands the comparison presented in Figure 3-1 to include a multipass type system.

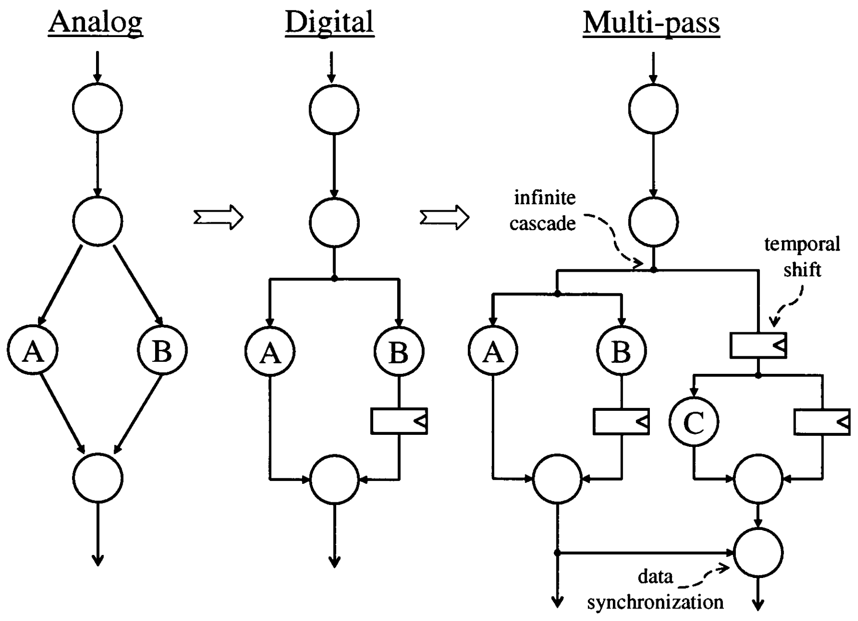


Figure 5-3: Noise source comparison between implementation types:
A circle represents a function and incurs a noise penalty in the analog case, but may not in the digital. A split incurs a noise penalty in the analog case, but not in the digital. Noise grows with the processing chain length and width in the analog case, but not in the digital. Synchronization of data, while hard in the analog case, requiring specialized circuits and resulting in jitter and added noise, is simple in the digital case. The multipass system retains all the benefits of the digital case, plus it has the flexibility to dynamically implement multiple concurrent processing chains and arbitrarily change functions.

Infinite cascade is used by multipass systems to copy data for the multiple passes without loss of resolution. This allows the same data set to be operated on multiple times in parallel without any losses associated with the number of passes or frequency of splits. Another benefit of infinite cascade is the ability to modify the number of functions in a pass without negatively impacting the data quality. This means that passes do not have to be carefully limited in design or carefully matched to avoid or mitigate the accumulation of noise along each pass.

Data synchronization is used in the interaction between the various passes. It allows the output of a pass, or part of a pass, to be aligned with and used in a different pass. It is also necessary for keeping track of the data outputs of the various passes. It is vital if any of the data is to be merged, re-aligned, or compared, or if the data will interact in any way between passes. Even if the data passes are independent, though, data synchronization still plays a role. Data synchronization is necessary for any splitting or merging of data within a pass, but is, even more importantly, needed to keep track of high-level data alignment issues. Since all passes use the same physical front-end hardware, all of their data must be sent in a single compound packet of information. Data synchronization is needed to keep track of the packet-level information, such as detailing where a packet or data chunk starts and ends, as well as to trigger the appropriate control and processing.

Temporal shift also works in conjunction with data synchronization to allow passes or data streams to interact without the need for explicit delay matching. Storage elements can be used to arbitrarily delay or store data streams until they are needed. The ability to perform data synchronization means that when the stored data is ready to be employed it can easily be matched to any other data streams. Temporal shift enables iterative interaction between passes. Data streams can be stored or buffered in memory. Thus, some passes may store a copy of a data stream while other passes are processing. The results of the processed passes can then be used to assist with processing the stored data streams. The example below, in Section 5.2, shows one way in which this interaction could be used.

5.2 Multipass System Example

A 2-pass multipass system is presented in this section, see Figure 5-4. Two data streams, Data1 and Data2, are combined at the transmitter. For both ease of implementation and didactic reasons, superposition is the system combining function used. The resultant single multipass stream is then transmitted across the wireless channel from the transmitter analog front-end to the receiver analog front-end.

At the receiver, the multipass stream undergoes packet level processing, such as header processing and channel correction. The corrected multipass stream is then copied, with one copy being sent to each pass's receiver chain. The Pass2 data stream is stored pending the processing of Pass1. The Data1 stream is then reconstructed by the processing of the Pass1 receive chain and this output is itself copied. One copy is presented as the Pass1 receiver output, Data1′, while the other is used in the recovery of Pass2. This copy of the received Data1′ is partially retransmitted. The purpose of the retransmission is to attempt to recreate a copy of the noise-free Pass1 stream used prior to the combining function at the transmitter. The recreated Pass1 stream is then subtracted from the stored single multipass stream in order to try to remove the effects of Pass1 from the multipass stream, leaving the Pass2 stream as the remainder. The remaining Pass2 stream is then scaled according to the combining function scaling used at the transmitter. The scaled Pass2 stream is processed by the Pass2 receive chain and the resulting Data2′ stream is presented as the other receiver output.

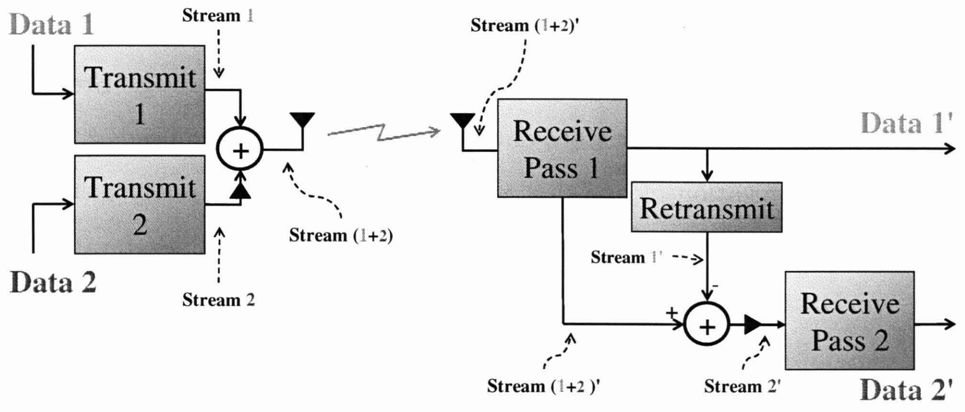


Figure 5-4: 2-pass multipass system example.

Each conceptual stage of the example multipass system will be shown in greater detail in the following sections. The data rate configuration shown uses a Quadrature Phase Shift Keying (QPSK) modulation scheme on both Pass1 and Pass2. As such, it is referred to as a QPSK:QPSK 2-pass multipass system, where the first QPSK preceding the : denotes the Pass1 modulation scheme used, and the second QPSK following the : denotes the Pass2 modulation scheme used. This notation will be used throughout this document. The QPSK:QPSK system provides the same data throughput as a single pass 16-Quadrature Amplitude Modulation (QAM) scheme. An implementation of this system is presented in Chapter 7.

5.2.1 Transmitter

The transmitter in a multipass system is responsible for individually processing each input data stream and then combining these streams into a single multipass data stream. Each data stream is individually processed by a transmit chain up until the point where the data bits have been channel corrected and mapped into signal space representation. The data streams are combined prior to, and independently from, all packet-level information. Since non-data-payload information, such as training symbols and packet headers, apply to to all data streams, this information is generated in a similar manner to the single pass case and then applied to the post-combining multipass data payload.

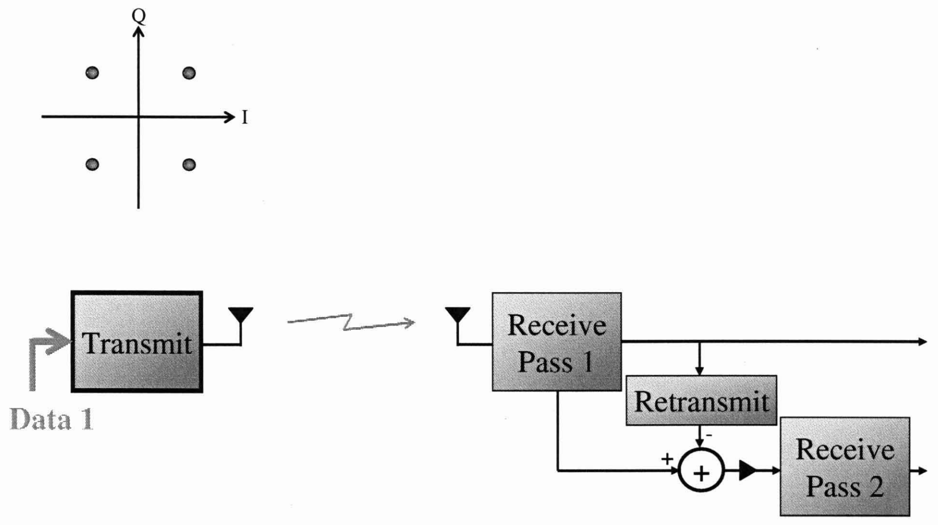


Figure 5-5: QPSK:QPSK 2-Pass Multipass System Example – Transmit Phase I.

The Data1 stream undergoes QPSK modulation where each set of two data bits is mapped into one of four signal space points, as shown in Figure 5-5. The data would then, in the single pass case, be forwarded to the D/A converter and the analog front-end. Instead, the Data2 stream also undergoes transmitter baseband processing. Although it is not required that both passes be of the same modulation type, for didactic reasons QPSK is also used for the Data2 signal. The processed symbols of Data2 are then combined with the processed symbols of Data1 using a combining function.

The function used here is superposition with scaling. Superposition is achieved by first scaling down one of the inputs to be combined, in this case Pass2, and then superposing the scaled signal on top of the unscaled. Superposition is used as a simple proof-of-concept combining function, but other functions can be used to optimize different signal properties. The use of other possible combining functions is discussed as future work in Chapter 11.

Figure 5-6 shows the result of the superposition process. Each of the larger, central dots in each quadrant represents, in signal space, a possible Pass1 symbol to be transmitted. The smaller four dots surrounding each possible Pass1 symbol point represent possible Pass2 symbol points. For any actual set of symbol outputs for Pass1 and Pass2, the Real and Imaginary symbol values resolve in Phase I to the coordinates of one of the four larger dots. From there the combined symbol value moves to the coordinates of one of these smaller circles. The superposition and scaling combining function can be thought of as using the Pass1 possible values as just an origin shift before applying the Pass2 values. The smaller distance of the Pass2 values from their relative origin (the Pass1 values) as compared to the Pass1 points from their origin are a result of the down-scaling of the Pass2 symbol power.

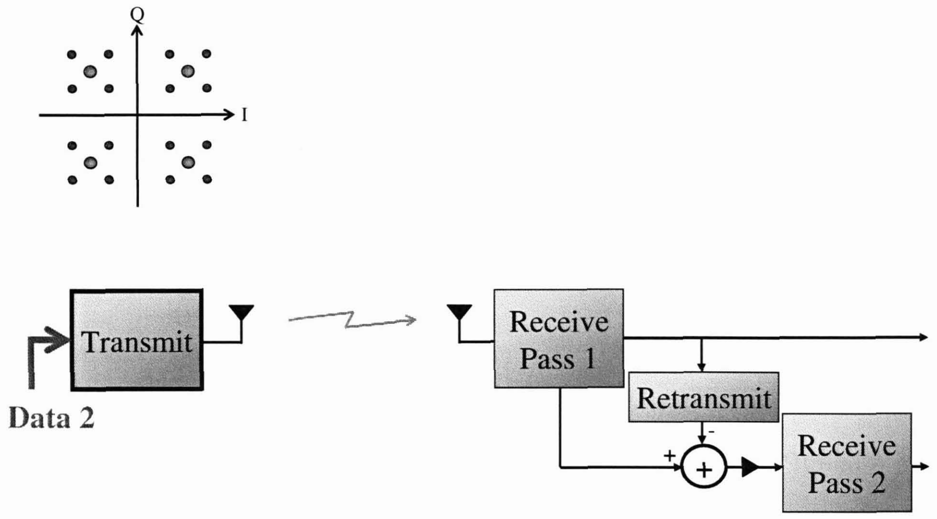


Figure 5-6: QPSK:QPSK 2-Pass Multipass System Example – Transmit Phase II.

The scaling down of Pass2 is done to minimize the interference of the addition of the Pass2 symbol values on the Pass1 symbols. There is a trade-off between the amount Pass2 is scaled down and its impact on Pass1. The scaling of Pass2 reduces its power and makes the signals of this pass more susceptible to noise. Hence scaling can negatively impact the ability to receive Pass2.

From the perspective of Pass1, however, Pass2 looks like additional noise as the two passes are independent. Therefore, Pass1 benefits from a large amount of Pass2 scaling.

There is an imbalance, however, as the results of Pass1 are used to help in the recovery of Pass2 from the combined multipass signal at the receiver. Thus, a too large scaling value has a negative impact on both Pass1 and Pass2. In a system where both passes are equally valued, it is necessary to balance the scaling of Pass2 to maximize the likelihood that both signals are correctly received. In a case where data can be prioritized, however, the Pass2 scaling value can easily be adjusted to reflect the relative priority of Pass1 over Pass2, thus increasing the likelihood of correct Pass1 reception at the expense of Pass2. A more in-depth exploration of the impacts and trade-offs of the superposition combining function scaling can be found in Section 8.2.

In this example, the constellation of signal space positions for the combined multipass symbols corresponds to the constellation for a 16-QAM modulation scheme. This is by design. The 16-QAM constellation was chosen for two reasons. First, it is an efficient packing in signal space. It provides a large amount of space between signal points, thus increasing the amount of noise power necessary to cause an error, while minimizing the signal power in the transmitted signal. This is equivalent to minimizing the distance of the signal constellation points from the origin. It is therefore a good overall choice. The second reason this constellation is used here is to emphasize some of the capabilities of a multipass system. Both symbol streams used here, Pass1 and Pass2, are QPSK modulated. By combining them in this manner the example highlights the fact that the throughput of the two streams is equivalent to that of a single 16-QAM symbol stream. One could even use the two QPSK streams to transmit a 16-QAM stream of data. That single stream of data intended for the 16-QAM system could simply be demultiplexed into two input streams before entering the multipass system and then multiplexed at the receiver. This would result is the system communicating the exact same information using two QPSK passes as a single 16-QAM pass.

Once the two data streams have been combined into a single multipass data stream, that stream is converted to the analog domain and sent through the transmitter front-end. There it is up-converted to the carrier frequency and radiated from the antenna across the wireless channel.

5.2.2 Channel

The combined symbols propagate across the wireless channel and arrive at the receiver corrupted by noise. Here, this additional noise will be modeled as Additive White Gaussian Noise (AWGN), see Section 2.1, with a noise power spectral density of $N_0/2$, see Figure 5-7. The addition of noise to the transmitted signal results in a spreading of the possible signal space values which can be received at the receiver. Channel noise turns the precise signal space points of the symbols which were transmitted into probability clouds of possible values at the receiver, see Figure 5-8.

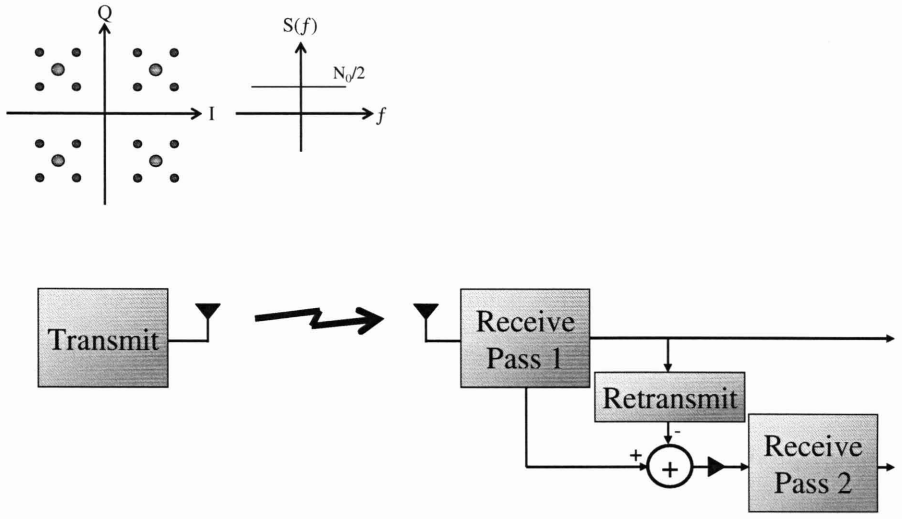


Figure 5-7: QPSK:QPSK 2-Pass Multipass System Example – Wireless Channel.

The system assumption is that the size of the noise, and hence these probability clouds, is small enough such that the Bit Error Rate (BER) of all passes at the receiver is acceptably low. If this is not the case then the channel cannot support the attempted data rate and fewer passes and/or lower modulation types must be used. Specifically, the noise power must be sufficiently small relative to the Pass2 scaled power.

5.2.3 Receiver

The waveforms which arrive at the receiver consist of the multipass data stream plus noise. The receiver front-end filters, amplifies, and down mixes this received signal and converts it to the digital domain. The digital received signal is then corrected for channel effects and copied into two streams. One of these streams is sent through the Pass1 receive chain and is processed as through a single-pass system. A second stream, meanwhile, is stored to be used as input to Pass2. This splitting into multiple streams occurs just before the demapping of symbols into bits. This enables the system to correct global errors on all passes but still preserves the information from additional passes for later retrieval.

These streams consist of symbols in signal space. The real and imaginary symbol values which, before transmission, corresponded precisely to the possible Pass2 locations in signal space, now have been fogged by noise such that their location can only be represented by a cloud of likely possible locations. The small shaded circles in Figure 5-8 represent these clouds of likely possible symbol coordinates around each potentially transmitted symbol value. For Data1 to be correctly recovered from the Pass1 stream, the shift in location of the symbol coordinates from the original Pass1 location caused by the addition of both the Pass2 symbol values and the channel noise must be less than the distance to the decision boundaries for the Pass1 symbol. For the QPSK case, the Pass1 decision corresponds to a choice of quadrant. The decision boundaries are the real and imaginary axes and are represented in the figure by the dashed straight lines. If the final received symbol falls outside of the intended quadrant in either the real or imaginary dimension, then a reception error will have occurred.

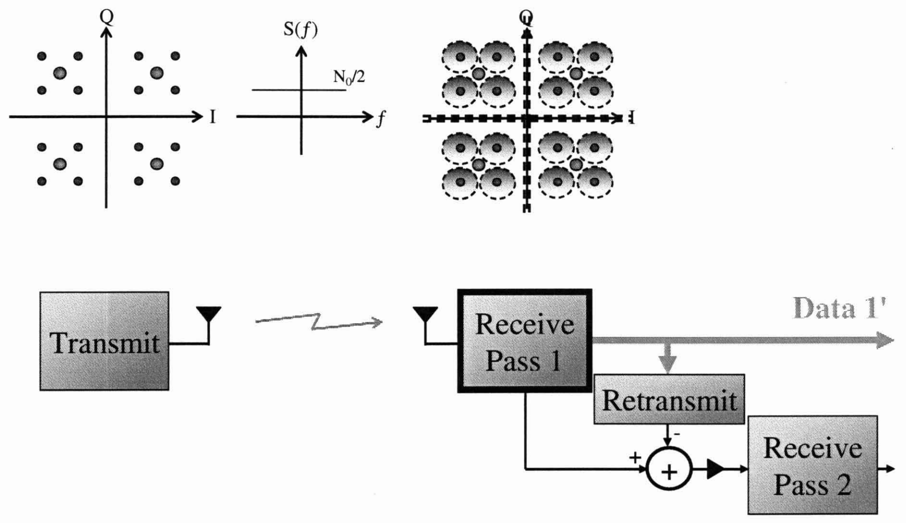


Figure 5-8: QPSK:QPSK 2-Pass Multipass System Example – Receive Phase I.

The output of the Pass1 receive chain is the Data1′ data stream. Assuming no errors, the Data1′ stream is a recreation of the Data1 stream of the transmitter and is the result of the successful transmission of this set of data across the wireless channel. This Data1 stream is copied and sent to two locations. The first of these is the output of Pass1. The second destination of the Data1 stream is in a retransmission path to assist in the decoding of Pass2.

The Data1′ stream is the receiver's most likely construction of the information which was sent by the transmitter. It is the best estimate of the receiver as to which bits were sent. If these bits were correctly communicated, then the system should be able to recreate a perfect, noise-free version of the symbol values which were originally created by the transmitter. Since the functions at the transmitter were all deterministic, the intended transmitted symbol values can be recreated simply by processing the Data1′ bits through a copy of the same functions which were originally used by the transmitter. The portion of the transmit path needed to transform input bits back into symbol values is used to recreate the Pass1 symbols, labeled Pass1′, from the Data1′ bits, see Figure 5-9.

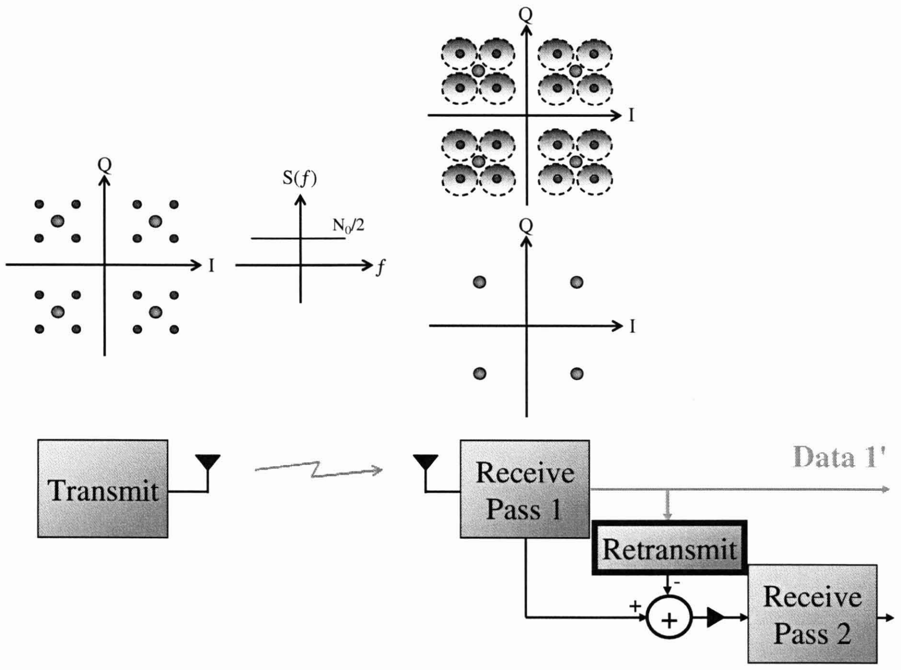


Figure 5-9: QPSK:QPSK 2-Pass Multipass System Example – Receive Phase II.

The regenerated Pass1′ symbol values can then be used to inform the reception of Pass2. The saved copy of the received multipass symbol stream can be combined with the recreated Pass1′ symbol stream to isolate Pass2 from the multipass values. The Pass1′ symbols are subtracted from the full multipass received symbols so that each multipass symbol is offset by the coordinate values of the Pass1′ symbol. This is equivalent to an origin shift of the multipass symbol in signal space, one which removes the Pass1′ symbol influence leaving only the Pass2 symbol plus noise. This resultant symbol stream can then be amplified to undo the effects of the Pass2 scaling performed at the transmitter, see Section 5.2.1. The net result is to undo the effect of the combining function. The subtraction of the Pass1′ symbols removes the effects of superposition and the amplification of this result undoes the Pass2 scaling. The consequence is a Pass2 symbol stream which resembles a single-pass received symbol stream, see Figure 5-10.

It is important to note that this amplification process amplifies the noise as well as the Pass2 signal, but the ratio between noise and Pass2 signal power stays the same. If the noise power is larger than the original scaled Pass2 power then an error will occur. If this is consistently the case, then the attempted data rate and/or the combined modulation scheme is likely beyond the capabilities of the current channel. A smaller modulation scheme with a correspondingly smaller scaling, a different combining function, or a slower data rate is needed.

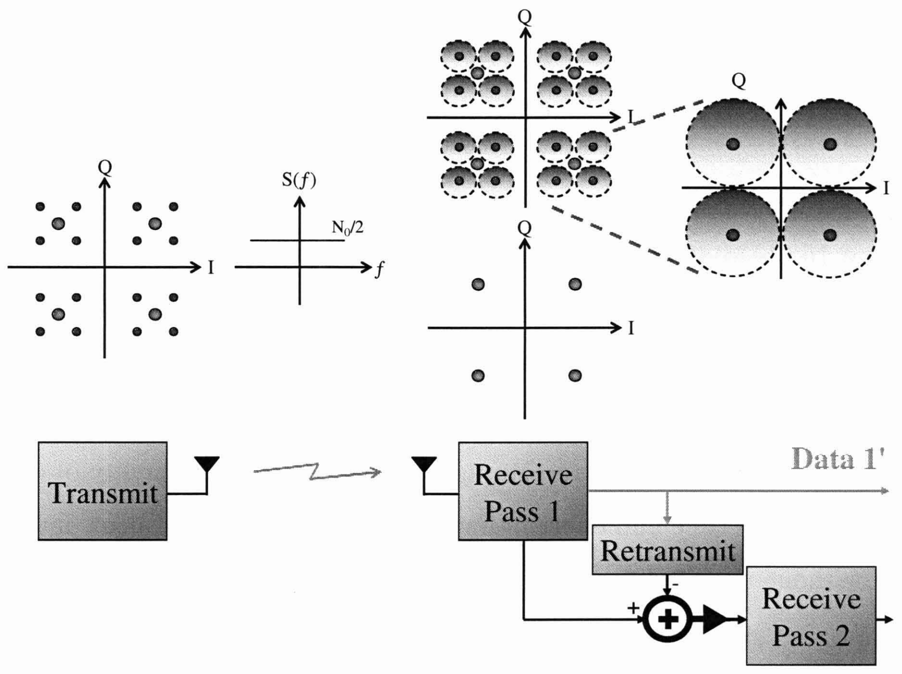


Figure 5-10: QPSK:QPSK 2-Pass Multipass System Example – Receive Phase III.

The Pass2-plus-noise-only portion of the multipass received stream is processed by the same type of receiver chain used in Pass1, see Figure 5-11. The Pass2 receive chain is a second instantiation of the same processing blocks used in Pass1, but configured for the Pass2 data rate. The output of this pass is the recovered Data2′ data stream.

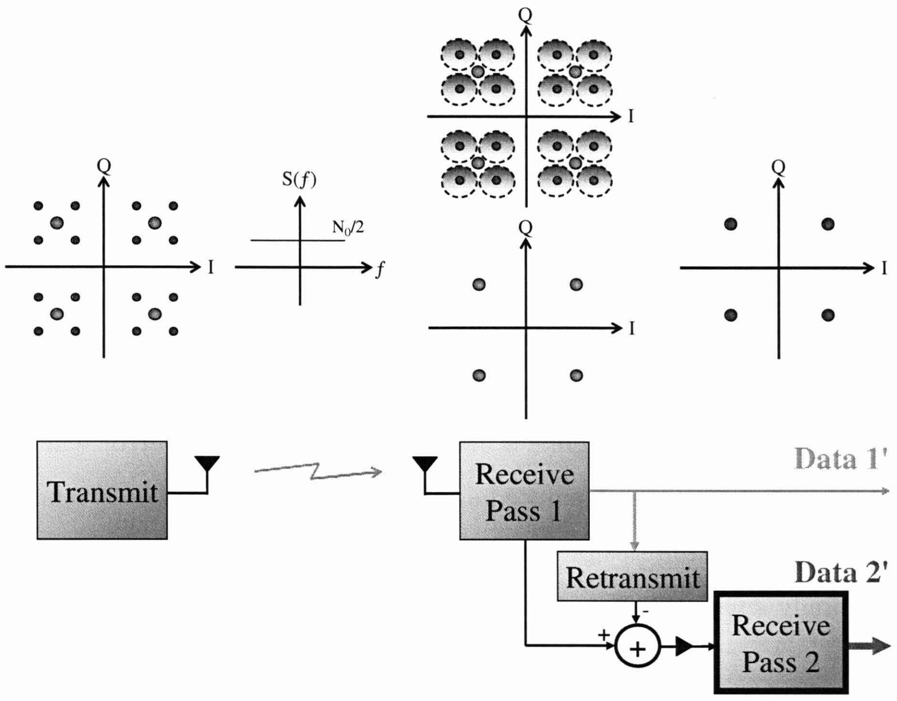


Figure 5-11: QPSK:QPSK 2-Pass Multipass System Example – Receive Phase IV.

Very little new design effort is required for the reception of two streams at QPSK instead of the single 16-QAM stream required to achieve the same data throughput. The Pass1 and Pass2 receive chains are identical and the partial retransmit chain is simply a copy of a portion of the transmitter processing chain. The additional elements are simply the necessary routing and storage and the processing to undo the combining function applied at the transmitter, in this case subtraction and scaling. This commonality of design elements translates to reduced time and effort in implementation. The equivalent data rate design is achievable without the need to carefully tweak and iterate designs.

The multipass system uses two passes to achieve the same throughput as a high data rate single pass. It requires more physical resources for the parallel functions of the passes and associated processing, but allows all of these resources to operate more slowly. This example system allows a trade-off between the amount of processing resources used and the rate at which those processing resources must operate. The system spatially pipelines the processing across more hardware resources in order to achieve the same throughput without the associated increase in clock speed and temporal reuse required by a single pass system to achieve the higher data rate.

5.3 Further Multipass Compositions

The multipass system concept can be extended beyond two passes to larger compositions. Additional passes simply translate into additional iterations of the two pass approach, see Figure 5-12. The properties of the two pass case apply equally to the large multipass case. Each pass can still operate at its own data rate and the cumulative rates combine to form a higher overall rate. The system still benefits from the design reuse of functional blocks as in the two pass case. In fact, additional design reuse is enabled as those few multipass-specific elements used in the two pass case, such as the combining function blocks (subtraction and scaling) and even the storage and routing elements, can be reused for the later pass iterations.

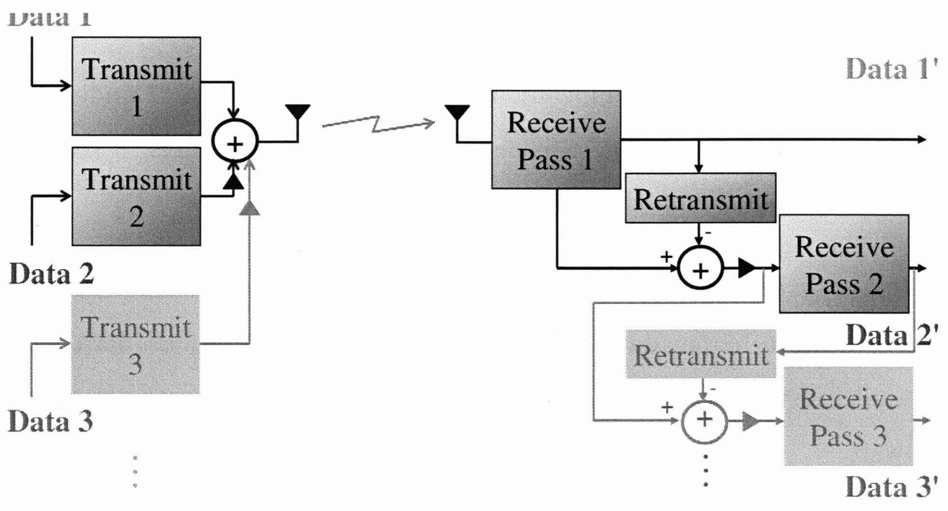


Figure 5-12: Beyond 2-Pass Multipass System.

Larger multipass systems provide the opportunity for even more flexibility of design. Passes can be dynamically added and removed, and pass data rates adjusted to provide the combinations of rates and passes which are appropriate for the given situation. Additional passes can reduce the required processing rate, or enact high data rates as such capabilities become available. Composing additional passes using the same combining function becomes just a matter of migrating processing resources from other tasks to enacting the already written receiver pass software.

Theoretically, as many passes can be added as there are processing resources available. No additional design is required; the software need simply be implemented on the newly available hardware. There are real-world limitations on the number of passes which can be used, however, as discussed in Section 5.5.

5.4 Benefits of a Multipass System

Multipass systems provide the capability of leveraging the properties of the digital realm to improve the flexibility of communication systems and enable their implementation through general-purpose processing. By using multiple parallel processing passes, multipass systems enable the following benefits:

- Multipass systems give the ability to task processors for communication based on channel, data rate, and application needs, not *a priori* hardware decisions.
  - The capability to dynamically adjust resource usage avoids under-, un-, or over-utilized resources. This translates to system-wide savings in terms of physical area, power, design time, and cost.
- Multipass systems layer multiple streams of data onto the same transmitted waveform to achieve higher data rates through the use of additional hardware resources in parallel instead of through an increased processing rate on the currently available hardware.
- Through the spatial distribution of processing loads, multipass systems enable a corresponding reduction in processing rate requirements. This is achieved by limiting the impact of hardware processing bottlenecks through the spatial pipelining of functions.
  - Multipass systems make feasible the processing of higher communication data rates than previously achievable on GPP/DSPs, while maintaining the flexibility of general-purpose processing.
  - Lower processing rates also enable slower clock rates which can translate into lower power consumption (see Section 5.4.1).

- Multipass systems allow a trade-off between achievable data rate per pass with the number of passes and processing per pass. The system provides the ability to explicitly balance and dynamically modify the system design between communication and computation.

In addition, multipass systems also allow the system structure to mirror the characteristics of the data being communicated, for instance, the prioritization of data through the combining function used. For the superposition combining function, the earlier passes, with larger power, can be used to convey more critical data, while the data in later passes can convey increasingly lower priority information. Multipass provides a means of potentially transmitting the lower priority information while helping to ensure that the higher priority data arrives through the choice of the scaling values used. The use of scaling values to prioritize data is discussed in Section 8.2.1.

5.4.1 Reduced Power Consumption

The use of the multipass algorithm allows a trade-off between the area of the system and a lower required system clock rate. A lower system clock rate can be used to reduce the power consumption of the system. This can be done by addressing two of the main sources of power dissipation in digital CMOS circuits: dynamic switching power and static leakage power. The dynamic switching power can be reduced through a lowering of the supply voltage, V_{dd}. The static leakage power can be reduced by using devices with higher threshold voltage, V_t, in order to minimize subthreshold leakage current.

The alpha power-law MOS model provides a means of relating the frequency of operation to these values [83, 79, 80]

$$f \propto \frac{(V_{dd} - V_t)^{\alpha}}{V_{dd}} \tag{5.1}$$

where α models short channel effects and is typically 1.3 for modern devices [84].

For a multipass system using 2 passes the frequency of operation could be scaled down by a factor of 2 while doubling the area and resources used. The dynamic power dissipation is given by

$$P_{dynamic} = C_{switched} V_{dd}^2 f \tag{5.2}$$

where $C_{switched}$ is the total effective switched capacitance. The doubling of area impacts dynamic power consumption by increasing the $C_{switched}$. This increase in dynamic power is linear, however, while the decrease due to lowering V_{dd} is quadratic, suggesting a net reduction in power consumed.

The transistor subthreshold leakage power dissipation is given by

$$P_{leakage} = V_{dd}(I_0 10^{(V_{gs}-V_t)/S}) \tag{5.3}$$

where I_0 is the drain current with $V_{gs} = V_t$ and S is the subthreshold slope (typically in the range of 60–90 mV/decade) [79, 80, 81]. The effect of doubling the area to reduce clock speed corresponds to a doubling of transistors in the off state and a corresponding doubling of leakage power dissipation. The exponential relationship of V_t to leakage power, however, suggests that an increase in V_t will also result in a net reduction in power dissipation.

Holding V_t constant at the nominal value and reducing V_{dd} to match the new operating frequency of operation of $f/2$ results in a new multipass supply voltage of $V_{dd}^{mp} = .499$. Comparing the ratio of dynamic power dissipated between the V_{dd}^{nom} and V_{dd}^{mp} supply voltages results in at least a 7× reduction in dynamic power dissipation for the lower frequency multipass design over a single pass design.

Similarly, holding V_{dd} constant and solving for the V_t value which corresponds to the lower operating frequency gives $V_t^{mp} = .613$. Choosing a representative subthreshold slope of 70 nm/decade the ratio between the static leakage power dissipation of the nominal, V_t^{nom}, and multipass, V_t^{mp}, threshold voltages can be determined. This results in at least a 397× reduction in static leakage power.

This example represents the extremes focusing on the reduction of either static or dynamic power dissipation alone. Depending on the ratio of static to dynamic power dissipation some combination of reducing V_{dd} and raising V_t is optimal. It is also important to consider that while the option to change the threshold voltage may not be available, the prevalence of processor voltage scaling makes adjusting the supply voltage a readily available option.

5.5 Limitations on Multipass Systems

The number of passes in a multipass system is limited by the channel characteristics and the noise associated with the system components. The factors limiting the number of passes are combining function dependent and could be the result of front-end system components (such as amplifier saturation or non-linearity or the dynamic range of the DAC), system-wide components or effects (such as oscillator jitter or self-mixing) or channel characteristics (such as interference or inter-symbol interference). For the superposition combining function, the noise floor of the system (including the output referred transmitter noise, the input referred receiver noise, and channel noise sources) inherently limits the number of passes available.

The noise floor provides a limit on the number of useful passes. Even in the absence of channel effects there is a maximum number of passes. The transmitter noise minimum is fixed and amplified along with the signal. This already limits the maximum possible signal to noise ratio and thus the minimum noise power. The receiver noise minimum is similarly fixed. Increasing the received signal power can minimize the effect of this receiver noise, but there are limitations on the maximum useful signal power. A too large received signal power can saturate the receiver's lower noise amplifier (LNA) or surpass the dynamic range of the DAC. Thus a maximum possible signal to noise ratio is set at the receiver as well. This maximum SNR translates into a minimum noise floor which sets a definite limit on the number of passes which can be successfully operated, even in the perfect channel case.

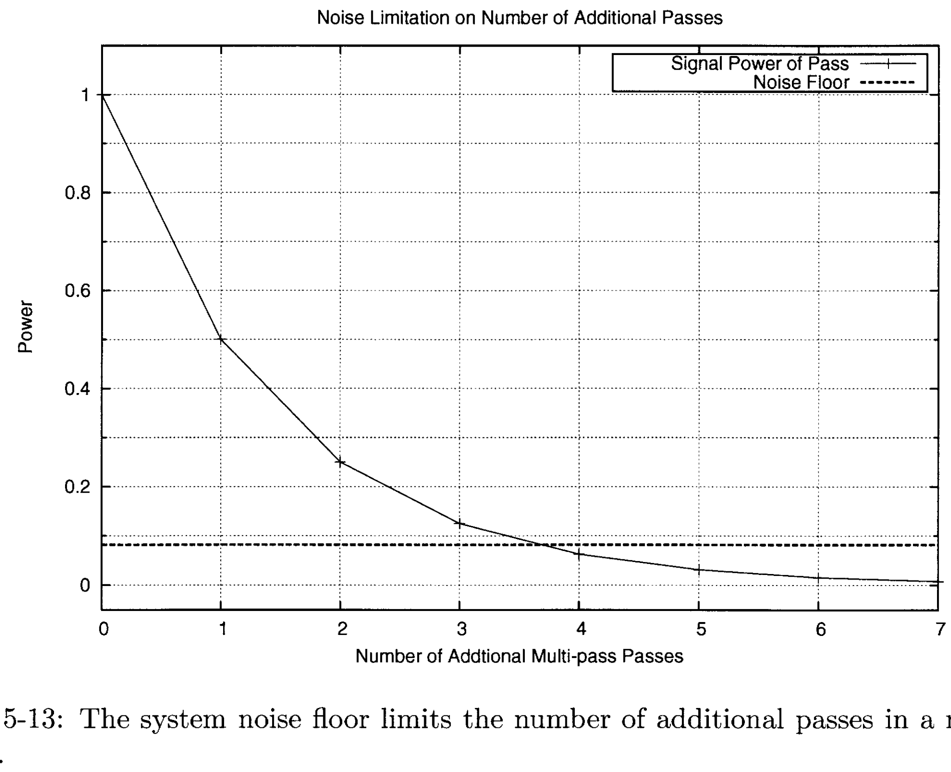


Figure 5-13: The system noise floor limits the number of additional passes in a multipass system.

Each new pass in a superposition combining multipass system must be scaled down to provide disambiguation from the previous passes. The attenuation of each subsequent pass is cumulative, however, and the signal power for each additional pass falls rapidly while the system noise-floor remains constant, see Figure 5-13. From an equivalent perspective the noise grows exponentially with each processing pass for the superposition combining function. The use of superposition results in the signal power decreasing as $(\frac{1}{S})^N$, where S is the scaling used for each pass, while the minimum noise remains constant. Another combining function might result in a different relationship between the noise and the number of passes, but there will always be a limitation on the number of passes which can be sustained.

Chapter 6

The Raw Processor – A Tiled Architecture

The Raw processor is a research architecture design undertaken by the MIT Raw research group [85, 86].

> Fast moving VLSI technology will soon offer billions of transistors, massive chip-level wire bandwidth for local interconnect, and a modestly larger number of pins. However, there is growing evidence that wire delays become relatively more significant with shrinking feature sizes and clock speeds. Processors need to convert the abundant chip-level resources into application performance, while mitigating the negative effects of wire delays.
>
> ...
>
> The Raw project addresses the challenge of whether a future general-purpose microprocessor architecture could be built that runs a greater subset of ... ASIC applications while still running the same existing ILP-based sequential applications with reasonable performance in the face of increasing wire delays. [74]

The Raw processor is an example of a tiled architecture. It consists of multiple uniformly replicated tiles, each of which contains a MIPS-style processing pipeline. These tiles are general purpose in nature and each is able to run its own independent instruction stream. The tiles are connected in a 2-D mesh arrangement by multiple interconnection networks. All interconnections are nearest neighbor in order to keep their lengths short. Tight integration of the the interconnects with the tiled processors allows low latency communication between tiles. This high speed interconnection network extends off-chip to enable efficient usage of pin resources and, consequently, large amounts of fast I/O.

6.1 Design Philosophy

The stated purpose of the Raw project is given below:

> Rapid advances in technology force a quest for computer architectures that exploit the new opportunities. Current architectures, such as hardware scheduled superscalars, are already hitting performance and complexity limits and cannot be scaled indefinitely. The Raw Architecture Workstation (Raw) is a simple, wire-efficient architecture that scales with increasing VLSI gate densities. The Raw architecture's goal is to provide performance that is comparable to that provided by scaling an existing architecture, but that can achieve orders of magnitude more performance for applications in which the compiler can discover and statically schedule fine-grain parallelism.
>
> The Raw project's approach to achieving these goals is to implement a simple, highly parallel VLSI architecture, and to fully expose the low-level details of the hardware architecture to the compiler so that the compiler or the software can determine, and implement, the best allocation of resources for each application. Eliminating a fixed instruction-set interface between the compiler and the hardware, Raw is composed of a set of interconnected tiles, each tile comprising, instruction, switch-instruction, and data memory, an ALU, FPU, registers, and a programmable switch.
>
> Our approach leverages the same set of features that make application-specific custom hardware systems popular for specific applications. First, Raw implements fine-grain communication between large numbers of replicated processing elements and, thereby, is able to exploit huge amounts of fine-grain parallelism in applications, when this parallelism exists. Second, it exposes the complete details of the underlying hardware architecture to the software system (be it the software CAD system, the applications software, or the compiler), so the software can carefully orchestrate the execution of the application by applying techniques such as pipelining, synchronization and conflict elimination for shared resources by static scheduling and routing. [86]

6.1.1 Scalability

The trend toward reduced transistor sizing has resulted in smaller integrated structures and a larger portion of the die area available for other use. An open question is how to take advantage of this newly available die area. Adding more transistors to the processing core dramatically increases the processor complexity and makes design and verification prohibitive. As a result, many designs have held the core constant across technology generations and simply used the extra die are for larger cache structures. While this does provide some benefit, there is a limit to the performance increase achievable through increases in memory structures alone.

In addition to usage concerns, the loss of the wire abstraction due to the increased frequency of operation is another primary concern. One of the main reasons for transistor shrinkage is the quest for higher performance through frequency scaling. Smaller transistors with lower threshold voltages and smaller capacitances may be switched more quickly, thus allowing a higher clock frequency. These higher frequencies, however, break the model of the wire as an instantaneous connection mechanism. As frequencies increase, the delay associated with long global on-chip wires becomes significant and signals can no longer traverse long wires in a single clock cycle [87]. As transistors get smaller and frequencies increase, the percentage of the die that can be covered in a single clock cycle decreases [88].

What is needed is a *scalable* architecture that would avoid these negative effects of transistor and frequency scaling. A scalable architecture is built of structures that can continue to operate efficiently in such an environment. A scalable architecture has the ability to arbitrarily adjust the size and number of structures without changing design. Each structure, or processing core, and the corresponding interconnect between those cores is designed in a manner relative to the core's size, instead of in an absolute sense for the specific die size and technology. Cores are designed to occupy and interconnect with each other in a fraction of the distance a data value can travel in a single clock cycle. So whether such a core covers $1/4$ of the die, with appropriate clocking frequency, or $1/100,000$ of the die, it will operate exactly the same. Therefore, as the transistor technology scales, the clock frequency will increase and the distance a signal can travel will decrease, but this is offset by the decreased size of the transistors and length of the wires needed to connect them.

6.1.2 Designability

Scalable processors are often built through the replication of smaller scalable cores. Such an approach has multiple benefits. The first of these is, obviously, scalability. If the processing elements are individually scalable, then their composition is scalable. All the available die area can then be used by composing together as many of the small scalable cores as will fit. Using this approach, the die can be thought of as a mosaic, with the processing cores laid down as tiles filling the space. Long wires are avoided because each tiled core is self-contained except along well defined boundary interfaces and is thus scalable.

Another important benefit of scalable architectures is significantly reduced design effort.

As processors grow in number of transistors and amount of complexity, the cost of design and verification, and the likelihood of errors creeping into the system, becomes unacceptably high. By replicating tiled processing cores, a scalable system can reduce the workload and cost significantly. Design effort can focus on a single small and relatively simple scalable tile which can be built and verified and then reused. This reduces the design time and effort required, while still effectively utilizing the entire die and achieving impressive processing capabilities. The potential future design efforts are also reduced, as a later scaling in technology would not necessarily require a redesign. Since the tiles themselves are scalable, a technology scaling could simply result in more tiles per die with no further design effort needed.

6.1.3 Exposed Communication

The Raw approach treats communication as a first-class design element. The Raw processor provides large amounts of fast interconnect between tiles which can be explicitly managed by the software. This high speed interconnect communicates directly into each tile processors pipeline bypass network for efficiency and speed. The implementation of a scalar operand network [89] allows significant fine-grained parallelism to be achieved by providing extremely low-latency inter-tile communication.

The communication in the Raw processor is exposed in the programming model. This allows the interconnect between tiles to be explicitly managed and provides the type of low-level orchestration of data necessary to achieve efficient fine-grained parallelism. The careful scheduling required for the inter-tile data traffic is an example of static communication (communication which can be predicted at compile time) and therefore can be automatically managed by a compiler for ease of implementation.

6.1.4 Streaming I/O

The I/O model of the Raw approach treats pins as simply an extension of the on-chip high bandwidth networks. This provides the processor access to a significant amount of I/O that feeds directly into the on-chip networks. This type of setup is perfectly suited to a streaming data model because data can move on-chip quickly and in large quantities. Incoming data can be retrieved from the network, processed, and sent back out without the costly intermediate step of storing it in memory.

6.2 Communication Networks

The Raw processor has four 32-bit full-duplex on-chip mesh networks, consisting of over 12,500 wires. There are two classes of network available, static and dynamic. The two static networks are explicitly managed and statically scheduled while the two dynamic networks are dynamically routed at run-time.

6.2.1 Low-latency, High-speed Interconnect

The two static networks of the Raw processor are designed for regular communication that is known and can be statically scheduled at compile time. The programmer or compiler directly programs the switch processor, the router which controls the flow of data, on the static networks:

> The Raw ISA exposes these on-chip networks to the software, enabling the programmer or compiler to directly program the wiring resources of the processor and to carefully orchestrate the transfer of data values between the computational portions of the tiles - much like the routing in a full-custom application specific integrated circuit (ASIC). Effectively, the wire delay manifests itself to the user as network hops. It takes six hops for a data value to travel from corner to corner of the processor, corresponding to approximately six cycles of wire delay. [4]

Each static network provides a 32-bit full-duplex network link between each tile processor, its nearest neighbors, and the other network. These networks extend into the processor pipeline itself and are not only register-mapped, but also integrated into the bypass network of each tile's processing pipeline [89]. This allows high speed communication on the static network between tiles with an ALU-to-ALU latency of 3 cycles.

6.2.2 Dynamic Networks

The two dynamic networks are used for communication for which the exact routing is indeterminable until run-time. The dynamic networks are used to support cache misses, interrupts, dynamic messages, and other asynchronous events. The two dynamic networks have different functions: the memory dynamic network (MDN) is used to route memory traffic between the Raw tiles and the off-chip main memory and the general dynamic network (GDN) is available for application or user use. The MDN is precluded, by policy, from deadlocking. The acceptable communication patterns for the MDN avoid deadlock and are already enforced for hardware generated memory traffic. User access to the MDN is discouraged as is it intended for use only by trusted clients such as data caches, DMA, and I/O. The GDN, on the other hand, is freely available for use by the user or program. It is the programmer's responsibility to avoid deadlock on this network; it is not guaranteed not to deadlock. The GDN uses a watchdog timer to detect potential deadlock situations and to initiate deadlock recovery procedures.

The dynamic networks are dimension-ordered wormhole routed packet networks. Each dynamic network packet consists of a header word followed by message words. Since the network is wormhole routed, the header word configures the necessary dynamic network path that the data words will follow. This path is dimension-order routed; it follows the policy of scheduling all vertical routes before any horizontal routes. Once the header has established a route in a dynamic network node, that routing stays in place until the rest of the message words have been delivered. No other dynamic message can set up a route which interrupts a message already in progress.

6.2.3 Switch Processor

The router for the static networks is called the switch processor. It is a route sequencer processor which is flow controlled and operates independently from the tile processor. All communication between the switch and tile processors occurs through the register mapping of the static network into the tile processor register file. The switch processor is designed as follows:

> The 5-stage static router controls two routing crossbars and thus two physical networks. Each crossbar routes values between seven entities: the static router pipeline, north, east, south, west, the compute processor, and the other crossbar. The static router fetches 64b instruction words from an 8k-entry cache. Each word simultaneously encodes a small command (branch and decrement, local register file accesses), and 13 routes, one for each crossbar output. For each operand sent between tiles on the static network, there is a corresponding instruction in the instruction cache of each router through which the word will travel. These instructions are programmed by the compiler. Thus, the static routers collectively reconfigure the entire communication pattern of the network on a cycle-by-cycle basis, and enable Raw to handle both scalar and streaming data types.

> The static router is flow-controlled, and does not proceed to the next instruction until all of the routes in the current instruction have completed. This ensures that destination tiles receive incoming words in a known order, even when tiles suffer unpredictable delays from cache misses, interrupts, or branch mispredictions. The static router provides single-cycle-per-hop latencies and can route two values in each direction per cycle. Because Raw's network is point-to-point, and because operands are routed only to those tiles that need them, the Raw design decimates the bandwidth required for operand transport relative to a comparable broadcast-based superscalar. [90]

6.3 Raw Implementation

The Raw processor was implemented in the IBM 7SF SA-27E copper process. This process is a 180nm 1.8V 6-layer CMOS process. The processor has 16 tiles, each of which consists of a single-issue in-order 8-stage MIPS-style processing pipeline, a 4-stage single precision pipelined FPU, a 32KB data cache, two communication routers (one for the static network, one for the dynamic), and 96KB of instruction caches. The Raw processor has a total transistor count of 122 million transistors.

The choice of 16 tiles was determined by the die size available. The die area is 18.2mm × 18.2mm, although the tiles take up only 16mm × 16mm of this area. The larger die size was necessary to accommodate the column grid array (CGA) package in order to achieve a higher pin count. The CGA package has 1657 total pins of which 1080 pins are available for use as high speed transceiver logic (HTSL) I/O pins. A die photo of the entire 16 tile Raw processor can be seen in Figure 6-1.

6.3.1 BTL Simulator

The BTL Simulator is a cycle accurate Raw simulator. It has been verified extensively against the hardware:

> We verified that the simulator and gate-level RTL netlist have *exactly* the same timing and data values for all 200,000 lines of our hand-written assembly test suite, as well as for a number of C applications and randomly generated tests. Every stall signal, register file write, SRAM write, on-chip network wire, cache state machine transition, interrupt signal, and chip signal pin matches in value on every cycle between the two. This gate-level RTL netlist was then shipped to IBM for manufacturing. Upon receipt of the chip, we compared a subset of the test on the actual hardware to verify that the chip was manufactured according to spec. [74]

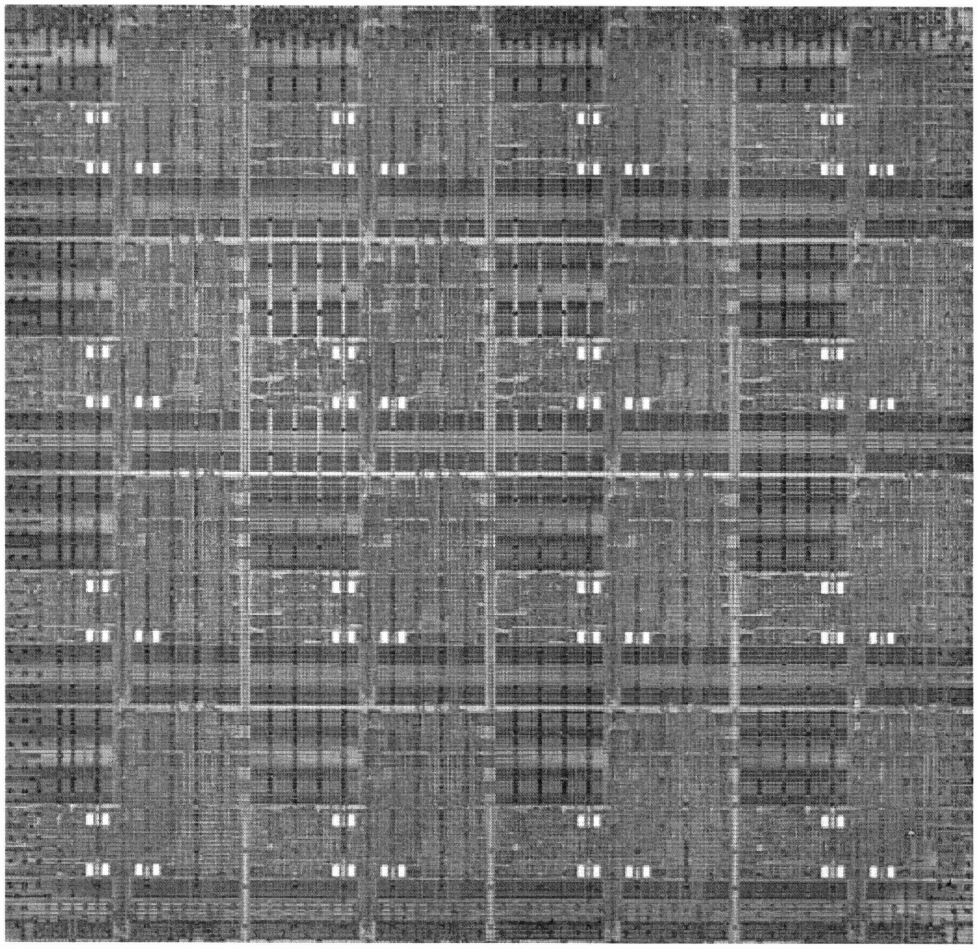


Figure 6-1: Raw die photo.

6.3.2 Technology Scaling

The Raw processor is a research prototype. The Raw design was not aggressively optimized and included numerous research-related features which would not be considered for a commercial processor. The Raw design was built in an ASIC standard cell process with no custom logic at all. The IBM process used, at 180nm, is now somewhat outdated as there are multiple subsequent process generations which could be used for a current optimized design. This implies that there are large amounts of performance, area, and power savings to be had if the design were reimplemented with fully customized logic or in a more recent process technology.

A commercial Raw design would likely be implemented using full custom logic in a more aggressive process technology. The estimated effects of a change in design style or process can be seen in Table 6.1.

Speculative Processor Technology Scaling

Processor	*Process (nm)*	*Frequency (GHz)*	*Area (mm^2)*	*# Tiles/Cores*
PowerPC970	130	1.8	66	1
PowerPC970FX	90	2.5	121	1
Standard Cell[a] Raw	180	.425	331	16
Standard Cell Raw	130	.558	331	36
Standard Cell Raw	90	.850	331	81
Full Custom Raw	180	1.28	331	36
Full Custom Raw	130	1.77	331	64
Full Custom Raw	90	2.55	331	144
Full Custom 2[b] Raw	180	1.28	331	49
Full Custom 2 Raw	130	1.77	331	100
Full Custom 2 Raw	90	2.55	331	225

[a]IBM SA-27E ASIC Process
[b]Full Custom Raw with 1/2 the SRAM

Table 6.1: Speculative effects of future technology scaling on Raw processors.

The PowerPC970(FX) was used as a reference because the change from the 970 to the 970FX was purely a process shrink. The effects on frequency of technology scaling were assumed to be linear per dimension for logic gates. The SRAM structures in the IBM SA-27E process are specially implemented and optimized and, as such, are assumed to carry over from process to process without shrinking. The effects of moving to full custom design from standard cell for both frequency and area were applied as suggested in [91].

For research purposes, the various memories on the Raw prototype were sized to be quite large. A full custom commercial design would be likely to use on the order of half the amount of SRAM implemented in the prototype. The "Full Custom 2" Raw implementation reflects this kind of commercial design. It is a full custom version of the Raw design which contains half the amount of SRAM.

Reimplementing a commercial full custom version of the Raw processor in a current, but not cutting-edge, technology (90nm) would allow 144 Raw tiles per die and an increased in clock speed to over 2.5GHz. If the amount of SRAM were halved, the number of tiles available would jump to 225. In future process technologies, these gains would be much larger, thus supporting the design philosophy of treating tiles as an abundant resource.

Chapter 7

Multipass Implementation System

An 802.11a based multipass system was implemented utilizing the Raw tiled architecture. The implementation system provides a proof-of-concept for multipass systems and acts as a platform for the exploration of communication systems on tiled architectures. This system evolved through multiple iterations. An iterative approach was used to verify correctness and congruency between implementation platforms, to improve simulation efficiency, and finally, to achieve real-time compatibility. The implementation iterations included software implementations in C for an x86 platform; in C targeting the Raw platform in both simulation and on the Raw hardware itself; and in a mixture of C and Raw Assembly languages again targeting both hardware and simulation.

The initial iteration was solely a single pass 802.11a implementation which was used to gain experience with 802.11a systems and with the use of Raw as a platform for communication systems. The single pass system provided a well-defined implementation goal, while allowing exploration of the implications and intricacies of the system design, the design of each building block, as well as the potential for spatially distributing the system. From this single pass system targeting a single Raw tile, the next iteration targeted spatially pipelining the single pass system across multiple tiles. The next steps involved optimizing the bottlenecked elements of this distributed single pass system to make even the lowest data rates compatible with the general-purpose processing operation of Raw. Finally, the elements of this distributed single pass system were reused, along with the necessary routing and retransmission blocks, to build a 2-pass multipass system targeting the Raw processor.

The evolutionary system design strategy highlights the design reuse aspect of multipass systems. The incremental effort required to create a multipass system from a single pass system was small. The majority of the design effort went into building the single pass system and optimizing it in Assembly language. The multipass system implementation, for the superposition combining function, required only a small additional effort.

7.1 Selection of 802.11a Baseband

To explore communication on tiled processors, the selection of a relevant and sufficiently ambitious wireless specification was needed. The choice of a standard specification, as opposed to a generic wireless system mockup, was deemed vital. The use of a real-world system specification provided a means of showcasing as realistic a system as possible, while highlighting the relevance of multipass and tiled architecture-based communications to current wireless regimes. As important to the decision to use a currently wireless specification was the ability to provide sufficient verification of such a system. A mature and readily available specification with accessible example implementations provided a means of grounding the simulation results in reality and of confirming the correctness of system operation. The growing relevance, relatively high data rates, mature specification, and amenable system structure of Wi-Fi made it a logical choice.

Wi-Fi is a wireless local area network (WLAN) specification which has attained general adoption and usage. Wi-Fi is a marketing term used to describe three IEEE 802.11 wireless standards: 802.11a [24], 802.11b [92], and 802.11g [93]. 802.11b represents the original Wi-Fi specification which operates in the 2.4GHz ISM band and provides for data rates from 1Mbps to 11Mbps. 802.11a was then instituted as a higher data rate WLAN option with operation in the 5GHz band and data rates from 6 to 54Mbps. Later, 802.11g was proposed as a combination of the previous two specifications. It was an incremental step beyond 802.11b which allowed data rates similar to 802.11a while operating in the 802.11b frequency band of 2.4GHz and providing compatibility with 802.11b. 802.11g makes use of the 802.11a baseband to achieve the data rate capabilities of 802.11a, while retaining the ability to fall back to 802.11b baseband processing for compatibility.

The 802.11a baseband (also used by 802.11g) provides the capability for data rates from 6 to 54Mbps. This large range of possible data rates is well suited to a multipass approach. While the low data rates may be possible on certain DSP's or DSP/ASIC combinations, the high data rate of 54Mbps is not currently implementable in a general-purpose processing environment.

An additional benefit of 802.11a are the modulation schemes used to achieve the data rates. The 802.11a specification requires the use of various levels of modulation from binary phase shift keying (BPSK) to 64-point quadrature amplitude modulation (64-QAM) to achieve the different data rates. The lower level modulation schemes used, BPSK and quadrature phase shift keying (QPSK), can almost be thought of as lower QAM constellations. In fact, QPSK is often called 4-QAM and the constellations of the two modulation schemes are identical. BPSK can be thought of as a single dimension version of 4-QAM. This is particularly appealing for the implementation of a superposition combining function multipass system, as the symbols of the various passes can be composed in such a way as to resemble larger QAM modulations. Thus, the multipass system can be designed as a 2-pass system which uses QPSK for both passes and has a resultant signal space constellation identical to 16-QAM, see Section 5.2. Other combinations of pass modulations result in similar equivalencies, see Table 7.1. Such equivalent data throughputs are useful as a comparison for evaluation of the performance of multipass systems, as will be discussed in Chapter 8.

Data Rate Equivalencies			
Pass 1 Modulation Type	*Pass 2 Modulation Type*	*Raw Data Throughput*	*Modulation Equivalent*
QPSK	–	12Mbps	QPSK
QPSK	BPSK	18Mbps	– (8-QAM)
QPSK	QPSK	24Mbps	16-QAM
QPSK	16-QAM	36Mbps	64-QAM
QPSK	64-QAM	48Mbps	256-QAM
16-QAM	–	24Mbps	16-QAM
16-QAM	BPSK	30Mbps	– (32-QAM)
16-QAM	QPSK	36Mbps	16-QAM
16-QAM	16-QAM	48Mbps	256-QAM
64-QAM	–	36Mbps	64-QAM
64-QAM	BPSK	42Mbps	– (128-QAM)
64-QAM	QPSK	48Mbps	256-QAM

Table 7.1: Data rate equivalencies of multipass (2-pass) 802.11a modulation types.

Another interesting aspect of the 802.11a specification is its use of Orthogonal Frequency Division Multiplexing (OFDM). OFDM is described in more detail in Section 7.2.1, but it provides the ability to directly shape the frequency domain representation of the transmitted signal. This provides an intriguing opportunity to use a combining function which directly takes advantage of this capability, especially in the case of a non-flat channel response. Such combining functions will be touched on briefly in Chapter 11.

The system presented here operates as an 802.11a system, but at the 802.11g frequency of operation of 2.4GHz. This was necessitated by the RF circuity which was available for use, see Section 7.3.1. Since baseband processing is post down-conversion, it is irrespective of carrier frequency and, as such, this difference is not relevant to the baseband operation of the systems discussed here.

7.2 802.11a Specification Overview

802.11a is a high speed wireless LAN protocol operating in the unlicensed ISM 5GHz bandwidth. It provides for data rates from 6 to 54Mbps with a channel spacing of 20MHz. In order to achieve these relatively high data rates within the given channel bandwidth, 802.11a employs an OFDM scheme, see Section 7.2.1, to minimize ISI. Some additional aspects of 802.11a are the use of convolutional coding for channel encoding, training symbols for channel estimation and detection, and pilot symbols for phase detection. A block diagram of an 802.11a receiver along with an explanation of the various blocks can be found in Section 7.2.2. A quick listing of parameters for the 802.11a specification (data rates, timings, etc.) has been included in Appendix A for reference.

7.2.1 Orthogonal Frequency Division Multiplexing (OFDM)

Othogonal Frequency Division Multiplexing (OFDM) [94, 36, 30, 95, 32] is a means by which to avoid Inter-Symbol Interference (ISI) at high data rates. The essence of OFDM is to send multiple symbols of data in parallel over the same channel at once. Each data symbol is allocated a small portion of the overall channel width and these data subcarriers are then combined to create an OFDM symbol which occupies the full channel and consists of multiple parallel data symbols.

The benefit of OFDM is that, for a given data rate, the rate at which OFDM symbols must be sent is a fraction of that of a non-OFDM system since each OFDM symbol encompasses multiple data symbols. This reduced OFDM symbol rate helps to reduce ISI by allowing more time per OFDM symbol for symbol transitions.

As with many other systems, an OFDM system performs modulation by mapping a stream of data bits into data symbols. The mapper divides the bits into groups and assigns a complex value (which represents a sinusoid of a specific amplitude and phase) to each group based on the bit values contained within, see Section 2.2.

The OFDM symbols are explicitly constructed from the data symbols in the frequency domain, see Figure 7-1. The stream of data symbols is demultiplexed into as many parallel data symbols as there are OFDM data subcarriers. Each data symbol is then allocated to an OFDM data subcarrier by assigning it to the corresponding frequency location. The process of frequency assignment is achieved simply by feeding each data symbol into the input for the appropriate frequency bin of an Inverse Fast Fourier Transform (IFFT). The IFFT transforms the constructed frequency domain symbol into the time domain for transmission across the channel. At the receiver, the time domain symbol is transformed by an FFT back into the frequency domain so that the subcarrier data symbols can then be separated and multiplexed back into a data stream.

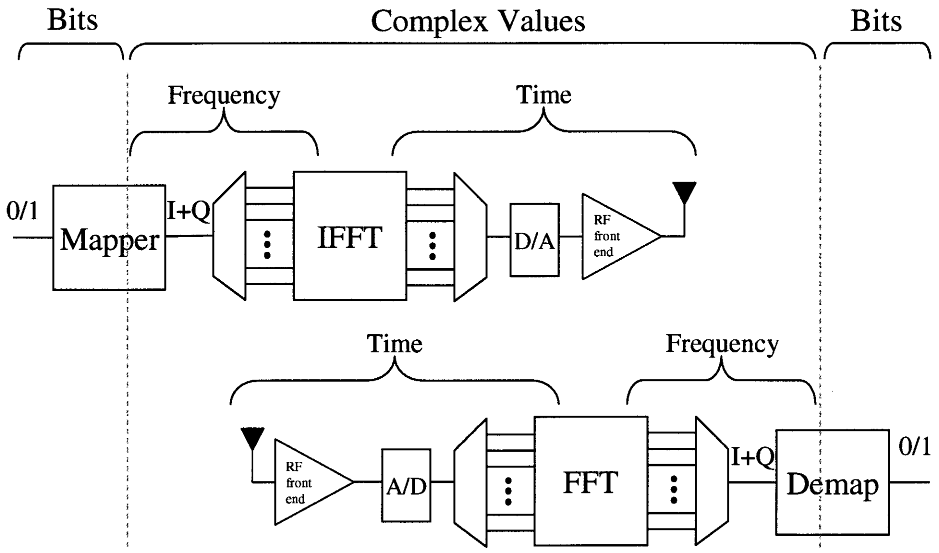


Figure 7-1: OFDM Overview.

7.2.2 802.11a Block Diagram

A block diagram of the implemented 802.11a receiver is shown in Figure 7-2. Descriptions of the individual blocks appear below.

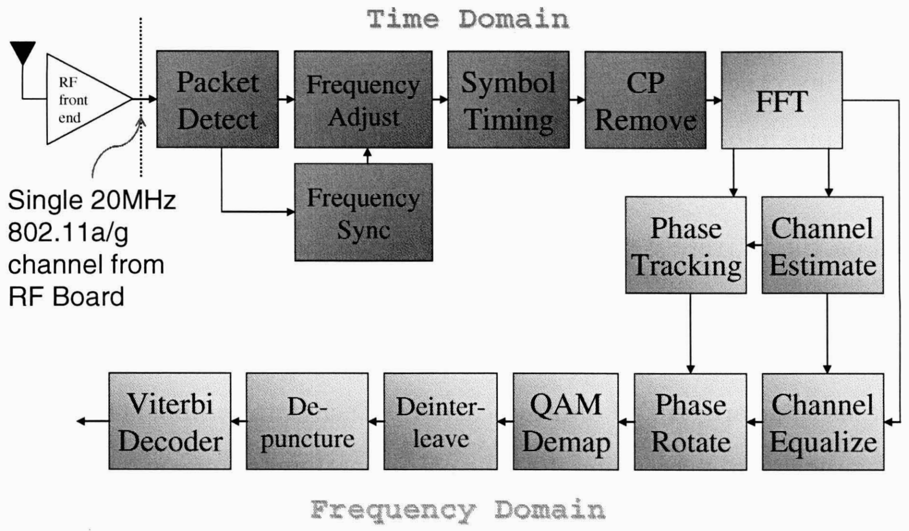


Figure 7-2: 802.11a receiver block diagram.

Packet Detect Detects the packet header which denotes the start of a data packet. The packet detection block correlates the incoming stream of values from the front-end against the known packet start OFDM training symbols. If the correlation power rises above a threshold, then a packet is assumed to have begun.

Frequency Synchronization & Adjust Uses the OFDM training symbols to detect systemic frequency offsets between the received signal and the receiver. Frequency offsets are mainly the result of slight frequency differences in the oscillators of the transmitter and receiver. Once a packet has been detected, the received packet header OFDM training symbols are compared with the known OFDM training symbol values. The data stream is then multiplied by an exponential to adjust its frequency accordingly.

Symbol Timing Detects the end of the OFDM training symbols and the beginning of the OFDM header and data symbols in the packet. Correlation with the known OFDM training symbols is used to find this demarkation point. The OFDM symbol timing value is used as a relative reference for all of the subsequent OFDM symbols in the packet.

Cyclic Prefix (CP) Remove Discards the Cyclic Prefix values. The cyclic prefix is a repetition of 16 data values in the OFDM symbol which provide buffering. The cyclic prefix provides extra data values to compensate for the possibility of the first few data values being lost due to the symbol timing being slightly off or to corruption by multipath ISI.

Fast Fourier Transform (FFT) Transforms the Time Domain OFDM symbols back into the Frequency Domain. Performs the Fast Fourier Transform function, see Section 7.4.5.

Channel Estimation & Equalization Estimates the channel system function and then attempts to compensate for it through equalization. The channel estimator uses the difference between the known transmitted OFDM training symbols and the received OFDM training symbols to infer channel characteristics. The channel equalizer then uses this channel estimate to try and remove the channel effects from the symbol values.

Phase Tracking & Rotate Tracks and fixes the shift in phase of the symbol values using known pilot symbols. Phase errors can be detected by comparing the received values of the pilot symbols embedded in each OFDM symbol to the known transmitted pilot symbol values. Phase error is usually the result of mismatches in the timings between oscillators at the transmitter and the receiver. While the frequency synchronization block addresses this in an overall packet sense, residual error remains on a symbol-by-symbol basis. These slight frequency offsets appear as phase drift at the receiver. A rotation is applied to the received symbol values to compensate for the detected phase offset.

Quadrature Amplitude Modulation (QAM) Demap Performs the demapping from complex values in symbol space back into bits. QAM is used in a general sense and could apply to QPSK or BPSK here. The demapper compares the corrected complex data symbol values against decision boundaries to determine each data symbol's corresponding Veronoi region. The demapper then outputs the bits associated with that region.

Deinterleave Undoes the interleaving of bits performed at the transmitter. The deinterleaver stores the demapped bits in a buffer so that they can be retrieved from the buffer in the uninterleaved order. It is the inverse of the transmitter interleave block which reorders the data bits before mapping. This shuffling of bits is an error mitigation technique. At the receiver, the reordering of bits is undone by the deinterleaver in an attempt to spread out the errors resulting from any symbol demapping mistakes throughout the bit stream. This increases the likelihood that the Viterbi decoder can compensate for and correct these errors.

Depuncture Fills in the gaps in the bit stream caused by puncturing at the transmitter. Certain data rates in the 802.11a specification are achieved by dropping bits in a regular pattern from the output of the convolutional encoder output bit stream. Depuncturing is the process of adding back in null values to allow the Viterbi decoder input rate at the receiver to match the convolutional encoder output rate at the transmitter. These null values are ignored by the Viterbi decoder and do not contribute to the decoding process.

Viterbi Decoder The Viterbi decoder [21, 22] restores the pre-channel coded data bit stream. It is the receiver complement to the channel coder (the convolutional encoder) at the transmitter. The function of the Viterbi decoder is to try to re-create the bit stream input to the convolutional encoder at the transmitter. The Viterbi decoder does this by estimating, given knowledge of how the convolutional encoder operates and the received bit stream seen so far, which sequence of bits would be the most likely to have produced this received bit stream from the convolutional encoder.

The Viterbi decoder operates by creating a trellis of all possible convolution encoder states and inputs. This trellis is then compared with the received bit stream and for each set of received bits, the convolutional encoder input bit and state most likely to have produced those received bits is chosen. The convolution encoder input bit of that most likely trellis state is then chosen as the output of the Viterbi decoder. The Viterbi decoder is addressed in more detail in Section 7.4.5.

Descramble Undoes the effects of the transmitter's scrambler. The scrambler/descrambler is simply a linear feedback shift register whose output is XORed with the data stream. This turns the data bit stream into a pseudo-random sequence containing almost equal numbers of 1's and 0's. The purpose of scrambling is to ensure a lively mix of bit values to prevent the DC offset issues cause by a long strings of 1's or 0's. The descrambler is the exact same function as the scrambler and begins in the exact same initial state. This results in a descrambler which presumably, in the error-free bit stream case, applies an XOR of the same value applied by the scrambler XOR at the transmitter. The net result is to negate the effects of the scrambler.

7.3 System Hardware

A prototype Raw wireless system was built as a proof-of-concept. The hardware system provides a platform for the testing of 802.11a and multipass Raw baseband processing in a more realistic environment. It is also a means of verifying the software simulation for correctness and realism. An advantage of the hardware system is that it provides a much higher wall-clock speed equivalent to the software simulation. This was of particular use in verifying the Raw 802.11a single and multi-tile implementations, see Section 7.4.

7.3.1 Raw Wireless System

The Raw wireless system performs both baseband processing and front-end transmission and reception. The 802.11a communication protocol is used for baseband processing. The front-end radio system operates in the 2.4GHz band. Since all of the Raw related baseband processing occurs post down-conversion, the use of the 802.11a protocol versus the more frequency appropriate 802.11g is irrelevant. The use of 2.4GHz is solely an artifact of the RF chip-set used, which was generously donated by the Engim Corporation. Operation at the more traditional 5GHz could easily be achieved by using the wireless board's 5GHz radio path and a 5GHz RF chip-set instead, see Section 7.3.3.

The wireless system (see Figure 7-3) consists of two boards: the Raw Wireless board (see Section 7.3.3) which performs the front-end radio functions for both transmission and reception from the antenna through conversion between the analog and digital domains; and the Raw Handheld motherboard which performs the baseband processing of the digital data stream and control functions for the wireless board. The two boards mate directly through an expansion connector interface through connector headers on both boards. A picture of the full system can be see in Figure 7-4.

7.3.2 Raw Handheld Board

The Raw Handheld Board acts as a motherboard for the Raw processor. The board consists of a Raw processor surrounded by FPGAs which are in turn connected to various types of

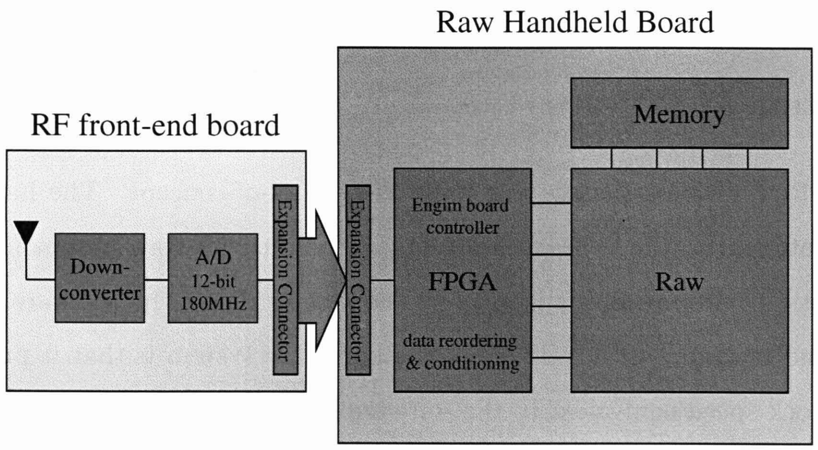


Figure 7-3: An overview of the Raw Wireless System. The system is shown here in receive mode.

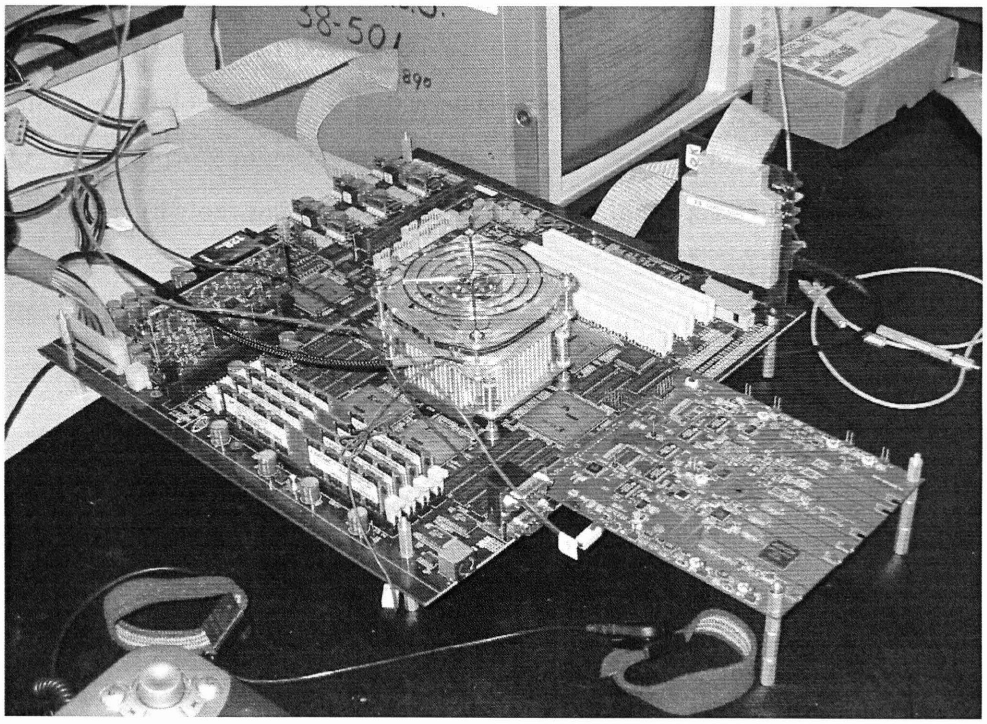


Figure 7-4: Photo of the Raw Wireless System.

I/O interfaces and memory, see Figure 7-5. The FPGAs provide a flexible interface between Raw and the rest of the world. They enable translation between Raw and any connected devices in terms of data formating and clocking. The FPGAs also provide sufficient logic to perform control functions, such as memory control, and minimal data processing. Multiple signaling standards are supported on the FPGAs and enable the ability to translate the electrical signaling standard applied to the data traveling across the expansion connectors between the Raw I/O signaling standard, HSTL, and those used on the expansion boards.

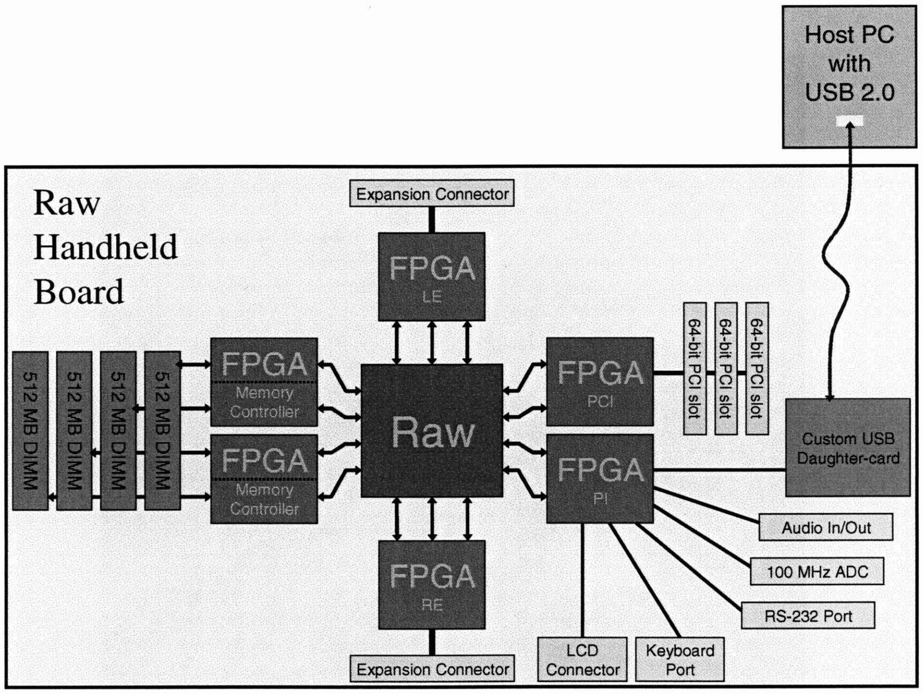


Figure 7-5: Block diagram of the Raw Handheld Board.

The ring of FPGAs act as both logical extensions to the on-chip Raw mesh network and as translators between that mesh network and the rest of the system. As such, the FPGAs can be considered in terms of the Raw ports to which they connect. The two FPGAs on the east ports contain the memory controllers and connect the Raw processor's memory interface to the four 512MB DIMMs which act as system memory for the handheld board. The north and south FPGAs translate between their respective ports and the handheld board's expansion connectors. The expansion connectors allow custom boards, like the wireless board, to interface with the Raw handheld system. The FPGAs provide multiple signaling standards and allow the ability to reformat data coming into and out of the expansion connectors between the Raw network standards and those used on the expansion boards. The west FPGAs provide interfacing with some standard I/O devices and systems. The USB and PCI connections allow the system to interface with standard components and cards. There is also a serial port, a keyboard port, an LCD port and other I/O ports connected to these FPGAs. Currently, the USB interface is used to connect the Raw handheld board to a host computer. This USB acts as a surrogate for a network interface and allows programs and data to be downloaded to and uploaded from the handheld board. A picture of the Raw handheld board situated in its chassis along with an extra Raw chip can be seen in Figure 7-6.

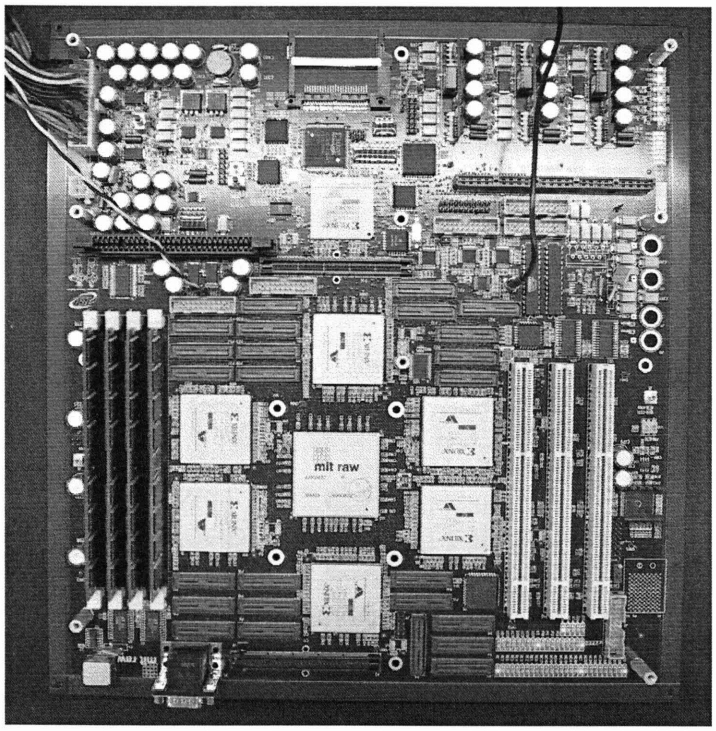


Figure 7-6: Raw Handheld Board.

7.3.3 Wireless Board

The Raw wireless board is a version of an Engim 802.11a/g board design which has been modified to interface with the Raw handheld board. Details of the wireless board implementation and testing can be found in [96]. The wireless board contains Engim chips which provide radio functionality and A-to-D and D-to-A conversion. The converters operate on a 180MHz bandwidth at 12 bits. This is more than is strictly necessary for an 802.11a channel, but should provide the ability to explore multiple channel or larger bandwidth systems, see Section 11.

The wireless board was designed to send and receive data at both 2.4GHz and 5GHz. It is compatible with both the 802.11g and 802.11a specifications. As only the 2.4GHz Engim RF chips were readily available, only the 2.4GHz data path is enabled. This resulted in the scenario of an 802.11a baseband operating with a 2.4GHz front-end. Since, in this case, the front-end processing does not directly affect the baseband processing, the use of a 2.4GHz radio is not incompatible with the use of an 802.11a-only baseband. This is especially true as the system only addresses the PHY layer and operates in a closed environment. As such, there is no need to be concerned about MAC processing capabilities or interoperability with 802.11g headers or other compatibility issues which might arise from operating at 2.4GHz.

The wireless board moves streams of data back and forth between the converters and the FPGA on the Raw handheld board. An expansion connector mates with the Raw handheld board and drivers/receivers on the wireless board communicate the data and clocks between the converters and FPGAs. A picture of the wireless board showing various components along with its connection to the handheld board can be seen in Figure 7-7.

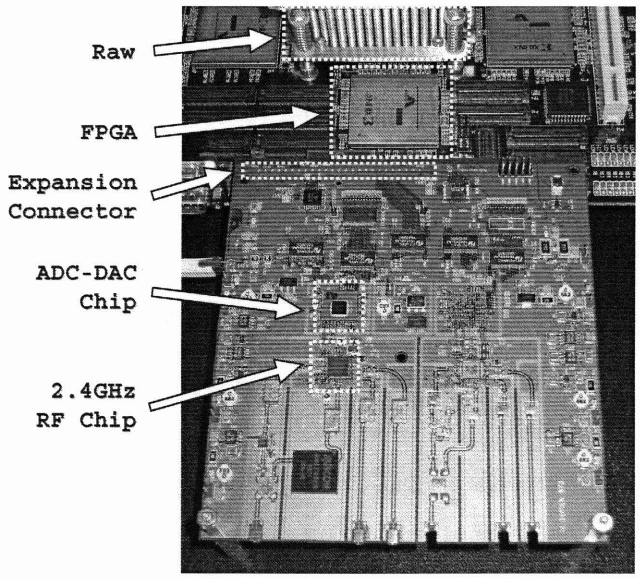


Figure 7-7: Raw Wireless Board.

The wireless board provides three consecutive 802.11a channels of data with 12 bits of precision. This data is streamed from the wireless board into the FPGAs on the handheld board, where it is packetized into messages for injection onto the Raw memory network. In this way, the data stream is stored in the Raw main memory by DMA. The Raw processor can then operate on the data from memory and filter this data down to a single 802.11a channel and begin baseband processing.

The Raw wireless board has been successfully operated in conjunction with the Raw handheld system. The full Raw wireless system has been used to both send and receive data through the wireless interface. The Raw wireless system demonstrates the feasibility of building a wireless radio utilizing Raw.

7.4 System Software

The system software evolved through multiple iterations encompassing single pass versions of the 802.11a baseband running on a single processor, to single pass versions of the 802.11a baseband running on multiple processors, to multipass versions of the baseband running on both multiple and single processors. The implementations focused on the receiver architecture, as this is the most difficult part of a communication system. This stems from the fact that all of the values are deterministic at the transmitter. The uncertainty of the received signal makes the receiver processing much harder and more complex and, as such, the receiver provides an upper bound on the processing required by the communication system. The following receiver software systems have been implemented:

1. C language – we call this Sequential, Single Pass.
2. Matlab – we call this Matlab, Single Pass.
3. Raw single tile targeted C language – we call this Single Tile, Single Pass.
4. Raw multiple tile targeted C language – we call this Multi-tile, Single Pass.
5. Raw optimized multiple tile mixed C and Assembly language – we call this Optimized Multi-tile, Single Pass.
6. Raw optimized multipass mixed C and Assembly language – we call this Optimized Multi-tile, Multipass.
7. Optimized multipass C language – we call this Simulated Optimized Multi-tile, Multipass.

These systems will be addressed in the following sections, with a general overview below.

The first step in building the system software was to generate a generic C language version of the 802.11a receive baseband – the Sequential, Single Pass design. This provided a means of better understanding the specification, as well as a functioning baseband. In order to verify the performance of this single pass design, a Matlab version of the 802.11a baseband – the Matlab, Single Pass design – was created utilizing internal Matlab functions whenever possible. This Matlab system provided verification that the C language baseband functions were implemented correctly and that the C language baseband operated in a similar fashion to the Matlab baseband. From there, the C language design was re-targeted to operate on a single Raw tile – the Single Tile, Single Pass design. This re-targeting was necessary to in order to allow the design to operate on the Raw hardware and the cycle accurate Raw simulator, called BTL, and mostly consisted of adjusting the system to utilize the Raw static network to stream data in and out of the Raw processor. This design was then expanded across multiple tiles – the Multi-tile, Single Pass design. The major baseband function blocks were each assigned a tile and modified to communicate information through the static networks. The slowest operating of these functional blocks, the FFT and Viterbi decoder blocks, were then themselves parallelized across multiple tiles to optimize system performance – the Optimized Multi-tile, Single Pass design. The multipass functionality was then added to this optimized design – the Optimized Multi-tile, Multipass design. Finally, a C functional simulation of this multipass design – the Simulated Optimized Multi-tile, Multipass design – was created to improved system simulation run-time for performance data gathering.

7.4.1 Sequential, Single Pass Design

The initial step in the system design was a C implementation of the 802.11a baseband. This implementation was intended as an introduction to the 802.11a baseband design and an opportunity to understand and debug the algorithms necessary for the baseband to operate correctly. Using a simple implementation platform, the C language on a Pentium 4 processor, allowed the use of the debugging infrastructure and high compilation and processing speeds available in that environment. This helped ease the initial implementation effort and allowed algorithm exploration and development to be the primary focus of this implementation. The example message given in the 802.11a specification [24] was used heavily to verify this implementation and the implementations to follow. This implementation included transmitter, channel, and receiver portions. The transmitter and channel were needed to generate input for this and later receiver designs.

Figure 7-8 plots the Bit-Error Rate (BER) curves for the C language single pass system implementation operating with two modulation schemes, QPSK and 16-QAM. A BER curve gives a metric of communication system performance. It plots the log of the probability of a bit error (the likelihood that a bit input to the system is not the bit output by the system) against the relative adversity through which the system must operate (denoted in decibels). The measure of this relative adversity is called $\mathrm{E}_b/\mathrm{N}_0$ and is the ratio of the energy per information bit, E_b, to the noise power spectral density, N_0. $\mathrm{E}_b/\mathrm{N}_0$ is a measure of the Signal-to-Noise Ratio (SNR) of the symbols received scaled by the number of information bits per symbol. The performance of a communication can be represented by how reliably the system performs (how many errors it makes in communicating information bit) in the presence of noise. Given that any correctly operating communication system can transmit information with negligible probability of error with large enough signal power, or equivalently small enough noise, the goal is to minimize the power required to do so. This corresponds to achieving more reliable performance for smaller $\mathrm{E}_b/\mathrm{N}_0$.

BER curves are typically monotonically decreasing, as can be seen in Figure 7-8, as increased signal power (or decreased noise) translates into more bits being correctly communicated. Better communication system performance corresponds to curves which are shifted further to the left. In Figure 7-8, the QPSK curve falls to the left of the 16-QAM curve showing that the QPSK system is more reliable in noisy conditions. The trade-off here is that a 16-QAM system can operate at twice the data rate of a QPSK system. Therefore, if the noise power is low, it is more efficient to use a 16-QAM modulation scheme; otherwise, the QPSK system is more reliable.

The two curves presented in Figure 7-8 are the BER curves for the single pass 802.11a baseband using QPSK modulation and 16-QAM modulation. These curves represent a baseline metric for the multipass system. For a multipass system to be useful, it must deliver communication performance on the order of that of an equivalent data rate single pass system. These two single pass curves are provided on all future BER plots (in Chapter 8) as a point of reference.

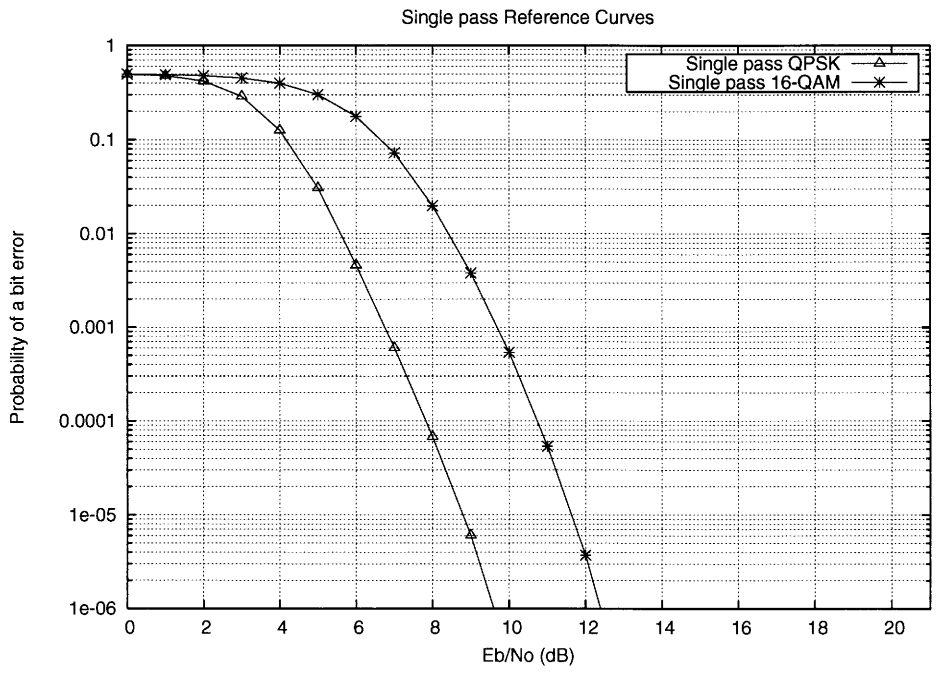


Figure 7-8: The Bit-Error Rate (BER) curve for the C language single pass system implementation. The x-axis show E_b/N_0 in decibels. E_b/N_0 represents the ratio of the energy per bit to the system noise power. It is a measure of the Signal-to-Noise Ratio (SNR) of the system scaled by the number of information bits per symbol. The y-axis denotes on a log scale the probably that a bit input to the communication system is output erroneously. The performance of each of the single pass systems is mapped by plotting the how likely the system is to make a bit error as the relative amount of noise changes.

7.4.2 Matlab, Single Pass Design

A Matlab version of the 802.11a baseband was also implemented. The Matlab version was built using built-in Matlab functions whenever possible. This provided an additional validation of the C implementation results. An additional benefit was the ability to test both the C and Matlab versions with an AWGN source added to the channel model. This provided an additional test of the noise compensation abilities of the baseband implementations.

A graph of the bit error rate (BER) of the Matlab and C single pass versions can be seen in Figure 7-9. The outputs closely align and provide confidence in the C language implementation. The relative choppiness of the Matlab BER curve is a result of fewer packets being used to generate the Matlab curve than the C model curve.

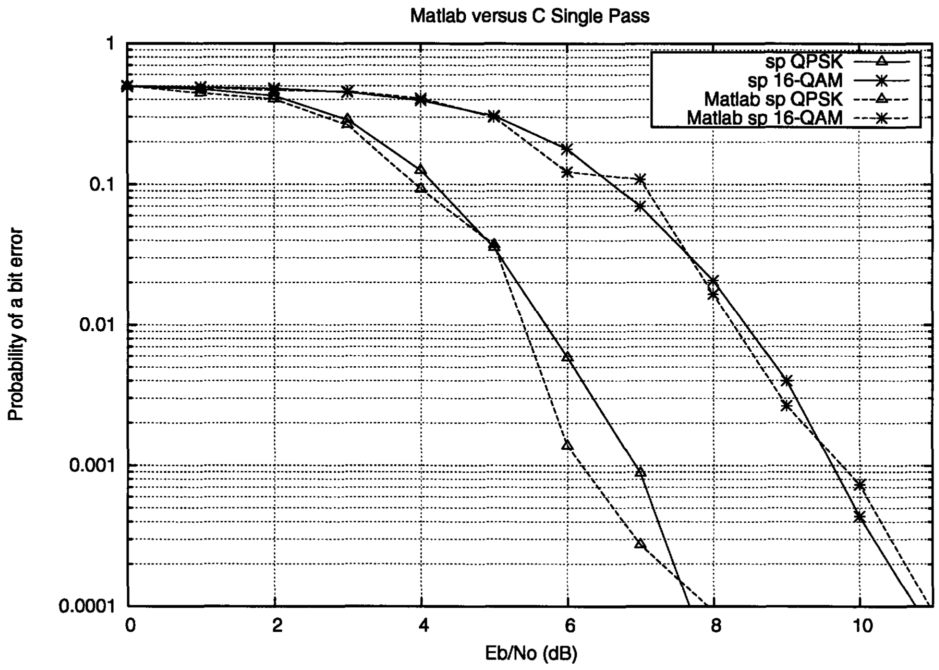


Figure 7-9: The BER of the Matlab implementation of the single pass system is compared to the C implementation.

The Matlab model was also used to validate the final optimized C language model in Section 7.4.6.

7.4.3 Single Tile, Single Pass Design

Following the validation of the C implementation on the P4, the software was ported to run on a single tile of Raw. This version of the baseband targeted a single tile of the Raw processor and could take advantage of the Raw gcc back-end, or rgcc, which targets a single Raw tile [74]. The porting to Raw simply involved removing any unsupported system level calls and building the software infrastructure to stream the data into and out of the tile running the baseband. This was achieved using the static network.

The initial single tile design was tested on BTL, a cycle-accurate simulator of the Raw processor [74], due to its superior debugging capabilities. The Raw hardware was used in conjunction with BTL for debugging, however, as it operated significantly faster than the simulation. The single tile system, running on both BTL and the Raw hardware, was validated against both the 802.11a specification and the previous C implementation using the P4 model.

The single tile 802.11a baseband provides an intermediate step in the porting of the baseband to Raw. It is a means of verifying that the software used is Raw compatible, but, as it only uses a single tile, it fails to utilize most of the advantages of the Raw processing platform. This results in the single tile system being very, very slow, requiring many thousands of cycles to operate on each received data symbol. This is much too slow to contemplate real-time communications operation.

7.4.4 Multi-tile, Single Pass Design

To improve the sustainable data rate of the 802.11a receiver baseband, the single tile Raw port was spatially pipelined over multiple Raw tiles. This allowed the baseband to use multiple processors in parallel to reduce the load on each individual tile. The design was broken up primarily along functional block lines, although compatible functions were combined on some tiles. The mapping of functions to tiles can be seen in Figure 7-10. The final multiple tile implementation used 10 tiles and distinct input and output ports.

The multiple tile design makes heavy use of the Raw static network and required programming the switches for both data flow and control flow. Hand coding the static network necessitated careful orchestration of communication between tiles which, in turn, gave greater insight into the best means of parallelizing across multiple tiles. The final mapping is as close to a linear flow of data as possible. The distribution among tiles generally follows functional boundaries, but functions which operate in mutually exclusive and/or complementary phases, such as channel estimation and equalization, or cyclic prefix remove and FFT, were collocated on the same processing tile.

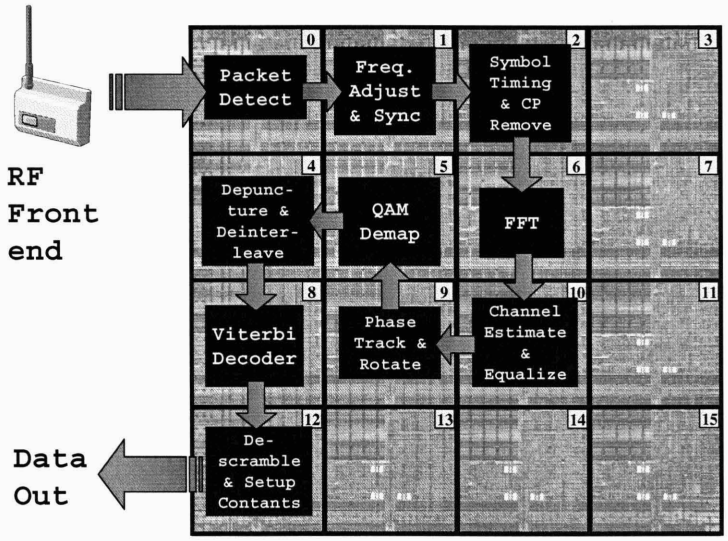


Figure 7-10: Multiple tile receiver mapping on Raw. Map receiver blocks to a different tiles to create a spatially pipelined receiver chain.

Since all of the functions were migrated from the single tile implementation, they are in the C language. The primary modification to the functions themselves was to reformat their inputs and outputs to use the network instead of memory structures. The use of C was very helpful in speeding up the implementation process and led to a functional system which could then be used to determine the major system bottlenecks.

A significant benefit of the multiple tile design was its relatively small size. By only using 10 tiles, the entire multiple tile design could fit on the current Raw hardware. This allowed the use of both BTL and the hardware for debugging, as in the single tile case. The significantly faster hardware running time was extremely helpful in quickly converging on a fully functional system. Once again, the 802.11a specification example, as well as the single pass version outputs, were used to verify and validate the identical Raw hardware and BTL multiple tile baseband.

The BTL simulator provided detailed information on the number of cycles required to perform each function in the multiple tile baseband implementation. This demonstrated two functions which significantly impacted system performance: FFT, and the Viterbi decoder. It is not surprising that these functions were throttling critical path performance, as each acts as one of the bottlenecks described in Section 1.2.2. In order to achieve processing rates compatible with real-time communication system operation, it is necessary to significantly improve the efficiency of each of these bottlenecked functions.

7.4.5 Optimized Multi-tile, Single Pass Design

The multiple tile baseband simulation highlighted two main functions which impeded high data rate baseband operation. These bottleneck blocks needed to be optimized and parallelized to achieve necessary processing speeds. This was done by rewriting the functions to spread them out over multiple tiles. The code for each tile was rewritten in optimized Raw Assembly language and the communication between the tiles was carefully orchestrated. The two major bottlenecks to high speed operation were the FFT function, a data chunking bottleneck, and the Viterbi decoder function, a data expansion bottleneck. The optimization of these functions was key to creating a system compatible with real-time operation.

The other functions in the multiple tile baseband system were not optimized. These other functions were not major contributors to system slowdown. Such functions could be relatively easily rewritten in Raw Assembly language if their operation began to impact system performance. The non-optimized functions fall into two categories: those operating on symbols and therefore, at the symbol data rate, and those operating on bits.

The symbol functions, those functions which come before the demapping stage, operate at the 20MHz symbol rate or slower. This slow rate allows ample time for the temporal reuse necessary for the symbol functions to perform their operations using a single tile. The FFT, which experiences the data chunking bottleneck, is the only block of primary concern here.

The bit functions suffer from a much more constrained timing as the rate of bits is higher than that of the symbols. Since each symbol can carry up to six bits, the rate of bits can grow by many multiples through the demapping stage. The only bit function for which this increase in bit rate is of major concern is the Viterbi decoder function as it requires significant amounts of processing for each bit input. All of the other bit function blocks either perform no computation on the data, only reordering or reformatting, or else very minimal computation, such as a single XOR in the descrambler.

FFT

The Fourier Transform and its inverse are functions which convert data between the time and frequency domain[41, 42]. Fourier Theory states that time domain signals can be losslessly decomposed into the superposition of complex sinusoids. The Fourier Transform is a means of quantifying the relative contribution of each of a set of harmonically related complex sinusoids at set frequencies in the composition of a time domain signal. The transform takes in N data values in the time domain and outputs N frequency values, or points. Each frequency point corresponds to the relative amount of a complex sinusoid at each of N frequencies, equally spaced by $\frac{2\pi}{N}$ between 0 and $(\frac{2\pi}{N})(N-1)$, in the time domain signal. The Fast Fourier Transform (FFT) is a computationally efficient version of a Fourier Transform for transform sizes which are exponents of 2 (size $= 2^N$) [23]. The Inverse Fast Fourier Transform (IFFT) performs the dual function to the FFT. The IFFT takes N frequency points and transforms them into N time domain data values. An example FFT flow diagram can be seen in Figure 7-11.

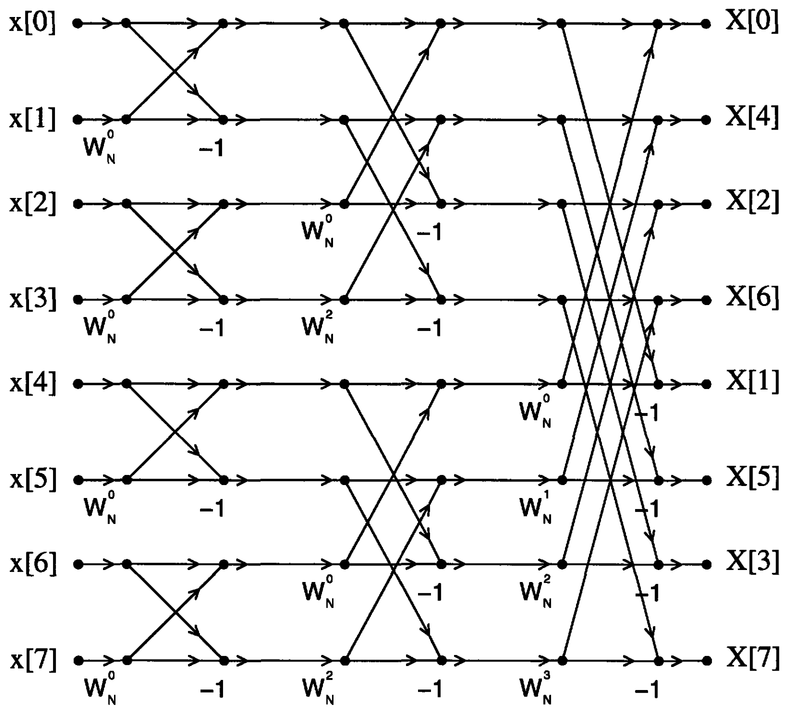


Figure 7-11: A small example FFT dataflow. The size of the FFT is 8-points ($N = 8$).

The core element of the FFT is the butterfly, see Figure 7-12. The butterfly is applied repeatedly with different constant coefficients to different sets of data to implement the FFT. The flow diagram of Figure 7-11 is simply repeated stages of butterflies using different coefficients and different strides in the communication between data values. While the basic structure of the FFT butterfly stays the same for each stage, or column, of the FFT flow diagram, the stride, or distance between each butterfly's input data, grows logarithmically with each stage.

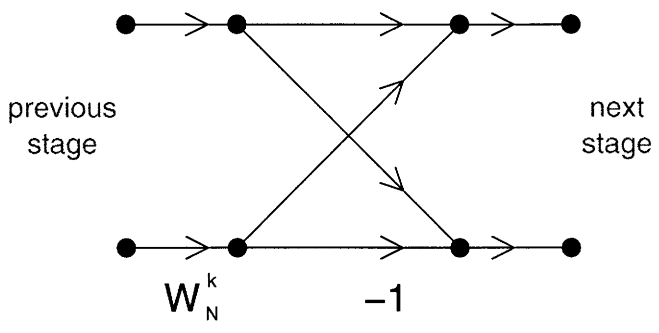


Figure 7-12: The butterfly is the core functional element of the FFT.

The key questions for the parallelization effort of the FFT are how to trade-off communication versus computation and how to manage that communication which is required. The distribution of the butterflies has a direct impact on the amount and types of communication, as well as the efficiency of computation of each butterfly.

The FFT butterflies do not map well to mesh networks in two regards. As only vertical and horizontal nearest neighbor connections are present in a mesh, but not diagonal connections, the elements of a butterfly do not map well across tiles. This diagonal routing makes locating all of a butterfly on a single tile much more efficient. Secondly, if each butterfly is located on a single tile, and not spread across tiles, the gathering of the butterfly input data requires a larger amount of communication with each FFT stage. The stride between stages of butterflies, however, grows logarithmically. This logarithmic growth of the butterfly strides between stages of the FFT requires the reshuffling of data in between every stage before the next set of butterflies can commence. That makes mapping individual butterflies, or even sets of butterflies, onto a single tile relatively communication expensive as, for the later FFT stages, every tile requires a large amount of communication over a long distance.

This issue is addressed in the optimized Raw FFT by partitioning the FFT into phases (see Equations 7.1– 7.5). Each phase is assigned a number of consecutive stages of the FFT flow diagram. A stage's worth of butterflies are then allocated across the number of tiles to be used. With intelligent partitioning of the butterflies across the tiles, a single set of data is all that is required during a phase. That is, the butterflies of a phase can be distributed across the tiles in such a manner that the output of each butterfly stage is consumed by a butterfly in the next stage located on that same time. This reduces the communication problem to a shuffling of the data between phases to marshal the correct data values on each tile for the phase to come. While this requires an all-to-all communication in between phases, it results in a smaller amount of overall communication.

$$
\begin{aligned}
N &= \text{number of FFT points} && (7.1)\\
T &= \text{number of processing elements available} && (7.2)\\
X &= \text{number of FFT points assigned to each tile} && (7.3)\\
&= \frac{N}{T} && (7.4)\\
\text{Phases Required}(\Phi) &= \left\lceil \frac{\log_2(N)}{\log_2(X)} \right\rceil && (7.5)
\end{aligned}
$$

The optimized Raw FFT used here is designed with two phases of 8 tiles, each requiring 4 FFT butterflies and 8 complex data values per tile, see Figure 7-13. The phases are allocated to two different sets of 8 tiles. This simplifies the communication patterns involved as the all-to-all communication becomes a forwarding of values between phases as each of the 8 outputs of each Phase 1 tile is sent to a different Phase 2 tile.

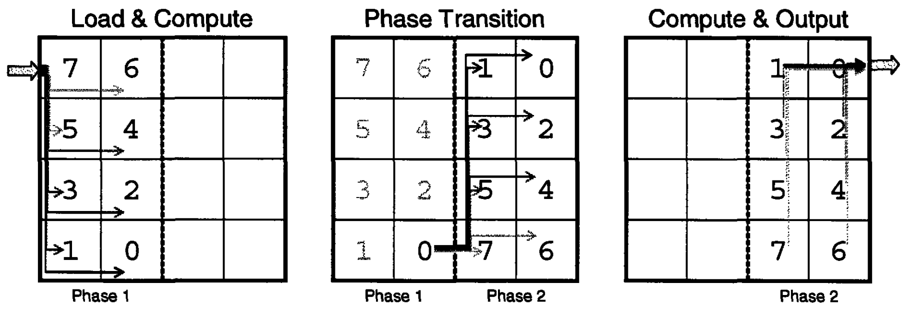


Figure 7-13: The optimized FFT operates in phases:

Load & Compute	The data is loaded into each tile (darker arrow signifies earlier communication). When each tile has loaded all its FFT values it begins computing, while the next tile in the load order receives its values.
Phase Transition	The values from each tile in Phase 1 are distributed to the tiles in Phase 2. As soon as a Phase 1 tile is finished computing its output, values are sent to each of the Phase 2 tiles (darker arrow signifies earlier communication).
Compute & Output	The second set of FFT butterflies are performed and the results output. When all tiles have finished their Phase 2 butterflies, their final values are output in a round-robin ordering (darker arrows signifies earlier communication).

The FFT performance is also improved by the pipelined nature of the communication involved. Each Phase 1 tile can begin its butterfly computations as soon as all of its data values are present. The butterflies are allocated across the Phase 1 tiles in a manner such that each consecutive set of 8 data values are all of those necessary to allow one whole tile's worth of FFT computation. Thus, once the first 8 data values are input to the FFT, the first tile to receive data can start its computation, even while the later tiles are still receiving data.

The staggering of the Phase 1 computations eases the forwarding of data values for Phase 2. When each Phase 1 tile finishes its computations, it must forward each of its output values to a different tile. The staggering of the start of computation on Phase 1 tiles translates to a staggering of the end of computation on those tiles and similarly staggers the communication for each tile between Phase 1 and Phase 2. This helps to reduce the amount of network traffic at any given time and thus the routing between Phase 1 and Phase 2 is significantly simplified and is as efficient as possible.

The use of multiple phases on separate sets of tiles also allows the FFT to be pipelined for even greater efficiency. The two FFT phases are disjoint and only interact during the interphase forwarding of data. The operation of each phase is also independent for each set of FFT input data. Therefore, each phase can begin to work on the next FFT operation as soon as it completes. For instance, once a Phase 1 tile has completed its computation on a set of data and forwarded that data on to Phase 2, it can immediately begin computation on the next FFT operation's set of data. This reduces the data chunking bottleneck and improves the throughput of the overall FFT.

The number of phases and tiles per phase were carefully chosen to balance the computation and communication latencies for each tile. This allows the pipelined FFT to remain fully occupied. It should be noted, however, that this balance is dependent on the specific architectural latencies involved and cannot necessarily be maintained with increased parallelization. Therefore, spreading the FFT across more than the 16 tiles used here will provide diminishing returns.

Table 7.2 compares two different FFT implementation approaches using Raw. Both implementations use the same basic algorithm for the FFT utilizing repeated radix-2 butterflies. Both FFT's implement an 802.11a specification-compatible 64-point FFT.

The first of these is a straightforward C implementation on a single tile. This version

Fast Fourier Transform (FFT): 3.2μs to perform 64-point FFT			
Implementation	*Number of Tiles*	*Cycles*	*Raw Clk*
C	1	100,000	31.25GHz
Raw optimized	16	240	75MHz

Table 7.2: Comparison of FFT Implementation Properties.

is written in C and compiled with rgcc, which targets a Raw tile. This implementation avoids the communication overhead of a spatially parallelized version of the FFT, but is significantly slowed down by the memory bottleneck of performing the butterflies. This occurs in spite of the fact that the memory footprint of the FFT is smaller than a single tile's data cache size of 32KB. A lot of memory traffic is generated by the FFT both in terms of the need to store large chunks of data, see Section 1.2.2, and in the shuffling of data and constants to perform the shifting butterflies. By using a single processor, the register file is constantly being spilled to swap out the values from the last butterfly and this creates a lot of churn in the memory subsystem.

This large overhead results in a relatively large cycle count per 64-point FFT of $\approx 100,000$ cycles. The 802.11a specification allows a maximum time of 3.2μs per FFT (or IFFT). This results in a minimum clock speed of 31.25GHZ to perform the FFT, an uncomfortably high frequency, especially since, in a single processor design, the entire processor would be required to perform nothing but FFT's in order to operate at that rate.

The optimized Raw FFT implementation is also presented in Table 7.2. This is a parallel Raw Assembly language implementation which uses 16 tiles to carefully trade-off the computation and communication requirements of the 64-point FFT. The optimized Raw FFT distributes the FFT butterflies over many tiles to avoid the use of the memory subsystem on any given tile. The number of butterflies per tile was carefully chosen to match the register file size. The use of the network to forward data between the computation phases of the optimized Raw FFT also avoided the use of the much slower memory network to reorder data, as was required in the C language FFT version. These efficiencies are reflected in the large improvement in performance of the optimized Raw FFT over that of the C language version. The clock cycles per FFT function were reduced by three orders of magnitude and the minimum clock speed was lowered to a very manageable 75MHz.

This large performance increase is a result of improvements to the optimized FFT over the C language FFT on multiple fronts. First, the optimized FFT achieves a 16× improvement through the parallelization of the function across 16 tiles. The FFT was carefully pipelined to keep all of the tiles working all of the time and to balance the latencies of computation and communication to keep them roughly equivalent, resulting in the linear speedup with 16 tiles. Second, the careful scheduling of instructions and precomputation of necessary constant coefficients translates to a 4× reduction in the number of cycles required for each butterfly computation. Third, the optimized FFT achieves an additional 5× improvement in cycle count by minimizing the number of instructions required for operations other than the butterflies. These extra instructions in the C language FFT are primarily the result of the reading, writing, and reordering of memory structures, all of which are inherently handled through the careful communication of values across the network in the optimized FFT. Finally, the optimized FFT achieves an additional performance benefit of approximately 1.5× through the careful scheduling of instructions to avoid processor stalls. Though the C language version was compiled using rgcc with optimization level of `-O 3`, this version still encounters a number of stalls (bypass, misprediction, and resource stalls) which the optimized FFT has been designed to avoid completely.

Viterbi Decoder

The Viterbi decoder is a receiver processing function which decodes the channel coding performed by the convolution encoder at the transmitter. Convolutional encoders provide channel coding by correlating consecutive data bit values through the encoder's state information. The robustness of this channel coding and correlation can be improved by having the convolutional encoder use multiple bits to represent each combination of current coder state and data. At the receiver, this convolutional encoding must be undone to retrieve the originally intended bits. The Viterbi decoder block performs this retrieval process. In order to regenerate the pre-encoded bits, however, the Viterbi decoder must build a trellis representing all of the possible states and inputs which might have occurred at the convolutional encoder in the transmitter. The Viterbi decoder then performs a winnowing process to restrict the trellis outputs to only the most likely states. Eventually, this process converges on the most likely data bit for the transmitter to have sent and this is considered to be the received data bit value.

Data expansion occurs in the building of the Viterbi decoder trellis. In order to represent all of the possible states and input, the amount of processing required at the receiver rapidly blows up to many times that of the data rate. For each two bits input to the Viterbi decoder, a large trellis must be constructed and operated on. This represents a very large relative increase in processing rate.

Most of the focus in Viterbi decoder design has been on ASIC and custom logic [97, 98, 99, 100, 101, 102]. This is a consequence of the current generation of GPPs and DSPs not having the processing capabilities to perform high speed Viterbi decoder operations [72]. There are just too many instructions required in too small a period of time and the related growth in bit rate makes implementing the Viterbi decoder on a single general processing element infeasible, see Section 1.2.2. Even using simply connected multiple cores, as in many CMPs, is insufficient. This is because the trellis structure of the Viterbi decoding requires a large number of differing computations for each stage of the decoder. These computations must therefore be distributed with a very fine grain of parallelism to achieve acceptable performance. In addition, the different processing tasks of the Viterbi decoder are interrelated, requiring very low latency interconnect between the distributed processing elements [103, 104].

The Viterbi decoder is used as a maximum likelihood estimator of the data bit stream input to the transmitter's convolutional encoder [21, 22]. The Viterbi decoder attempts to determine, given the bits seen at the receiver, which bits were most likely to have been input to the convolutional encoder at the transmitter. The decoder is attempting to re-create the convolutional encoder input stream using only the demapped bits of the received noisy values. The use of the convolutional encoder/Viterbi decoder pair provides channel coding for the communication system. This method of channel coding increases the likelihood of successfully recovering the transmitted data stream in two ways:

First, the convolutional encoder output for each input bit is influenced by the previous few bits input to the convolutional encoder through the convolutional encoder state elements, see Figure 7-14. This spreading of the bit information allows the Viterbi decoder to use a number of consecutive bits in estimating the most likely convolutional encoder input to have created those bits. Thus, even if some of the received bits are in error, the information contained in the correct bits around those errors can help to promote or recover an accurate decoding and re-creation of the convolutional encoder input bit stream.

Second, the convolutional encoder uses the redundancy of using a coding rate of less than one to improve the robustness of the system. A coding rate of less than one means that for a given number of input bits, a larger number of output bits is generated. These extra output bits give the decoder extra information to help determine the decoded bit stream. The rates used in 802.11a coding are 1/2, 2/3, and 3/4. A rate of 1/2 corresponds to two convolutional encoder output bits for each input bit, a rate of 2/3 means there are 3 output bits for every 2 input bits, and a rate of 3/4 has 6 output bits for every 4 input bits. The higher the coding rate, the more data bit information is encoded in the coded data stream (hence the use of these rates to help increase the system data rate). This also results in a smaller amount of redundancy and a corresponding higher probability of error in the decoded bit stream.

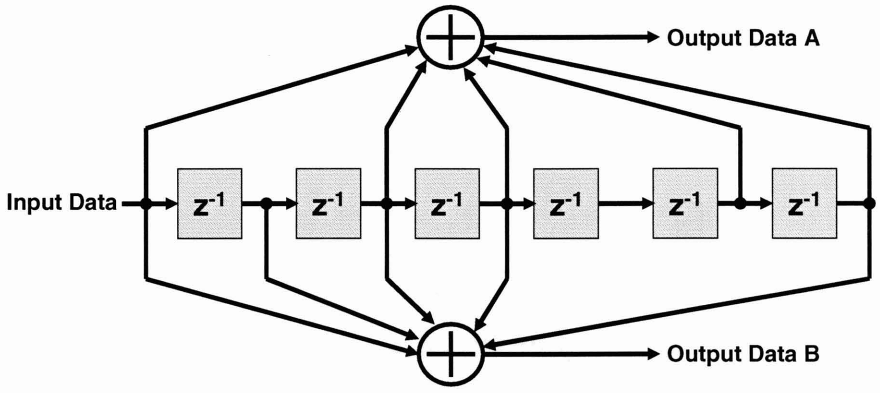
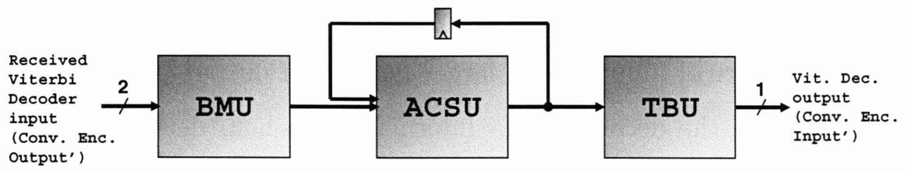


Figure 7-14: 802.11a Convolutional Encoder of depth $K = 7$ and rate $R = 1/2$ with generator polynomials $g_0 = 133_8$ and $g_1 = 171_8$.

The Viterbi decoder can be broken down into multiple blocks or units which collectively perform the decoding [105], see Figure 7-15. The required blocks for performing Viterbi decoding are the Branch Metric Unit (BMU) block, the Add-Compare-Select (ACS) block, the Memory (MEM) block, and the TraceBack Unit (TBU) block. The specific Assembly language implementation used on Raw adds two more blocks to perform the general TBU functions more efficiently: a second stage pipeline of the TraceBack Unit (TBU2) block, and an Accumulate (ACC) block which accumulates and condenses the TBU/TBU2 output into a single bit decision. The functions of the various units/blocks used in the optimized Raw Viterbi decoder are described below and the Raw mapping used can been seen in Figure 7-16.

BMU (Branch Metric Unit) Compares the bit pair input to the Viterbi decoder with the output of every possible combination of inputs and states for a convolutional en-


Received Viterbi Decoder input (Conv. Enc. Output')
2
BMU
ACSU
TBU
1
Vit. Dec. output (Conv. Enc. Input')

- BMU: Branch (or Transition) Metric Unit
  - Determines all possible branches (or next-states) from each current-state and their likelihood (branch metric) given the decoder input. Each branch corresponds to a possible input bit to the *Convolutional Encoder*. The metric is the difference between the potential encoder output and the actual decoder input.
- ACS: Add-Compare-Select Unit
  - For each next-state, adds the branch metric for all incoming branches to likelihood we were in that branch's current-state (path metric). Compares results and selects the "best" total metric. This becomes the new path metric and the corresponding next-state becomes the current-state.
- TBU: Trace-Back (or Survivor Memory) Unit
  - Stores the input bit value for each new current-state into that state's survivor memory. The oldest value from the current-state with the "best" path metric is the decoder output. The oldest value from all survivor memories is then dropped.

Figure 7-15: An overview of the general functional blocks of the Viterbi decoder.

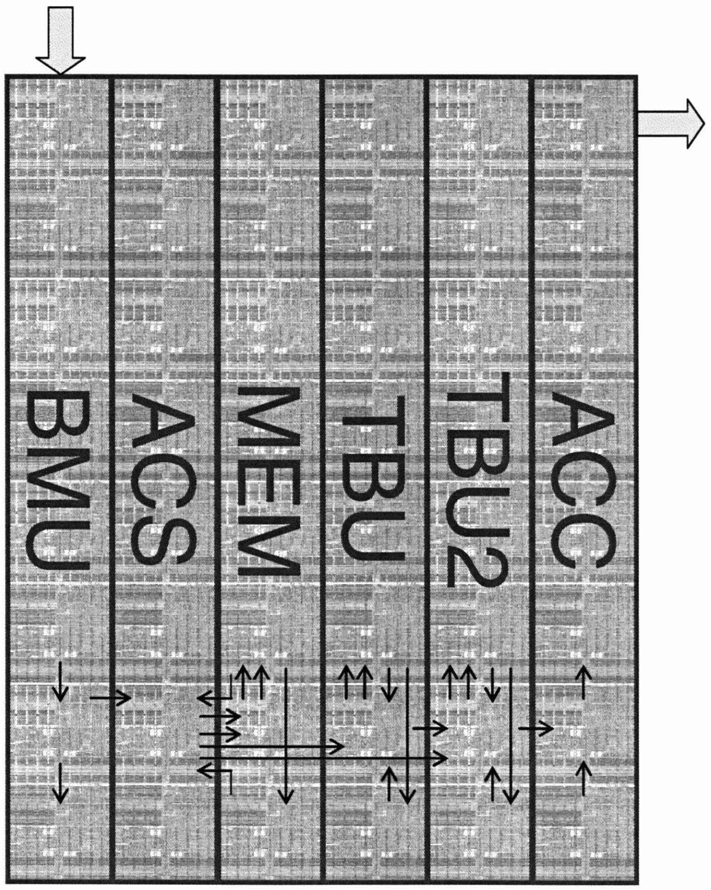


Figure 7-16: Parallel Viterbi decoder layout. The communication pattern for a single row is show in black.

coder. The resulting branch metrics are the inverse hamming distance (or the number of matching bit positions) between the Viterbi decoder input pair and the possible convolutional encoder outputs. The branch metric for a state-input combination represents the likelihood that a given Viterbi decoder input pair was generated by a convolutional encoder in that state with that value as the input. A branch metric is determined for each convolutional encoder state and possible convolutional encoder input. These branch metrics are generated for each pair of inputs to the Viterbi decoder. (The Viterbi decoder operates on pairs of bits because the base coding rate for this system is 1/2. Hence, two convolutional encoder outputs/Viterbi decoder inputs for each data bit.)

There are $K-1$ possible state values for a depth K convolutional encoder. These 2^{K-1} states correspond to the 2^{K-1} bit permutations which could be stored in the $K-1$ state elements of the convolutional encoder. In addition to the 2^{K-1} possible states, there are 2 possible bit values (0 or 1) which could have been input to the convolutional encoder for each state. Therefore, for the 802.11a convolutional encoder which has an encoding depth of $K = 7$, there are $2^{K-1} = 64$ possible states and 2 possible inputs which leads to $64 * 2 = 128$ branch metrics. The BMU assigns each one of these 128 branches a metric based on the bit pair presented to the input of the Viterbi decoder.

ACS (Add-Compare-Select) Chooses the most likely state-input combinations, reducing the total number of state-input combinations to $128/2 = 64$.

Each possible state of the convolutional encoder has a value called a Path Metric (PM) associated with it. This PM value represents the likelihood that a possible convolutional encoder state was the actual state of the transmitter's convolutional encoder at a given point in the data stream.

The *add* part of the ACS adds the BM, which is supplied by the BMU, to the PM for each state, which is supplied by the MEM unit. By adding the BM to the PM, the ACS finds the cumulative likelihood that the convolutional encoder was in a given state (the PM) and that that state and a given input produced the incoming Viterbi decoder pair (the BM). The ACS then *compares* these cumulative likelihoods and, for each resultant state of the Viterbi decoder, *selects* the state-input combination of the aliasing pair which is more likely. The cumulative BM and PM of each selected state then becomes the new PM of the resultant state. This new PM value is sent to the MEM unit to make it ready for the next Viterbi decoder input. The input value of each selected state-input combination is also sent to the TBU as the chosen convolutional encoder input for that convolutional encoder time step.

MEM (Memory shuffle unit) Provides the PM for each state-input combination to the ACS, receives the new PM's from ACS following the ACS computations, and then shuffles those PM's through the network such that the PM for each state is stored in a readily available location for usage in the next time step's ACS.

TBU (Trace Back Unit) Keeps track of the input bits selected by the ACS for each state selected by the ACS. The TBU is the first 32 bits of a 64 bit FIFO. Each input bit sent to the TBU from the ACS is added to the front of the FIFO which corresponds to the selected state while the oldest bit is sent on to TBU2.

The TBU contains a FIFO word for each possible state of the convolutional encoder. The TBU input for each state is the selected bit from the ACS unit. This ACS selected bit is used for two related functions in the TBU. First of all, it tells the TBU which of the two possible aliasing states resolved into a given state for the next time step. The TBU uses the information to assign the selected aliasing state's FIFO word to become the state word of the resultant state. Secondly, the ACS selected bit is then input to the resultant state's FIFO word to become the newest entry. The oldest entry from this FIFO word is sent to the TBU2 along with the ACS selected bit.

TBU2 (Trace Back Unit 2) Keeps track of the input bits selected by the ACS for each state selected by the ACS. The TBU2 is the second 32 bits of a 64 bit FIFO. The TBU2 has two input bits for each state. The selected bit from the ACS determines which FIFO word is assigned to which state, *à la* the TBU. The second input bit is the FIFO output bit of the TBU, as the TBU2 simply acts as a second 32 bit continuation of the TBU function.

A second TBU unit is needed to achieve 64 bits of state word storage. The use of 64 bits allows better convergence of the traceback function. The output of each of the TBU2 FIFOs is sent to the ACC in order to resolve a final Viterbi decoder output.

ACC (Accumulate unit) Uses a majority decision among the TBU2 output bits to make a final Viterbi decoder output data bit decision. Performs an accumulation of all the TBU2 output bits to determine if there are more 1's or 0's. The majority value among these bits is used as the Viterbi decoder output. In the absence of errors, all of the Viterbi states will trace back to the same originating convolutional encoder state and input bit. As the amount of error increases, not all states will have this perfect case traceback, but for recoverable errors, a majority of states will. The convolutional encoder input value of this majority vote state-bit combination is then the maximally likely convolutional encoder input bit and the best re-creation of the data stream bit and is presented at the Viterbi decoder output. This majority vote method is slightly suboptimal to performing a full trace back through of the path metrics, but it is much more computationally efficient [106].

A comparison of the performance of the Raw optimized Viterbi decoder written in Assembly language with C language implementations targeting a single Raw tile is shown in Table 7.3. The näive C language implementation is a reasonably written, but only marginally optimized, Viterbi decoder implementation. It is an acceptable implementation which faithfully reproduces the vanilla Viterbi decoder algorithm. It was compiled with rgcc using an optimization level `-O 3`. The result is a Viterbi decoder which requires $\approx 300,000$ cycles per Viterbi decoder output bit. A prohibitively fast processor of 16.2 THz would be required to allow this Viterbi decoder to operate at 54Mbps (the maximum rate for the 802.11a specification). The main impediment to fast operation for this Viterbi decoder is the huge amount of memory traffic required. All of the trellis structures, metrics, and FIFOs must be stored and retrieved from memory. The large number of trellis states, and the need to access every state's information for every input, results in frequent memory accesses. The representation of data is not wasteful, but neither is it especially concise, leading to a large amount of data to be processed for each input. The memory accesses account for a large part of this Viterbi decoder's latency. Another impediment to fast Viterbi decoder operation is the algorithms used in this implementation. No algorithmic short-cuts are used here. The algorithms are not tailored to the specific Viterbi decoder design or the processor used.

The second line of Table 7.3 represents a more sophisticated C language implementation of the Viterbi decoder using the same streamlined algorithms as in the Raw Assembly version, but still targeting just a single tile. The non-Näive C language implementation uses

Viterbi Decoder: At Maximum 802.11a Rate of 54Mbps			
Implementation	*Number of Tiles*	*Cycles*	*Raw Clk*
Näive C	1	≈300,000	16.2THz
Non-Näive C	1	≈150,000	8.1THz
Raw optimized	48	70	3.78GHz

Table 7.3: Comparison of Viterbi decoder Implementation Properties.

similar algorithmic approaches to that of the Raw Assembly version in an attempt to normalize that aspect of the design. More computationally efficient versions of the algorithms were applied and algorithms were specifically adjusted to match the Raw processor characteristics (e.g., to allow the use of specific ISA instructions, or designed to take advantage of the Raw word size). This design also attempted to minimize memory usage. Smaller, more efficient and Raw-friendly data structures were used and a minimal set of state information was saved. This optimization effort resulted in a significant reduction of requirements. The number of cycles per output observed fell $2\times$ to $\approx 150,000$ cycles. While this is a large improvement, such an implementation would still require a dedicated processor operating at an exceptionally high rate of 8.1 THz.

The multi-tile Raw optimized Viterbi decoder is capable of fully utilizing the parallelism of the Raw processing platform to achieve a much higher rate Viterbi decoder. The algorithms used in this implementation were chosen to work well with the underlying Raw hardware and were specifically tailored for the specific Viterbi decoder implemented. The design was distributed across 48 tiles in order to minimize the processing load on any given tile and to minimize critical paths. In addition, the design was partitioned to take advantage of the static networks and streaming nature of Raw. The design was carefully crafted to keep all currently useful data values live and to avoid cache accesses at all costs. The register files of the tiles, as well as the buffers in the static network, were used for storage in lieu of the memory subsystem. The partitioning used was chosen to equalize the computation requirements for each tile with the communication costs of transporting the data to and from those tiles across the network. For example, the number of cycles needed to permute all of the data across the static networks in the MEM units is roughly equivalent the number of cycles required for the ACS unit to complete its computations which prevents either unit from stalling while waiting for the other unit. This maximizes the benefits of parallelization, but means that further parallelization would not result in additional speedups, as the communication-computation balance would be skewed.

The result was an optimized Viterbi decoder capable of operating at 70 cycles per output. In order to build a 54Mbps data rate using such a Viterbi decoder, a Raw processor running at 3.78GHz would be needed. While this is a relatively high clock rate, it is in line with the clock speeds of current commercial processors.

Most of the increase in performance of the optimized Viterbi decoder is the result of distributing the Viterbi decoder workload evenly across 48 tiles, with a corresponding 48× improvement in performance. An additional improvement of approximately 17.5× is achieved through a reduction in the number of processor instructions. Some of this is due to avoiding the overhead of C infrastructure instructions resulting from things like function calls, but the large majority is due to the memory operations needed to read, write, and modify the memory structures used by the Viterbi decoder. The optimized Viterbi decoder avoids these instructions by using the tiled architecture's interconnection networks to achieve the same results with live data, studiously avoiding memory operations. In addition, the optimized Viterbi decoder enjoys a further 2.5× improvement in performance by avoiding the stalls associated with these additional instructions.

Integration of Optimized Functions

The multiple tile design was re-implemented utilizing the optimized functions, see Figure 7-17. The replacement functions, however, required many more tiles than were readily available on the hardware. Therefore, the optimized multiple tile design was implemented on the BTL simulator. The 802.11a specification and the previous implementations were again used to validate the implementation for correctness.

The optimized FFT and optimized Viterbi decoder implementations have both been parallelized to the largest extent possible on the Raw architecture. Further efforts to distibute either function across additional tiles would result in reduced performance. The optimized multiple tile design therefore achieves the highest performance available for this 802.11a baseband implemention. Additional increases in the achievable system data rate must be attained by other means.

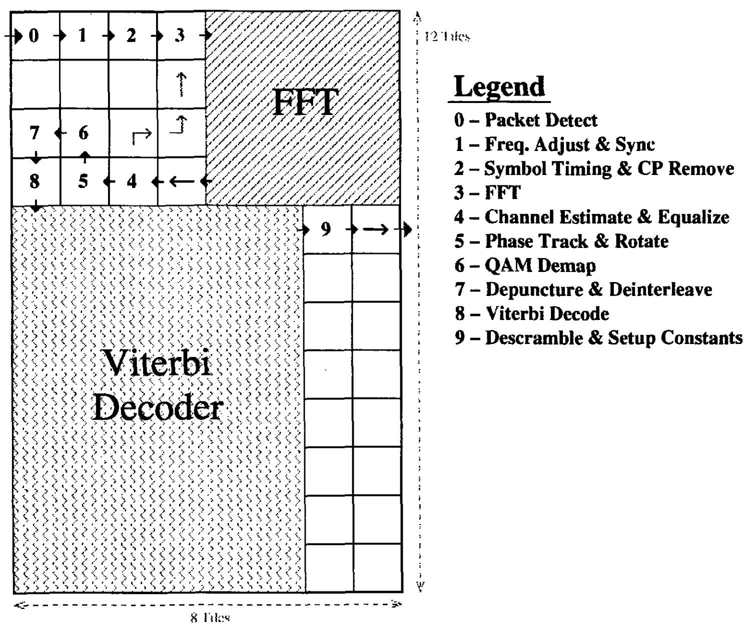


Figure 7-17: The mapping of the optimized multiple tile receiver onto Raw. The bottlenecked FFT and Viterbi decoder functions have been replaced by their spatially pipelined and optimized versions.

7.4.6 Optimized Multi-tile, Multipass Design

The optimized multiple tile baseband provided a starting point for a 2-pass multipass targeting the Raw platform. The multiple tile implementation was expanded to include a partial retransmit and second reception pass. A block diagram of the 2-pass multipass receiver can be seen in Figure 7-18. The single pass multiple tile optimized C/Assembly receiver blocks were reused. All that was required for the multipass receiver version was additional routing, a small bit of control functionality, and the partial retransmission and combining function blocks. These blocks were taken directly from the C language version of the transmitter.

The transmitter was augmented to operate on two passes. Functionality was added to read a second pass through the transmit chain (in addition to the first pass), to perform the combining of the two passes, and to generate the appropriate header symbol with the multipass information.

A small bit of additional control functionality was also added to read the slightly modified 2-pass headers and route the appropriate control information. For the multipass system, the dynamic network was used to distribute the control information, making this aspect easy to implement and allowing long distance communication of control values without impacting static network routing.

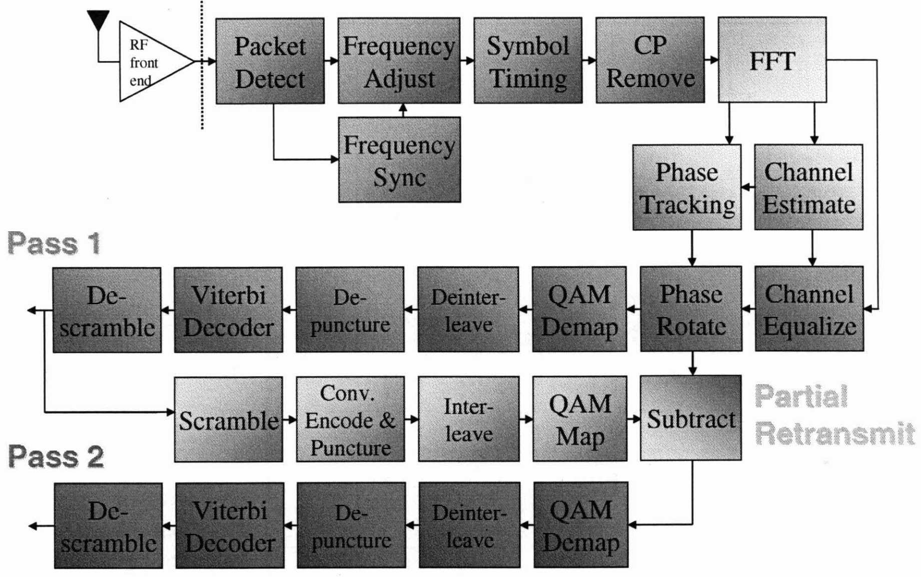


Figure 7-18: Multipass 802.11a receiver block diagram.

The full multipass implementation was again too large for the available hardware. The BTL simulator was used to simulate a sufficiently large Raw processor.

The mapping of the 2-pass multipass Raw implementation can be seen in Figure 7-19. The symbol stream is split into the two passes immediately preceding the demapping stage of Pass1 (block 6 in the figure). One copy proceeds to the Pass1 receive chain, while the second copy is injected into the static network. This second copy is buffered in the network of the tiles in between the split and the combining function block (the subtract block) to await the retransmission of the Pass1 output. Once the Pass1 reception is complete, its output is also split. One copy becomes the Data1$'$ output, while the other output is sent to the partial retransmit path. The retransmit path is simply the transmitter C functions ported to the Raw processor using the rgcc. The retransmitted output is then combined with the waiting received symbol stream and the result sent on to the Pass2 receive chain. The output of this receive chain then becomes Data2$'$. The functions used in Pass2 are exact copies of the functions used in Pass1. No new effort was required along this front.

The multipass design concept, as well as tiled architecture and the Raw processor itself, are predicated on the vision of future process generations where a huge number of tiles are available. Even using an estimated Raw processor design in a current process technology, however, the multipass optimized receiver achieves the goal of being compatible with communication at a 54Mbps data rate on a general-purpose processing substrate. 54Mbps

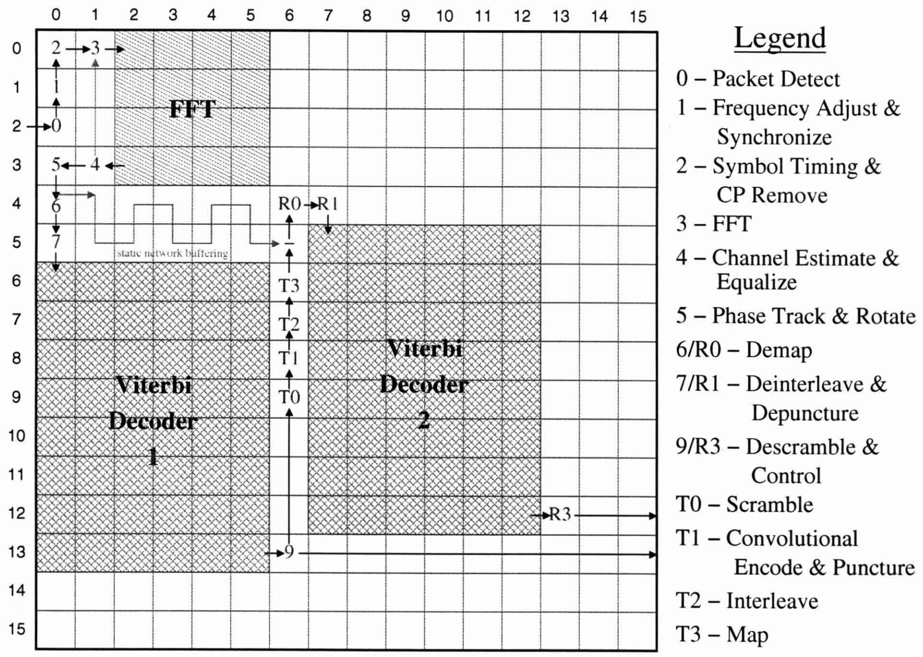

FFT
static network buffering
R0
R1
T3
T2
T1
T0
R3
9
Viterbi Decoder 1
Viterbi Decoder 2
Legend
0 – Packet Detect
1 – Frequency Adjust & Synchronize
2 – Symbol Timing & CP Remove
3 – FFT
4 – Channel Estimate & Equalize
5 – Phase Track & Rotate
6/R0 – Demap
7/R1 – Deinterleave & Depuncture
9/R3 – Descramble & Control
T0 – Scramble
T1 – Convolutional Encode & Puncture
T2 – Interleave
T3 – Map

Figure 7-19: Multipass receiver Raw mapping. Each block represents a Raw tile with its functions listed. Tiles without functions act purely as static network routers. The arrows represent data flow with black arrows as forward data flow on SN1, gray arrows as explicit backward control flow on the GDN (other control flow is not shown), and green arrows as forward data flow on SN2 which also acts as static network storage.

operation could be achieved in a 2-pass multipass system by using a 36Mbps 16-QAM pass plus an 18Mbps QPSK pass. This generates a combined constellation similar to 64-QAM, the modulation scheme used by the 54Mbps data rate. Such a multipass system would need to be able to run at the processing rate of the higher of the two passes. The critical bottleneck for the system is the Viterbi decoder, as the data rate does not effect the FFT stage. As such, the processor would need to be able to operate at a speed compatible with a 36Mbps Viterbi decoder. The optimized Viterbi decoder requires 70 processing cycles per output:

$$70\frac{\text{cycles}}{\text{outputbit}} \times 36\text{Mbps} = 2.52\text{GHz} \tag{7.6}$$

Therefore, a Raw processor with a minimum clock speed of 2.52GHz would be required. However, there is also a requirement to have sufficient parallel resources to implement the multipass system. The 2-pass multipass system of Figure 7-19 requires at least 141 active tiles to operate. Therefore, a second requirement of more than 141 tiles is imposed. Both of these requirements are met by the technology scaling estimates for the full custom version of Raw implemented in a 90nm process, as shown in Table 6.1. The full custom 90nm Raw is estimated to operate at 2.55GHz and to have from 144 to 225 tiles depending on the amount of SRAM in each tile. Either of these speculative processors would meet the minimum requirements for a 54Mbps multipass Raw implementation.

A 3-pass multipass system using the full 90nm Raw with smaller SRAMs would meet the requirements even more handily. Three 18Mbps QPSK passes could be used. The combination of passes would again resemble the 64-QAM constellation of 54Mbps. Combining two QPSK passes looks like a 16-QAM constellation, then adding in an additional third QPSK pass grows that to a 64-QAM constellation. The Viterbi decoder still operates as the bottleneck at 70 cycles per output:

$$70\frac{\text{cycles}}{\text{outputbit}} \times 18\text{Mbps} = 1.26\text{GHz} \tag{7.7}$$

This is well below the 2.55GHz estimated clock speed for such a processor. While the minimum tile requirement would be the 141 active tiles for the first two passes, plus an additional 56 active tiles for the third pass, this is again below the 225 tile estimate for the full custom 90nm Raw processor.

7.4.7 Simulated Optimized Multi-tile, Multipass Design

The BTL simulator, while a very accurate representation of the Raw hardware, proved to be exceptionally slow. In order to speed up simulation times, C language versions of the algorithms and structure used for the multipass design were implemented. This C-only version of the multipass system was verified against the BTL version using mixed C and Assembly language. The BTL multipass system provided a baseline of operation and was compared with the C-only version for correctness. The much faster C-only version was used to generate the BER curves in Chapter 8, however, as the running time required for the BTL versions would have been prohibitive.

Chapter 8

Results and Analysis

Multipass systems allow high data rate communications processing to be performed using general-purpose processing elements. Such systems bring the flexibility and ease of implementation provided by the general-purpose processing realm to communication systems. The usefulness of the multipass approach, however, depends on the ability of multipass systems to deliver communications performance comparable to a similar single pass communication system.

The performance of a communication system is a measure of how accurately the system transfers information under inclement conditions. The Bit Error Rate (BER) is a typical measurement of communication system performance. The BER is a ratio of the number of erroneous output bits (bits generated by a communication system which differ from the bits input to the transmitter) divided by the total number of output bits. BER curves are graphs plotting the system BER under different relative amounts of noise power. Such graphs provide a visual representation of how robust a communication system is to the impact of noise and, hence, how well the system performs.

The multipass implementations discussed previously provide a means of testing the performance of one type of multipass system relative to an equivalent non-multipass (or single pass) system. This begs the question of how well such a multipass system performs. Testing can help determine whether and how much performance degradation might be incurred by using a multipass system instead of a single pass one. Under a range of operating conditions, does a multipass system provide equivalent functionality to an comparable data rate single pass system? In what ways do the use of multiple passes impact the communication system performance and cause it to differ from the single pass case? Are there ways to mitigate any negative effects of multipass and/or take advantage of the multipass structure to improve performance? What performance trade-offs and design choices are enabled by using a multipass system? Performance testing of the superposition combining function multipass system was undertaken to address these questions. An analysis of these results appears in the following sections.

The multipass system tested here is a C language simulation of the optimized mixed C and Raw Assembly language multipass system which targets the Raw platform, as described in Section 7.4.6. The C language simulation executes on a Pentium 4 running Red Hat Linux 7.2. This C-only simulation uses the same algorithms as the mixed C and Assembly version and mimics the data communication patterns of the Raw version. A sufficiently large Raw hardware platform is not currently available and so the BTL cycle-accurate simulator must be used for the Raw implementation instead. The C language simulation was undertaken to compensate for the relatively long simulation times of the BTL simulator implementation. The C-only simulation was used for testing purposes as the running time for each simulation run was orders of magnitude faster than an equivalent C and Assembly simulation on BTL. The C language simulation output was verified against representative BTL simulations, however, to ensure congruence between the results.

8.1 Testing Methodology

The system BERs were determined by communicating multiple data packets and comparing the system output to the system input. The BER was found and plotted for a number of signal-to-noise ratio (SNR) values. E_b/N_0, the ratio of the energy per data bit transmitted to the noise power, was used as the SNR metric. The BERs for E_b/N_0 values from 0dB to 20dB were plotted. The range 0 to 20 was chosen to be inclusive: 0dB represents equal signal and noise power, a case where correct reception is unlikely; while 20dB represents 100$\times$ more signal than noise, making correct reception highly likely for the systems presented here. This allows the salient features of the BER curves to appear in the plots.

The test data for each SNR value consisted of 3200 packets of randomly generated data. The number of packets, 3200, was chosen to convey sufficient bits through the system to get meaningful BER results. The maximum payload length for each data packet is, as per the

802.11a specification, 4095 bytes (per pass). In order to generate results with BERs down to a reasonable error rate of 10^{-6}, at least one hundred times more than 1/(error rate) bits must be run through the system and, therefore, 3200 packets were used.

The test data was generated using the `ran1` function taken from [107]. The input bits were saved and compared against the system output bits to determine each run's BER. A new set of input bits was generated for each packet sent through the system. The total number of errors for all 3200 runs was compared against the total number of bits for all 3200 runs to find an overall BER.

The white noise source used in the system was generated as a zero mean, unit variance Gaussian. The various SNRs were implemented as attenuation on the Gaussian generator output [37]. The attenuation factor was determined by finding the signal power for each OFDM symbol and then dividing by the number of bits in that OFDM symbol to get E_b. For a desired value of $\mathrm{E}_b/\mathrm{N}_0$, in decibels (dB), the E_b value was divided by the inverse log of $\mathrm{E}_b/\mathrm{N}_0$ divided by 10 (the non-decibel ratio of $\mathrm{E}_b/\mathrm{N}_0$) to find the noise power, N_0. The noise power was then converted to an attenuation factor which was applied to the AWGN output. Thus, the AWGN magnitude is adjusted to create the desired SNR.

8.2 Pass2 Scaling

The Pass2 scaling values in the 2-pass multipass simulations were chosen to create combined signal space constellations identical to those of a comparable throughput single pass system. This was done for a number of reasons.

First, the QAM constellations which the multipass system mimics are an efficient signal space packing, both from the point of view of energy savings, as well as of minimizing the error probability. Hence, the popularity of the QAM constellations.

Second, while the scaling could be adjusted to favor one pass over another, that is not the case for the tests performed here, as the aggregate throughput is being observed, as well as the individual throughputs. The QAM constellations maximize the minimum distances between constellation points and thus balance the needs of both passes.

A smaller scaling factor would reduce the impact of Pass2 on Pass1, but negatively impact Pass2 reception. Since Pass2 appears as noise on the Pass1 signal, moving the Pass1 symbol away from its original location, a smaller Pass2 magnitude would help reduce the susceptibility of Pass1 to noise. A reduction in Pass2 power would at the same time, however, increase the susceptibility of Pass2 to noise. Less noise power would be required to move the Pass2 symbol across the decision boundaries between the Pass2 symbols based on the same Pass1 symbol value, see Figure 8-1.

A larger scaling factor would be detrimental to Pass1 and thus Pass2. The larger Pass2 power would move Pass1 symbol values farther away from their original location and make them more prone to error. This larger Pass2 power would have a negative impact on Pass2 as well, however. The increased Pass2 power would provide more spacing between Pass2 symbols based on the same Pass1 symbol, and thus reduce the likelihood of an error between such Pass2 symbols. The correct reception of Pass2, however, relies on the correct reception of Pass1. The increase in Pass1 error probability would translate into an increased Pass2 error probability and thus be a detriment to both passes, see Figure 8-2.

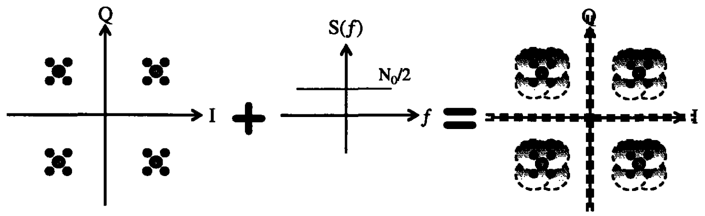


Figure 8-1: The effect of scaling down Pass2 by a large amount. Heavy scaling down of Pass2 leaves only a small amount of Pass2 power, reducing the likelihood of correct Pass2 reception. This scaling improves the likelihood of correct Pass1 reception, however, since the combined constellation points are close to their Pass1-only locations.

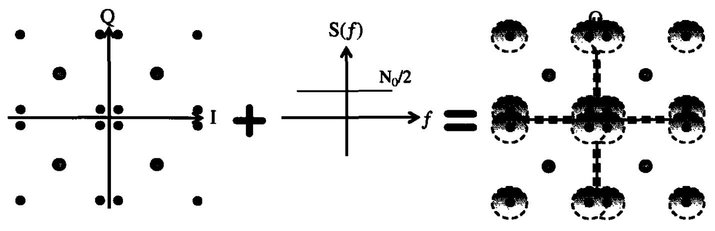


Figure 8-2: The effect of scaling down Pass2 by only a small amount. Light scaling down of Pass2 reduces the likelihood of correct Pass1 reception. As the correct reception of Pass2 is predicated on the Pass1 received data, this negatively impacts Pass2 reception as well.

Matching the QAM constellations, however, places the Pass2 constellation points in the middle as a compromise between these two scenarios. The Pass2 power is large enough to provide a reasonable likelihood of correct Pass2 reception, while being small enough to maintain an acceptable Pass1 probability of error.

Finally, by matching constellations with the single pass system, the average output power of the two systems also match. This provides a more accurate apples-to-apples comparison. If the multipass system were allowed to generate a larger output power for the same data throughput, it would skew the comparison of the two systems.

8.2.1 Asymmetric Passes

In a multipass system where both passes are of equal importance and multiple passes are used only to reduce the computational load, the goal is to choose a Pass2 scaling which balances maximizing Pass2 performance with minimizing Pass1 errors. The system discussed in Section 8.3 is such a system. It is possible, however, to prioritize some passes over others by adjusting the Pass2 scaling used. The relative power assigned to passes, and their corresponding likelihood of correct reception, can be asymmetrically skewed to favor one pass (or possibly some passes in a larger multipass system) over others. If the lower priority data is not successfully received, then appropriate action can be taken. Depending on the data involved, unsuccessfully transmitted lower priority data could be discarded, retransmitted, or retransmitted as higher priority information.

The Pass2 scaling value can easily be adjusted to reflect the relative priority of Pass1 over Pass2. A smaller Pass2 scaling will reduce the impact of Pass2 on Pass1 and improve Pass1's performance, at the expense of decreasing the performance of Pass2. This increases the likelihood of correct Pass1 reception at the expense of Pass2. This might be an acceptable trade-off in a situation where the Pass1 data is vital and the Pass2 data is useful, but not critical. An example of one such situation could be conveying images across the wireless link. Pass1 could be used to transmit the lower frequency data necessary to form the basic image, while less critical higher frequency details could be transmitted on Pass2. In a low SNR situation, the Pass2 data might be lost while the boosted Pass1 data is still available. This would result in the image details being lost, but the general image still coming through, allowing a graceful degradation of image performance with worsening channel conditions.

Another situation where asymmetric passes might be appropriate is one where two different types of information with different latency requirements, for instance speech and data, are being communicated in parallel. The speech information has a real-time requirement and might take priority over the transfer of less time-sensitive data (like a web page). Assigning the voice communication to Pass1 would help ensure that the speech data was correctly received without a need for retransmission. If the secondary data were lost, however, a later retransmission might suffice.

8.3 QPSK:QPSK 2-pass Multipass System

A QPSK:QPSK[1] 2-pass multipass system was used to evaluate multipass performance relative to single pass systems. The QPSK:QPSK multipass system has comparable cumulative throughput to a 16-QAM single pass system. Each QPSK pass provides a data rate of 12Mbps for a combined rate of 24Mbps. This is equivalent to the single pass 16-QAM system data rate of 24Mbps. The QPSK:QPSK multipass system is also scaled in such a fashion that the combined modulation constellation of the two passes is identical, in terms of both configuration and power, to a 16-QAM constellation. Therefore, it is natural to compare the QPSK:QPSK multipass system to a 16-QAM single pass system in order to evaluate the performance of such a system.

8.3.1 QPSK:QPSK Pass1

The first Pass of the QPSK:QPSK 2-pass multipass system can be thought of as a single pass QPSK system distorted by the addition of Pass2. The data rates, decision boundaries, and receive chains of Pass1 and a single pass QPSK system are the same. This suggests that a reasonable performance comparison would be between a single pass QPSK system and Pass1 of the multipass system. Such a comparison is presented in Figure 8-3.

From the point of view of Pass1, Pass2 looks like an additional noise source. The two data streams are independent and the purpose of the scrambling and interleave functions at the transmitter is to minimize correlations among the bits sent and thus help ensure this independence. As such, the Pass2 data stream appears as an additional random source and can be thought of as noise with one important additional property. The addition of the Pass2 "noise" power will never, on its own, cause an error in the Pass1 output. Only in conjunction with other noise can the Pass2 symbol power contribute to errors.

For large noise powers (small SNR) the impact of Pass2 on Pass1 is negligible as the noise power is greater than, or on the order of, the Pass2 power. As the noise power shrinks,

[1]QPSK:QPSK denotes Pass1 modulation:Pass2 modulation. In this case, the Pass1 modulation scheme is QPSK and the Pass2 modulation is QPSK. If it were QPSK:16-QAM, then the Pass1 modulation scheme would be QPSK and the Pass2 modulation scheme would be 16-QAM.

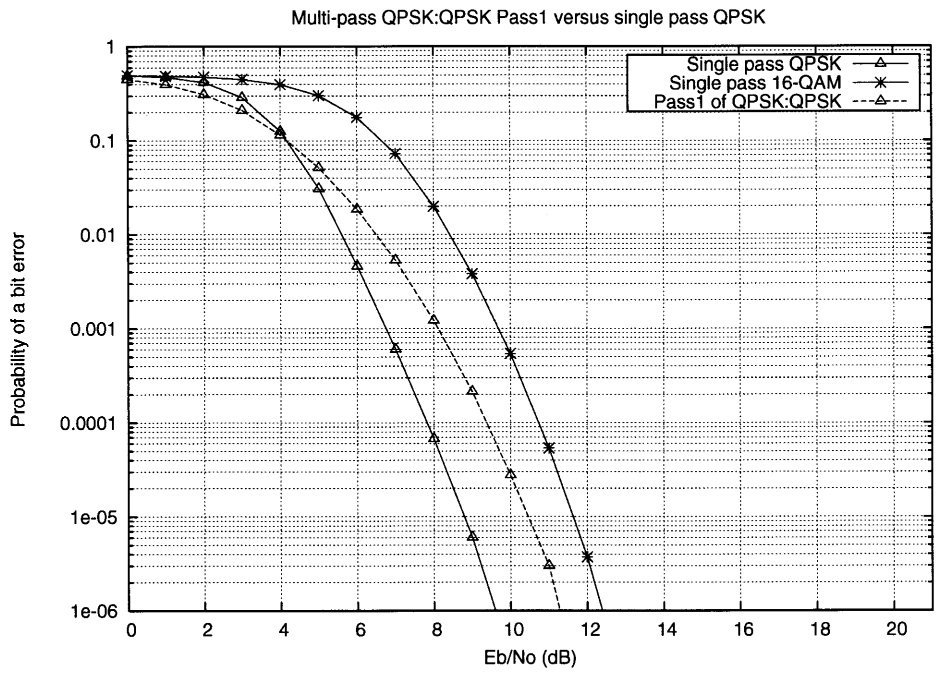


Figure 8-3: BER comparison between Pass1 of an optimized C simulation of the QPSK:QPSK 2-pass multipass system and the 802.11a single pass data rate (12Mbps) with the equivalent signal space constellation.

however, the impact of Pass2 becomes more prominent. For the low SNRs on the left side of Figure 8-3, the BER of the multipass system is similar to that of the single pass system. As the noise power decreases (larger SNR), however, the influence of Pass2 becomes more apparent and the Pass1 BER curve begins to pull away from the single pass curve.

At larger SNR values, the Pass1 BER begins to approach the Pass2 BER curve as the impact of Pass2 dominates the noise power. As the noise power is reduced, the Pass1 symbol locations begin to resemble those of a Pass2 constellation (16-QAM) plus noise, see Figure 8-4. This has the effect of positioning some constellation points closer to Pass1 decision boundaries, and the corresponding smaller noise power needed to move symbol values across those boundaries causes more decision errors. As the noise power falls, it reaches a level where the noise would be unlikely to cause an error if the Pass1 symbol value were at the QPSK location, but in those instances where Pass2 has moved the Pass1 symbol value closer to a decision boundary, an error still occurs.

Pass1 experiences BERs similar to QPSK for low SNRs and worse than QPSK for higher SNRs, but the Pass1 BER is still better than what would have been the case for 16-QAM. This is a result of the superposition combining function allocating more of the symbol power

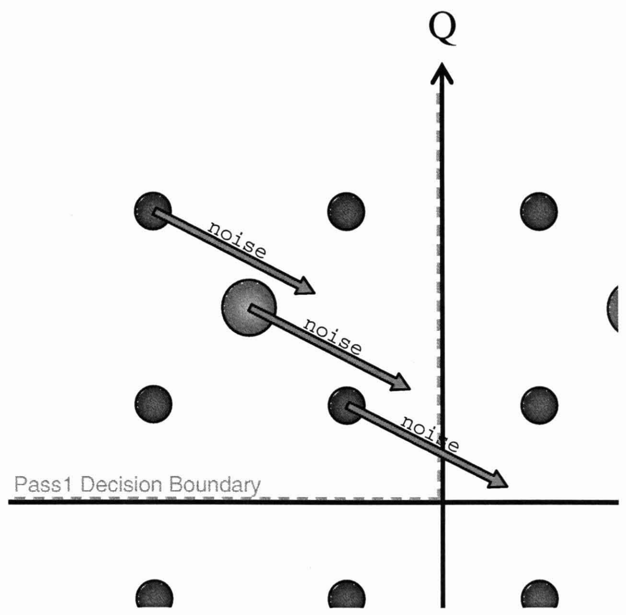


Figure 8-4: At higher SNRs the influence of Pass2 can cause an error in what would otherwise have been a successful demapping of Pass1.

to Pass1 and fits well with the concept of an asymmetric multipass communication system where earlier passes are prioritized and carry more critical information than later passes.

8.3.2 QPSK:QPSK Pass2

The second pass of the QPSK:QPSK 2-pass multipass system has the same signal space constellation positions as a 16-QAM single pass system. The logical choice for comparison with Pass2 performance is therefore the 16-QAM single pass system. Such a comparison is presented in Figure 8-5.

The BER curve of Pass2 of the multipass system follows the shape of that of the 16-QAM single pass system quite well. This is not surprising as the decision boundaries of Pass2 are equivalent to those of the 16-QAM system, while those of the multipass system are just broken into a two-stage decision instead of a single decision. The first stage, the demapping of Pass1, incorporates errors which cause the symbol values to cross quadrant boundaries. The second stage, the demapping of Pass2, encompasses those errors which cause symbol values to cross intra-quadrant decision boundaries. The scaling up of Pass2 at the receiver, following the removal of Pass1 (see Section 5.2.3), is a linear function which scales the noise along with the signal. This linear scaling preserves the relative impact of noise on the Pass2 signal. Consequently, the Pass2 symbol decision produces the same

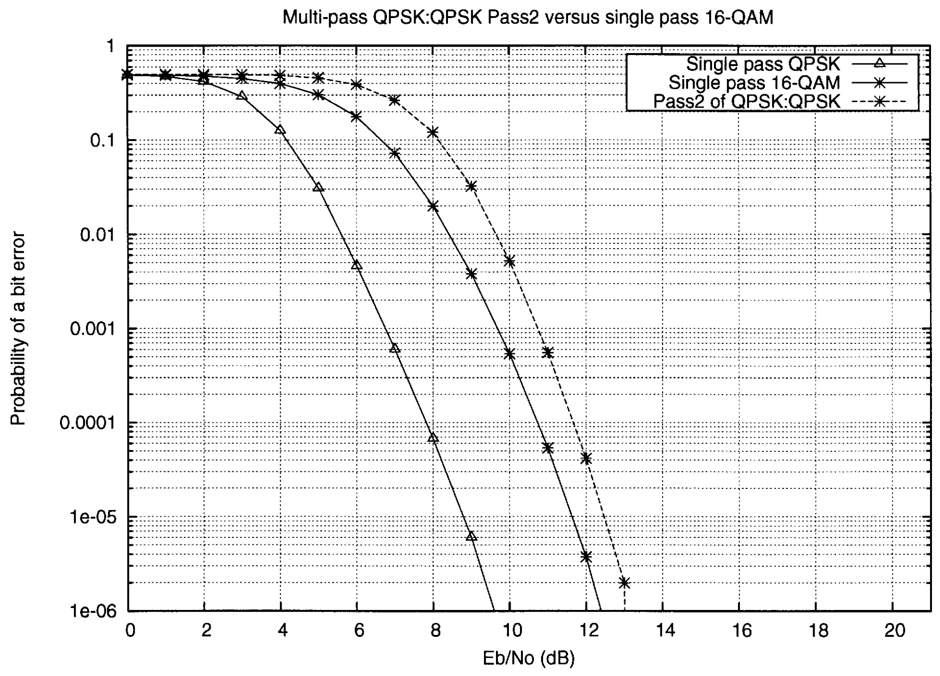


Figure 8-5: BER comparison between Pass2 of an optimized C simulation of the QPSK:QPSK 2-pass multipass system and the 802.11a single pass data rate (24Mbps) with the equivalent signal space constellation.

result as if the 16-QAM decision boundaries were applied to the still-combined multipass symbol in the quadrant corresponding to the Pass1 decision. Therefore, it is logical that the Pass2 result is similar to that of the 16-QAM. The Pass2 BER curve is slightly shifted to the right, however, denoting a decrease in performance of Pass2 over that of the single pass 16-QAM system. This is the result of an increased frequency of errors being presented to the Viterbi decoder of Pass2.

In a single pass system, such as the 16-QAM system, the two passes are demapped together and the combined results are presented to Viterbi decoder. Consequently, in the 16-QAM system, four bits of data are sent to the Viterbi decoder for each symbol received. The number of errors in this demapping, if any, are likely small for higher SNRs. This is a result of the Gray encoding of the mapping chosen for the 16-QAM system. The Gray encoding ensures that the Hamming distance is one between any two adjacent regions. Thus, an error which carries a symbol across a single decision boundary will incur only a single bit error.

In a multipass system, however, the bit demapping is broken into two stages, one for each pass. The two stages are not equivalent. For the combining function and Pass2 scaling used here, Pass1 retains the majority of the signal power and, as has been shown, has a correspondingly lower BER. Pass2, conversely, has a smaller share of the signal power and a significantly larger BER. If an error were to occur in symbol space demapping, the bits demapped by Pass1 are those which are least likely to have changed. Errors are much more likely to have occurred in the Pass2 demapping. As compared to the 16-QAM system, an error which would cause a single bit error in the 16-QAM demapping is much more likely to appear as an error in the Pass2 demapping. The larger likelihood of error at the output of the Pass2 demapping stage is augmented by the fact that each pass produces half the number of output bits than the 16-QAM system for each demapping. This translates into an increased Pass2 demapping error rate which impairs the ability of the Pass2 Viterbi decoder to mitigate the errors encountered.

The Viterbi decoder uses state information to infer when an error has occurred and overcome it. The ability of the decoder to recover from errors is related to the quality of the state information available. A higher rate of errors results in a larger amount of corruption of the state information and reduces the ability of the Viterbi decoder to recover from future errors. In spite of the increased errors on Pass2, the performance of Pass2 is within 1dB of that of the 16-QAM system.

8.3.3 QPSK:QPSK Cumulative Throughput

The overall performance of a 2-pass multipass system can be determined by averaging the BERs of the two passes. A multipass system may be used as a means of distributing the processing load of a communication system. Multiple lower processing rate passes are used instead of a single higher rate pass. In this case, the same set of information must be communicated as in the single pass case, but that information is just distributed across the passes. For this type of system, the metric of performance is the *cumulative* bit error rate across both passes. The individual BERs of each pass is less important than the total number of bits correctly sent.

The data rate for each pass in the QPSK:QPSK 2-pass multipass system is 12Mbps. This is the equivalent of the 24Mbps 16-QAM single pass system in terms of overall data throughput. A graph of the cumulative throughput BER as compared to the 16-QAM BER is presented in Figure 8-6. For very low SNRs, the cumulative BER is actually better than that of the 16-QAM BER. This is a result of the good performance of Pass1 at low E_b/N_0 values. For higher SNRs, the cumulative BER is worse than, but close to, the 16-QAM BER. This is caused by the decreased performance of Pass2 at these higher SNRs relative to the 16-QAM single pass system. The cumulative BER is improved by the performance of Pass1, but not enough to mitigate the errors on Pass2. In spite of this, the cumulative BER performance is within approximately 0.5dB of the BER performance of the 16-QAM single pass system.

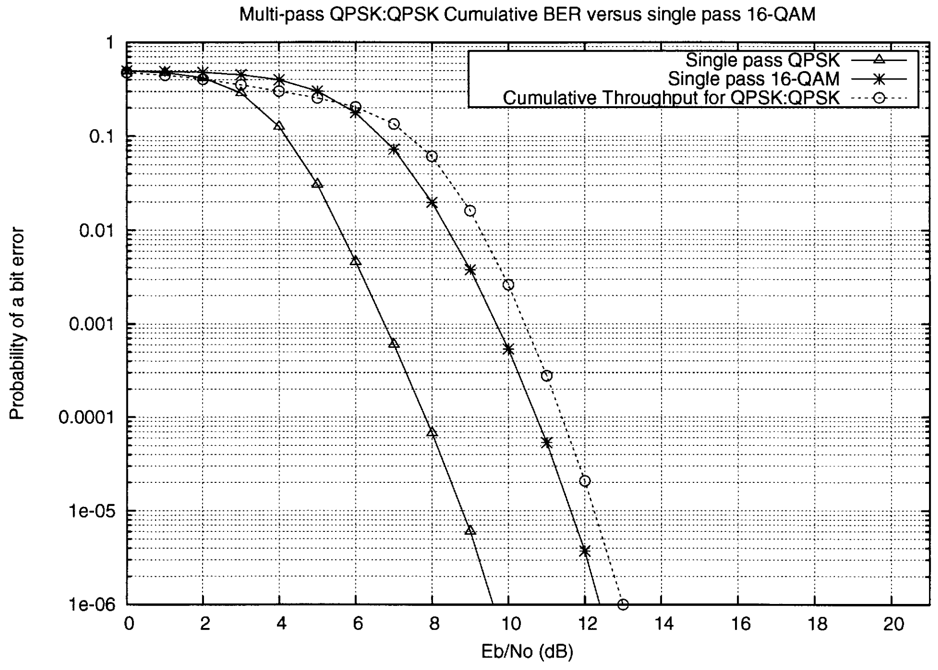


Figure 8-6: BER comparison between the cumulative throughput of both passes of an optimized C simulation of the QPSK:QPSK 2-pass multipass system and the 802.11a single pass data rate (24Mbps) with the equivalent signal space constellation.

8.3.4 QPSK:QPSK Multipass

The performance of the individual passes and cumulative results are put into context in Figure 8-7. Pass1 performs worse than the QPSK system, but its performance is still much better than that of the 16-QAM system. The Pass2 and cumulative performances are slightly less than that of the 16-QAM system, but are still reasonably close to single pass performance. This performance is quite good, given that the use of a 2-pass multipass system may make a 24Mbps system feasible in a situation where the 16-QAM system is not.

The multipass approach provides the flexibility, the ease-of-implementation, and the adaptability of a communication system enabled by operating the general-purpose process-

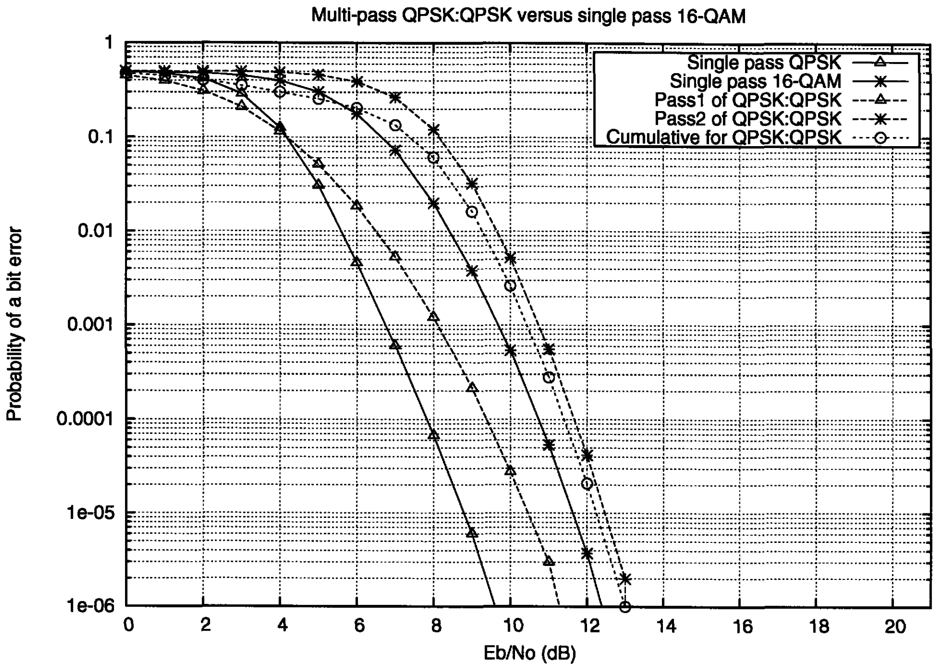


Figure 8-7: All curves of the BER comparison between an optimized C simulation of the QPSK:QPSK 2-pass multipass system and the equivalent 802.11a single pass systems.

ing domain. In some cases, the feasibility of operating in this domain is reliant on the use of a multipass approach. In addition, the close to single pass BER performance of the multipass system demonstrates that the system provides justifiable performance.

8.4 Scaled QPSK:QPSK Performance

The relatively high error rate exhibited by Pass2 of the QPSK:QPSK multipass system keeps the overall system from quite achieving single pass performance. If the errors on Pass2 could be reduced, however, while not incurring too many errors on Pass1, the overall performance of the multipass system would more closely match that of the single pass system.

8.4.1 Scaling Revisited

One way to improve the performance of Pass2 is to allocate a larger share of the signal power. How can this be done? A change in the scaling used in the superposition combining function could achieve a relative increase in Pass2 power. The scaling referred to in Section 8.2 allows too many errors on Pass2 relative to Pass1. A more equitable distribution is needed.

The original Pass2 scaling value used attempted to create a combined multipass symbol constellation which mimicked that of the 16-QAM system. The output of this system needed to be normalized, however, to ensure the same average output power across all systems. This was done to create equivalent comparisons across the systems. A consequence of this normalization was to move the relative location of the multipass symbol points and distort the signal constellation, resulting in a distribution of power favoring Pass1 over Pass2. The scaling must be readjusted, therefore, to take this normalization into account. Doing so results in a more equitable distribution of power and corresponding similarity in BER.

Equation 8.1 represents the desired relationship between the Pass1 and Pass2 final symbol locations necessary to match the 16-QAM constellation. Equation 8.2 states in a reduced form the relationship between the multipass constellation point locations and the average symbol power. For all the systems discussed here, this average power was normalized to $\overline{Power} = 1$.

$$P_1 - SP_2 = \frac{NP_1}{2} \tag{8.1}$$

$$(NK_{mod})^2[(P_1 + SP_2)^2 + (P_1 - SP_2)^2] = \overline{Power} \tag{8.2}$$

Where P_1 and P_2 are the magnitudes of the Pass1 and Pass2 symbol values, K_{mod} is the initial QPSK power normalization of $\frac{1}{\sqrt{2}}$ applied to Pass1 as per the 802.11a specification, N is an additional power normalization applied to the multipass power output necessitated by the addition of Pass2 to Pass1, and S is the combining function Pass2 scaling. Let $P_1 = P_2 = 1$, as the constellation of each pass before scaling and power normalization resembles that of QPSK. Substituting in the known values for P_1, P_2, and K_{mod} and reducing results in a more concise set of equations,

$$N = 2(1 - S) \tag{8.3}$$

$$S^2 = \frac{1}{N^2 - 1} \tag{8.4}$$

Equations 8.1 and 8.2 form a system of equations which can be solved to find for S and N. This yields a larger Pass2 scaling value than used previously.

A larger scaling applied to Pass2 provides a greater distance between Pass2 symbol values in the same quadrant. This translates into a larger probability that once a Pass1 quadrant has been isolated, the Pass2 symbol therein will be correctly demapped. Hence, an increased scaling helps to improve the Pass2 error rate if the symbol decision for Pass1 is correct. The issue, though, is that an increased Pass2 scaling increases the likelihood of an incorrect Pass1 symbol decision. What is needed to reap the benefit to Pass2 reception of an increased scaling is a means of helping to ensure correct Pass2 symbol decisions even if the Pass1 decision is in error. Applying a Gray encoding to the Pass2 symbols provides this means.

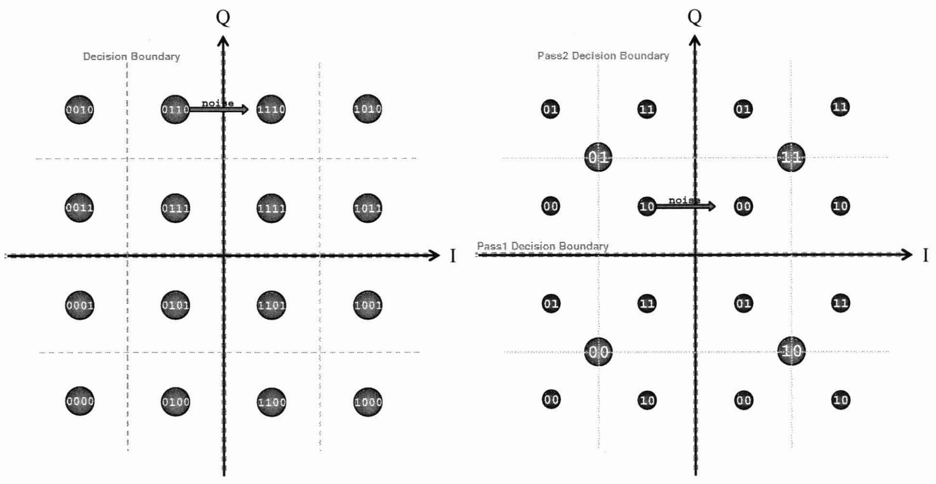
8.4.2 Gray Encoding

In most communication systems, the bits mapped to each constellation point in signal space are Gray encoded. Gray encoding is a way of structuring the mapping of bits to constellation points such that the Hamming distance, or number of differing bits, between adjacent constellation points is always 1. This is done to minimize the impact of errors. If the noise were to push a received symbol value across a decision boundary, it most likely moves the received symbol into an adjacent decision, or Vernoi, region. Noise of a large magnitude would be required to move a symbol value into a non-adjacent region, and even then Gray encoding ensures that the number of bit errors only grows as the number of regions crossed. If the Hamming distance between adjacent regions is minimized, then the number of bit errors incurred by a noise event which causes a symbol demapping error is also minimized. As the number of demapping mistakes decreases, the rate of bit errors decreases, and the likelihood that those fewer bits in error will be corrected by the Viterbi decoder increases.

The mapping of bits to constellation points used in the multipass system, as it was initially presented, breaks this Gray encoding property. While the bit assignment within each pass is Gray encoded, the Gray encoding property is not migrated across passes. When the combining function is applied to the passes in the original system, the Hamming distance for errors which cross both Pass1 and, necessarily, Pass2 decision boundaries sums and the result is a combined Hamming distance of more than 1, see Figure 8-8.

In order to address this problem for the superposition combining function, the two passes must be more tightly coupled. If the Pass1 values were to influence the bit encodings applied to symbol values for Pass2, then the bit encodings for the combined Pass1 and Pass2 constellation could be made Gray encoding compatible, see Figure 8-9.

With Gray encoding applied, an error in the Pass1 symbol decision does not automati-



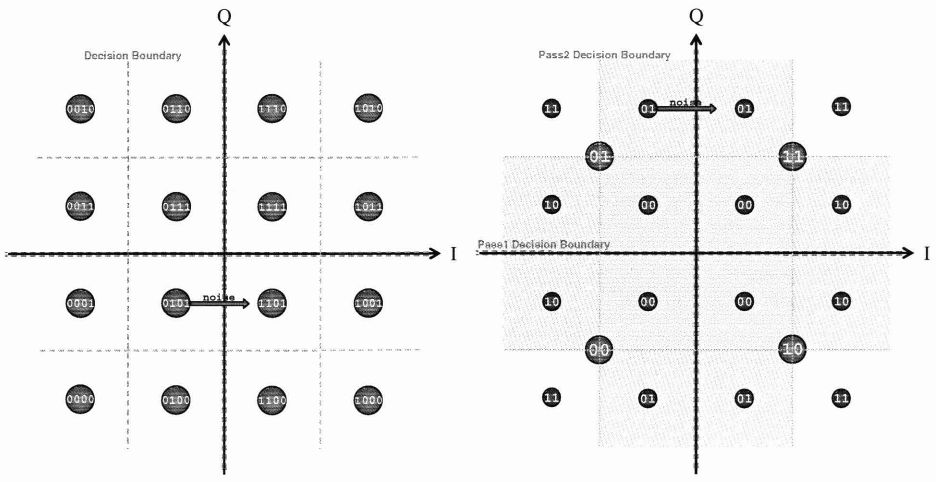
802.11a Specification Single Pass

Noisy Output:	1110
Correct Output:	0110
Hamming Distance:	1

QPSK:QPSK Multi-Pass

Pass1	Pass2	Cumulative
11	00	1101
01	10	0110
1	1	2

Figure 8-8: The bits assigned to constellation points in the QPSK:QPSK multipass case and the 16-QAM single pass case. The Hamming distances of the multipass and single pass systems are not equivalent.



802.11a Specification Single Pass

Noisy Output:	1101
Correct Output:	0101
Hamming Distance:	1

QPSK:QPSK Multi-Pass

Pass1	Pass2	Cumulative
11	01	1101
01	01	0101
1	0	1

Figure 8-9: Adjusting the Pass2 bit mapping based on the Pass1 symbol value results in a fully Gray encoded combined constellation. The highlighted regions show how Gray encoding creates redundancy of the Pass2 values across Pass1 decision boundaries. This helps insulate Pass2 symbol decisions from Pass1 symbol decisions.

cally translate into an error in the Pass2 decision. The Pass2 Vernoi regions have effectively been extended for those regions along the Pass1 decision boundaries. This extension of Pass2 regions helps insulate Pass2 symbol decisions from Pass1 symbol errors.

8.4.3 Scaled QPSK:QPSK Multipass

The BER performance of the scaled QPSK:QPSK multipass system can be seen in the BER curves below. The increased Pass2 scaling provides a more equitable distribution of power between the two passes. This trade-off can be seen by the decreased performance of Pass1 (see Figure 8-10) as compared to the increased performance of Pass2. The impact of Pass1 errors on Pass2 is minimized by the application of Gray encoding to Pass2. This results in a decreased impact of the lowered Pass1 performance on Pass2 (see Figure 8-11).

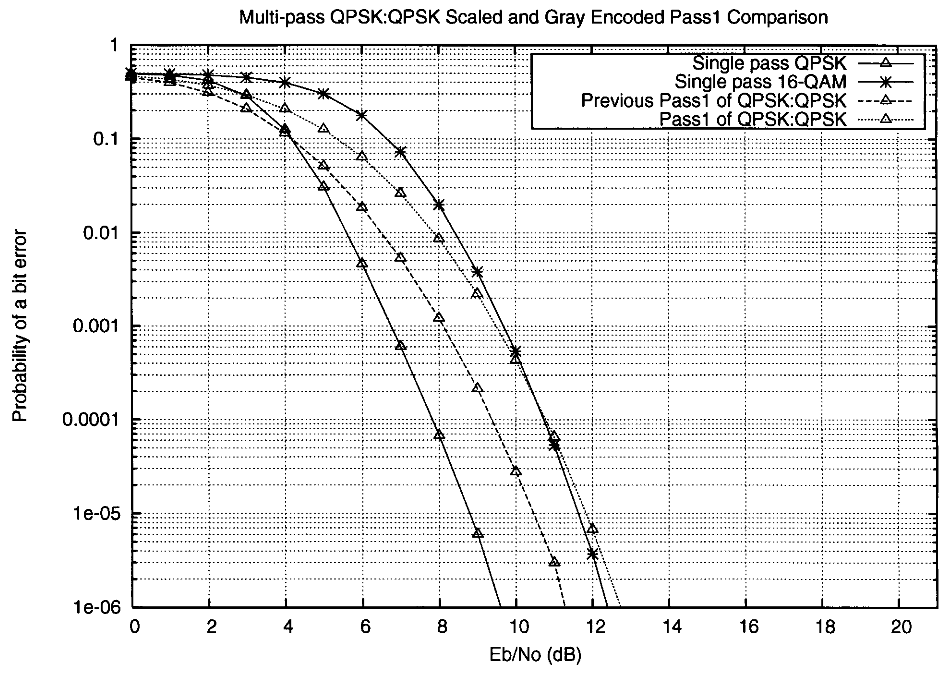


Figure 8-10: The application of Gray encoding coupled with a new larger Pass2 scaling for the optimized C simulation of the QPSK:QPSK multipass system makes a more equitable trade-off between the power and performance between Pass1 and Pass2. The increased scaling of Pass2 reduces the performance of Pass1 over that of the original QPSK:QPSK system.

The BER curve for the Gray encoded and scaled Pass2 remains slightly higher than that of the single pass 16-QAM system even though the Gray encoding helps insulate Pass2 bit decisions from Pass1 decision errors. This is due to the fact that the multipass system

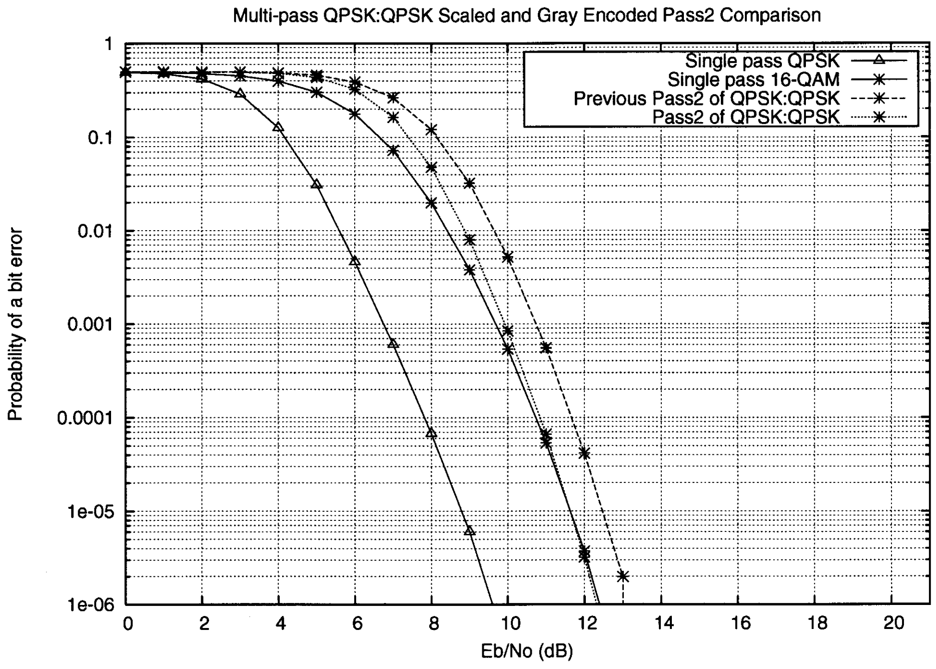


Figure 8-11: The application of Gray encoding coupled with a new larger Pass2 scaling for the optimized C simulation of the QPSK:QPSK multipass system makes a more equitable trade-off between the power and performance between Pass1 and Pass2. The increased scaling of Pass2 increases the performance of Pass2 over that of the original QPSK:QPSK system.

Pass2 still has fewer correct bits couching each Pass2 error as the QPSK Pass2 has only two bits per symbol decision, as opposed to the four bits per decision in the single-pass 16-QAM system.

For lower SNRs, while the total number of errors might be similar to that of the single pass case, the rate of errors (which is what matters for the Viterbi decoder's ability to error correct) is higher. As the SNR increases, however, the lower number of overall multipass Pass2 decision errors begins to have an effect. As long as errors are relatively infrequent, then the higher relative frequency of the Gray encoded and scaled multipass Pass2 system decision errors is dominated by the lower number of overall errors.

The effect of the change in the combining function scaling is readily apparent in the Gray encoded and scaled multipass system Pass1 BER curve. The shift of power to Pass2 through the scaling adjustment increases the BER of Pass1. The Pass1 curve now moves beyond the QPSK single pass BER curve and approaches the 16-QAM single pass curve as the SNR increases. This is a result of the larger Pass2 scaling moving the multipass symbol locations closer to the Pass1 decision boundaries, with the predictable result that less noise power is needed to cause Pass1 decision errors. Only the fact that the Pass1 QPSK decision boundaries are along the quadrant axes, as opposed to the smaller 16-QAM Vernoi regions, allows Pass1 to slightly outperform the 16-QAM single pass case, even with the less evenly space signal space locations. In other words, a smaller noise power is required at this scaling to push those scaled multipass symbols which are close to a Pass1 decision boundary across that boundary and cause a decision error. But this is overshadowed by the fact that in a majority of cases, the effect of the AWGN will be either to move the symbol location farther away from the Pass1 decision boundary *or* to move it towards a boundary which is not nearby, see Figure 8-12.

The net result is a cumulative BER which is much closer to that of the 16-QAM single pass BER (see Figure 8-13). The trade-off of Pass1 and Pass2 performance through scaling and Gray encoding pushes both passes towards convergence at the cumulative BER (see Figure 8-14). The performance of the scaled and Gray encoded 2-pass QPSK:QPSK multipass system is almost identical to that of the single pass 16-QAM system. Therefore, the multipass system gains the benefits of a general-purpose processing environment, as well as the opportunity for reduced processing rates, while achieving the same data rates as and similar performance to that of a non-multipass system.

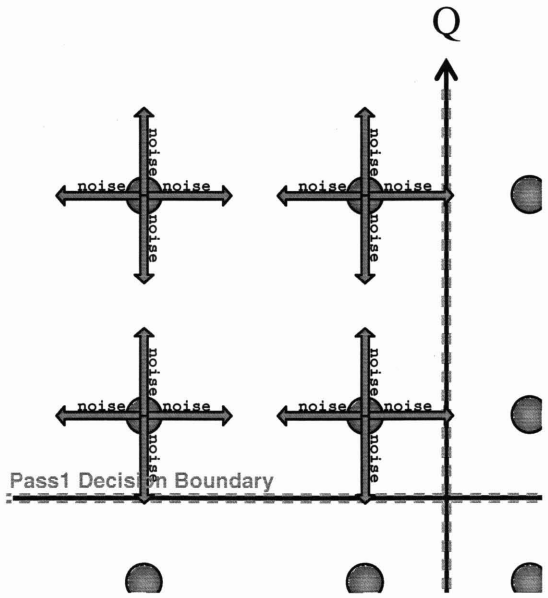


Figure 8-12: The potential effects of noise on Pass1 in the Gray encoded and scaled QPSK:QPSK system. In the Gray encoded and scaled QPSK:QPSK system, Pass2 has more power, which results in the combined multipass symbols being moved closer to the Pass1 decision boundaries. In spite of this, in a majority of cases, the AWGN noise still fails to move the symbol locations across Pass1 decision boundaries. The arrows show the possible effects of noise on I and Q individually for each of the potential signal locations. Only in 4 of the 16 scenarios would the addition of noise be likely to cause a Pass1 decision error.

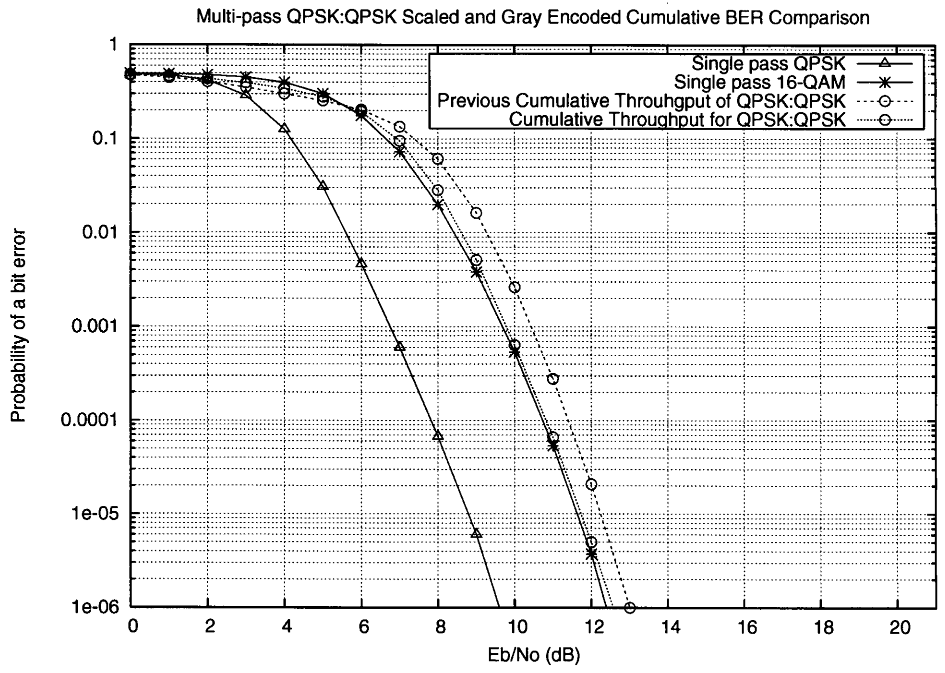


Figure 8-13: The application of Gray encoding coupled with a new larger Pass2 scaling for the optimized C simulation of the QPSK:QPSK multipass system makes a more equitable trade-off between the power and performance between Pass1 and Pass2. The net trade-off between the Pass1 and Pass2 performances is an improved cumulative BER over that of the original QPSK:QPSK system.

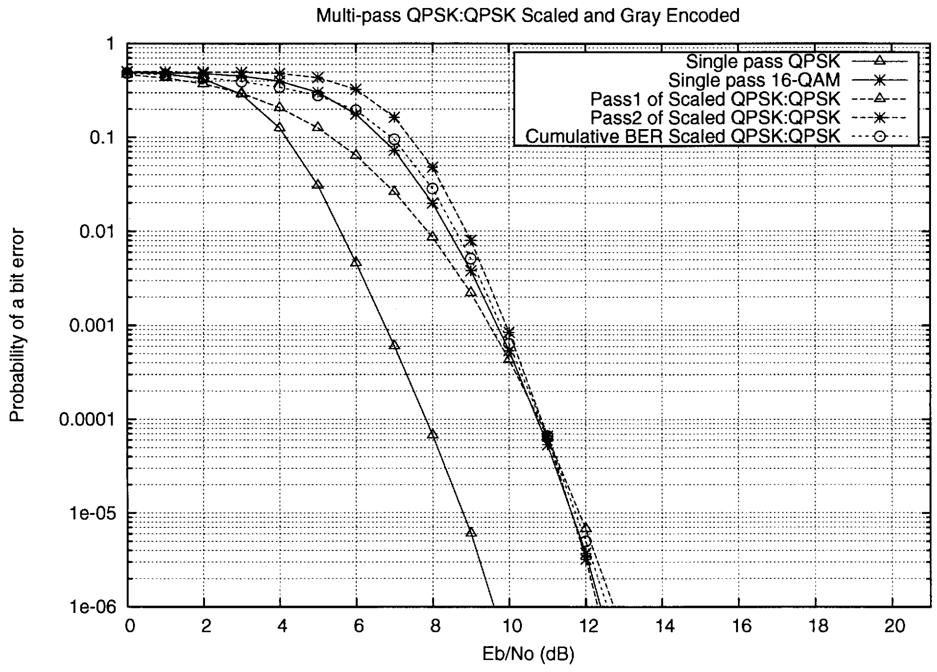


Figure 8-14: An overview of the performance of the optimized C simulation of the QPSK:QPSK multipass system using Gray encoding and with a new larger scaling applied. The single pass QPSK and 16-QAM BERs are provided for reference. The increased scaling of Pass2 reduces the performance of Pass1 while improving that of Pass2. The net effect is an improved overall performance as can be seen by the minimal distance between the cumulative BER and the 16-QAM single pass BER.

8.5 Summary

The multipass system presented here was shown to achieve performance similar to that of an equivalent data rate single pass system. This demonstrates that it is possible to implement a communication system of a desired data rate using a multipass approach without suffering a reduction in BER performance. By allocating power evenly among the passes, a multipass system can be implemented which achieves a cumulative BER which is comparable to that of a single pass system at the cumulative data rate.

The multipass approach provides other opportunities, however, by allowing the system to individually trade-off the performance of the various passes. By tailoring the scaling used in the superposition combining function a multipass system can change the allocation of power among the various passes to favor some over others. This results in a lower BER on the higher power passes at the expense of the BER of the other passes.

Chapter 9

Conclusions

The multipass approach successfully makes available general-purpose processing for communication applications. The multipass approach is capable of achieving performance on par with the equivalent throughput single pass system while providing the flexibility to dynamically adjust the number and type of passes used. As systems move to multiple-core processors, especially embedded processors in handheld devices, multipass enables the system to fold the baseband processing into the already present general processing resources. This provides the opportunity to use a multipass approach to remove the need for specialized communication processing components, as well as allows communication processing to be managed, in terms of scheduling and allocation of resources, in conjunction with the rest of the system tasks.

The system presented here, utilizing a superposition combining function, has been shown to be capable of achieving bit-error-rates comparable to that of a single pass system. This demonstrates that a multipass system is feasible from a communication performance perspective and is implementable on a current process technology tiled architecture. A general-purpose processing architecture can therefore be used for high data rate communication baseband processing, a feat which is unattainable using current GPPs and a single pass approach.

Specifically, it has been shown that with current and future process technologies, a multipass system based on the Raw tiled architecture is sufficient to implement an 802.11a 54Mbps baseband. Such a system requires an implementation of the current Raw architecture in a process technology of $90nm$ or smaller. In addition, it is necessary to use an optimized implementation, particularly for bottleneck prone functions. Such a system is consistent with 54Mbps operation, however, which is not the case for current generation GPPs and is not likely for any uniprocessor.

The necessary system hardware components (including the Raw processor, the Raw Handheld motherboard, and the Wireless Board) were built to explore the implementation requirements and to show the feasibility of an actual system. While not utilized in the BER testing, the hardware confirms the plausibility of the system and provides additional confidence in the results.

A number of other aspects of multipass became apparent through this work. Among them are: the impact of the specific combining function used and the need to specialize the system accordingly; the necessity of intra-function parallelism, in conjunction with inter-function and system-level parallelism; the specific features of tiled architectures which help to promote the multipass approach; and the additional system characteristics which become available and can then be explicitly controlled when using a multipass approach.

In order for the system presented here to achieve single pass performance using multiple passes, it was necessary to specialize the system to take advantage of and adapt to the combining function used. For the superposition combining function, this entailed modifying the combining function to preserve the Gray encoding of the bits across passes, as well as adjusting the scaling used in the combining function itself to balance the power distribution across passes. These combining function specific modifications were crucial to optimizing the performance of the multipass system.

The use of multiple passes in the system reduced the data rate requirements for any given pass, but successful system operation is also predicated on the optimization of the functional blocks themselves in order to achieve even these lower data rates. The multipass approach allows the system to trade the use of additional resources for lower data rate processing. This reduces the stress on the system of performing bottleneck functions and makes the operation of these functions possible. These bottleneck functions still require significant processing resources (hence the need to reduce their processing requirements) and parallelizing them across multiple processing cores can help to reduce the load on any given core. The exact trade-offs between the clock speed necessary to operate at a given data rate, the amount of processing resources allocated to intra-function parallelism (reducing the clock speed requirements of the bottleneck functions for a given data rate), and the amount of processing resources allocated to increasing the number of passes (reducing the data rate requirement of each pass in order to reduce the bottleneck function requirements) are a function of the specific implementation parameters and the architectural platform.

For the system built here, a tiled architecture, the Raw architecture, was used. A tiled architecture provides a number of features that are particularly useful for communication systems and the multipass approach. Among them are: the ability to readily accommodate streams of data through the use of high-bandwidth I/O; the availability of multiple tightly-coupled processing cores across which to distribute even intra-function spatial pipelines; and the ability to minimize the overhead of memory interactions by communicating values between processing cores through the use of low-latency networks. Using the Raw processor and leveraging the features of tiled architectures enabled a superlinear speedup in critical path communication functions. The FFT function experienced a $1,000\times$ speedup using 16-tiles, while the Viterbi decoder function, the primary system bottleneck, experienced a $10,000\times$ speedup using 48-tiles. While these attributes of tiled architectures were very helpful for the system presented here, depending on the desired communication application and its requirements and the specific architecture features of the platform used, the multipass approach could be utilized on other multiple processing element systems (e.g., CMP, DSPs).

Splitting up the communication processing into multiple passes allows extra degrees of freedom for the system which can be managed by the user or system. One new parameter available to the system is the allocation of power among the various passes used. The reliability of passes can be traded-off to allow prioritization of passes. Another degree of freedom is the trade-off between communication and computation. This provides more flexibility to the system with the ability to use just enough resources while operating at just a high enough clock rate to get the job done. This enables potential savings in power, cost, space, and/or time.

Finally, the use of the multipass algorithm allows a trade-off between area (in the form of resources consumed) and processing rate (in the form of the required minimum clock rate). This trade-off allows additional area to be used to reduce the processing rate. The resulting lower clock rate can be used to make viable a communication system platform which would otherwise be too slow (as discussed above), be used to improve the ease of implementation and time to market for the system design by relaxing circuit timing constraints, or be used to reduce the system power consumption through the use of voltage scaling or lower V_t devices.

Chapter 10

Related Work

The multipass system presented here touches on multiple different areas of research and related work. To our knowledge, multipass is a new concept and, consequently, directly related work does not exist. There are, however, multiple of areas of work which share conceptual roots with multipass or are enablers for the multipass approach. The multipass approach to communication, as a means of orthogonalizing for disambiguation in order to successfully transmit multiple data streams in parallel, is similar in nature to approaches used in other types of systems. The use of software on a general-purpose system as a means to perform communication processing relates to other software radio work. And finally, while the specific hardware implementation platform used in this case was the Raw tiled architecture, the multipass concept is applicable to other research tiled architectures and commercial CMPs.

10.1 Multipass-like Systems

Many current systems use multipass-like approaches for orthogonalization, albeit in a very limited or distributed manner. These systems provide a means of orthogonalizing data, communicating the results contemporaneously, and then distinguishing and processing the individually restored data sets.

For instance, the splitting of data into in-phase (I) and quadrature (Q) data streams to isolate the real and imaginary elements of each symbol is a common practice [25, 27, 108]. The separation, detection, and demodulation of each channel, the I and the Q, represents a processing pass on the data (see Figure 10-1).

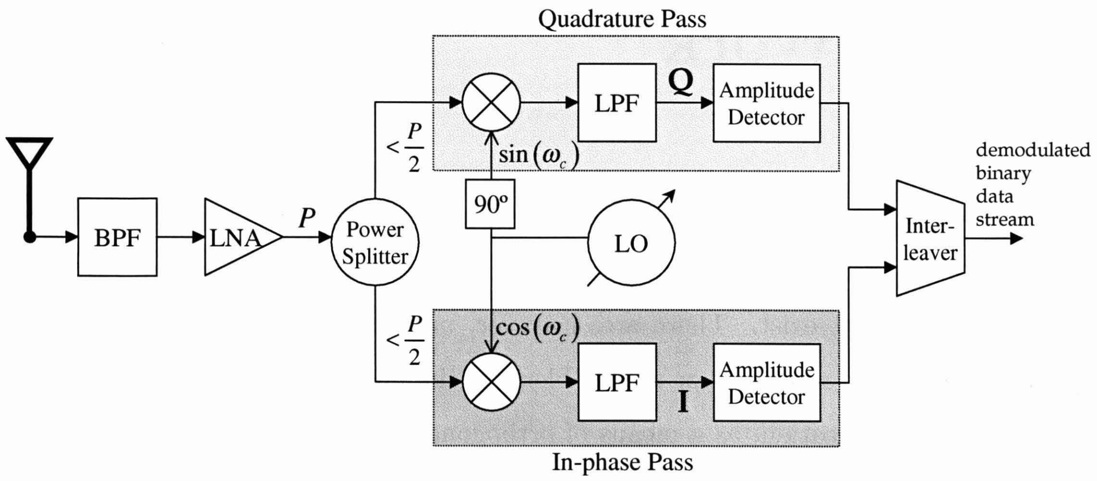


Figure 10-1: Direct conversion quadrature receiver block diagram. Band pass filtering is necessary to remove out-of-band noise and interferers. Low noise amplification then gains up the input waveform while adding as little additional noise as possible. The power splitter performs a copy operation to create two identical data waveforms. Note that this operation is lossy and the output waveforms each contain less than half the input power. Each copy then goes through its own down-conversion pass. Each waveform is mixed with an in-phase or quadrature oscillation to down-convert the I or Q channel, respectively, to baseband. Low pass filtering is necessary to remove the resulting images before detection. Amplitude detection then recovers the bit information stored in each orthogonal symbol component, I or Q. These bits are then interleaved to produce the final demodulated bit stream. Note that two orthogonal components, I and Q, results in an output data stream of twice the data rate.

I and Q channels are demodulated separately to take advantage of orthogonalization along the real and imaginary axes in the complex plane. Such systems often avoid data synchronization issues by digitizing the I and Q outputs before combining them. There is added noise, however, from the analog components in the two different paths, as well as from the lossy power splitting required to copy the input waveform. This type of system is also limited to two dimensions of orthogonal data streams, as only the two single real-time passes (I and Q) are available to extract information from the data.

Multiple access schemes are another example of using orthogonalization to improve system performance. Whether creating uniqueness in time (TDMA), frequency (FDMA), or through convolution with a code word (CDMA), each scheme imposes orthogonality on the data such that the output from each transmitter can be uniquely identified [25, 32]. Such schemes represent a distributed and very primitive form of multipass. They either fail to take optimal advantage of the available resources (TDMA and FDMA), or they provide only a minimum of orthogonal dimensions per data link (CDMA).

Interest has been increasing in an additional class of multiple access schemes those which use spatial locality to help differentiate between data. Adaptive beamforming [109, 110, 111] and space-time [112] coding are examples of using differences in location to provide improved orthogonality and isolation between data streams. Adaptive beamforming focuses energy in a specific direction in order to maximize and isolate inputs located in that direction. Space-time coding provides an arbitrary number of dimensions of orthogonality given a suitable scattering environment. The advantages of such a large number of orthogonal components can be seen in the large achievable data rates of space-time systems. Unfortunately, both beamforming and space-time systems rely upon an individual antenna and accompanying radio front end per dimension of orthogonality. In addition, the various antennas must be spread over a suitable distance. This results in an expensive, large, and fixed system. In this case, each receiver can be thought of as a separate processing pass for each data stream.

An example of perfect spatial orthogonality can be found in the wired communications world. Fiber optics is the epitome of spatial orthogonality in that there is zero coupling between fibers in a bundle. Each data signal is completely contained by a fiber and has no effect on any other signal. So, again, each data stream is processed by a separate pass, in the form of a receiver chain, on reception.

The optical fiber realm also provides another interesting analog to an multipass system. Wavelength-division multiplexing (WDM) is a means of transmitting multiple data streams in parallel over a single optical fiber [25]. The different streams are distinguished by their associated wavelengths. Multiple streams can be used in parallel to provide a higher aggregate data rate [113, 114].

10.2 Software Radios

As discussed earlier, software radios bring the adaptability of digital processing to radio functions. Most software radio systems have focused on inter-operability, as their primary function. As processors get faster and cheaper, however, software radios are also being considered for their commercial viability in allowing a single inexpensive COTS platform to bring multiple standard compatibility.

The Speakeasy Multiband Multimode Radio (MBMMR) [46, 47, 48] was one of the first software radio systems. It was a software radio project commissioned by the military to create an inter-operable radio system. Speakeasy used multiple TI TMS320C40 DSPs in conjunction to perform low data rate (less than 20kb/s) baseband functions.

The SpectrumWare work done at MIT [51, 115, 53] was some of the first to take an in-depth look at some of the advantages of performing communication processing on a general-purpose processor. SpectrumWare describes using temporal decoupling to provide a softening of real-time constraints in order to accommodate the overhead of operating in a fully general-purpose environment. The focus of SpectrumWare is on how to implement composable communications functions for inter-standard operability while accommodating the operating system and other unrelated system functions. While the opportunity for software radios to enable new communications algorithms is mentioned in [51], this possibility is not explored there.

Gnuradio [54] is an open source freely available software radio for general-purpose processors. It allows users to download optimized baseband radio functions and combine them with an available RF front and ADC/DAC to create a reconfigurable radio on their desktop [116]. Gnuradio is Open Source and accepts contributions from willing developers as its stated purpose is to allow decentralized communications systems which do not rely on establish telecommunication companies.

The commercial viability of software radios has been proven by Vanu, Inc. [117]. Vanu, Inc. has deployed cellular basestation systems running software radio functions. These systems provide inter-operability as well as upgradability for celluar systems. The majority of the software used is written in high-level portable code to allow software reuse among systems. These systems target general-purpose stock CPUs in order to take advantage of the highly optimized compiler infrastructure available.

10.3 Other Tiled Architecture Platforms

The Raw tiled architecture was the implementation platform for the work presented here. The multipass approach, however, is system independent and could be applied to other parallel general-purpose processing architectures and specifically, other tiled architectures.

The TRIPS architecture [9] uses ALUs as its tiled entity and operates on data in a data-flow manner. This dynamic execution and routing of data contrasts with Raw's statically scheduled instructions and routing. The dynamic data-flow model offers the promise of simpler network routing as long as sufficient bandwidth is present. The primary concern for a TRIPS implementation would be the mechanism for streaming input data. High-bandwidth memory access channels have been suggested, but not implemented.

Another data-flow based tiled architecture is Wavescalar [71]. Wavescalar is similar to Raw in that it provides a processing substrate which scales with technology changes and die sizes. Data-flow processing occurs on processing elements within a tiled unit, called a cluster. Inter-cluster communication takes place using dynamic routing with a "hop" of a cluster per cycle. This would allow low latency communication between the clusters as long as contention were kept to a minimum. The dynamic nature of this communication, however, makes it more difficult to assure known latencies, as is the case in a statically scheduled system. The variable latencies of a data-flow style system can be an issue for communication processing in general, although an average latency model would probably suffice for most applications.

Smart Memories [118] uses a two-tiered structure with the processor tiles organized into quads of four tiles. Communication within each quad is over nine busses. Beyond the quads, this architecture uses a dynamically-routed packet-based network to communication data between quads and off-chip. These limitations on routing resources could potentially cause issues for communication functions which do not neatly fit within a tile or a quad.

The work closest in nature to multipass on Raw is Synchroscalar [72]. Synchroscalar operates on a similar design space to the multipass on Raw system, namely 802.11a, but it is focused specifically on low power and voltage and frequency scaling. The Synchroscalar architecture is another two tiered tiled design. It groups processing elements based on the ADI Blackfin DSP ISA [119] into four processing element columns. These columns are then replicated. Each column operates in a SIMD manner and represents a single thread of control. Tiles communicate intra-column through a 256bit segmented bus, and intra-column through a lower bandwidth bus. The design networks were optimized around a single 802.11a 54Mbps Viterbi ACS. As this is the most demanding part of the 802.11a receive chain, it is likely possible to connect together multiple copies of the current Synchroscalar design in order to generate a platform large enough to implement all of 802.11a or a multipass system. The bus-based approach would not scale with larger designs, however.

Summary

While each of these areas of research (multipass-like communication systems, software radios, and multiple-core processors) have been considered individually, the multipass system presented here is unique in its combining of all three of these aspects.

10.4 Viterbi Decoder Work

The Viterbi decoder parallelization effort drew upon much previous work. The high-level block diagram structure of the software Viterbi decoder presented here is similar to many hardware designs. The low-level intra-function partitioning and algorithmic implementation, however, are design specific and are heavily optimized to take advantage of the Raw architecture.

Works by Viterbi [120] and Forney [22] were very useful in understanding the Viterbi algorithm in general, while Butler [97] and [105] provided insight into some of the specific implementation details required of a fully functional Viterbi decoder implementation, such as the choice of path metric computation and the normalization of metric values.

[121] and [101] provided useful overviews of the common Viterbi decoder parallelization approaches, with the decoder used here falling into the node-parallel class of Viterbi decoder architectures.

A number of parallelization approaches provided context for the parallelization effort by addressing the limitations imposed by the feedback loop internal to the Viterbi decoder. [122, 103, 104], [123, 124], and [125, 102] discuss methods for operating on multiple feedback paths concurrently, thereby minimizing the recursion and improving the decoder speed. [126] presents an algorithmic approach to speeding up the Viterbi decoder through the use of a Viterbi transform operation. While the Viterbi decoder used here is not directly based on these ideas, these approaches were useful in examining the space of Viterbi decoder implementations and in highlighting the parallelization challenges presented by the decoder components.

Chapter 11

Future Work

There are numerous avenues of further research suggested by the multipass algorithm and system. The multipass approach provides multiple new degrees of freedom in system design.

From an algorithmic point of view, the choice of combining function can have a large impact on the usefulness of the system for given communication needs and under anticipated operating conditions. In order to fully evaluate such approaches, however, the system must undergo additional testing with more complex and realistic channel models and radio effects.

From an implementation perspective, there is much freedom in the choice of platform. The multipass algorithm itself is equally applicable to many different types of system designs, whether general-purpose, application specific, or anywhere in-between.

From a general-purpose processing perspective, the use of the multipass algorithm on GPPs, while making possible the use of this implementation medium, only begins to take advantage of the flexibility enabled by the use of GPPs for communication processing. The ability of digital radios to algorithmically modify their processing chains and functions to adapt to current channel conditions and needs is a powerful tool and invites further investigation.

11.1 Alternate Combining Functions

Multipass functionality can be exploited in a number of different modulation and detection schemes. Different types of orthogonalizing combining functions could be appropriate depending on the scheme used. The purpose of the combining function is to join passes in an orthogonal manner so that they can later be distinguished at the receiver.

Superposition is a less than optimal combining function in some important ways. Superposition itself provides no orthogonality. There is no way to distinguish between combined signals if superposition were directly used to combine the signals. The scaling function provides the means of disambiguating the combined signals. Scaling down one of the signals, however, reduces the power in that signal and makes it more susceptible to noise. This creates an asymmetric situation where Pass1 has a higher probability of being correctly received. In fact, given that the only way to recover Pass2 signal at the receiver is to subtract an estimate of the Pass1 signal (thus undoing the superposition), the reliability of Pass2 is coupled even more tightly to the correct reception of Pass1. In addition, the superposition combining function has a limit on the amount of scaling which can be applied as any passes which are scaled to a power level below the system noise floor become unrecoverable (see Section 5.5). Another issue with the superposition combining function is that it creates dependencies between the passes where none might otherwise exist. Earlier passes must be decoded before later passes can be, and the correctness of this earlier pass decoding can impact the ability to correctly decode the later passes.

A choice of combining function other than superposition could have a dramatic impact on the system. Other combining functions may avoid some of the limitations of the superposition combining function, but could well introduce others. It would therefore be logical to explore other possible combining functions and determine for which scenarios, and for which types of data, they might be a more suitable option. A few possible alternate combining functions are discussed below.

11.1.1 Data Layering

Data layering is a combining function technique. Data layering uses a code-based orthogonalization which layers multiple virtual channels on the same transceiver into a single data channel. At the receiver, a pass for each codeword retrieves these individual virtual data channels. Codewords are a mechanism to provide an explicit and configurable means of orthogonalization.

Data layering uses multiplication at both the transmitter and receiver to provide filtering on each virtual data stream (see Figure 11-1). The first multiplication by a codeword at the transmitter decorrelates each virtual channel from all others. The second multiplication at reception then correlates a copy of the incoming data stream with a corresponding codeword,

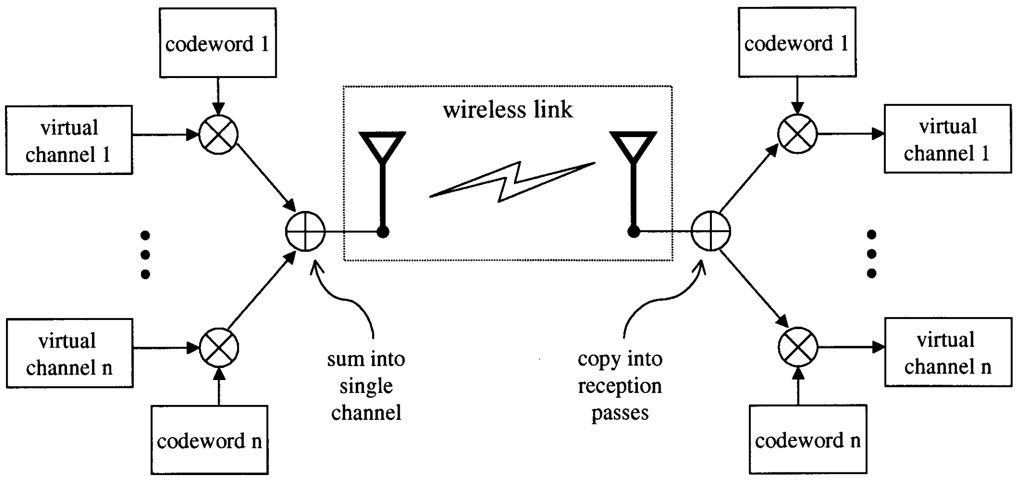


Figure 11-1: Data layering: virtual channels are multiplied in the time domain with corresponding orthonormal codewords. The transmitted data stream can then be copied into correlating passes at the receiver. Each pass multiplies (correlates) the data with a codeword to retrieve that codeword's virtual channel.

suppressing the presence of the other virtual data streams.

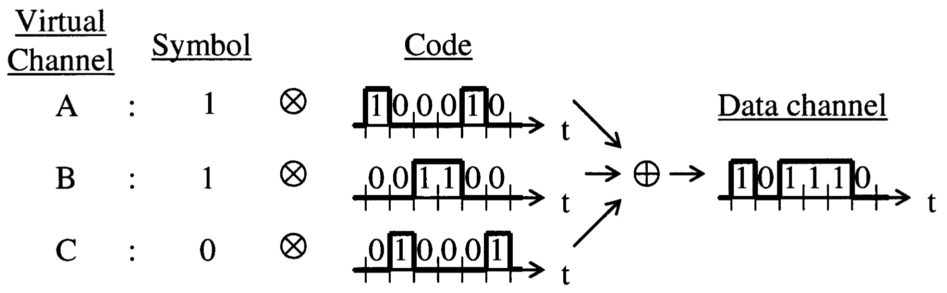


Figure 11-2: Data layering example. Three virtual channels are multiplied in the time domain (convolved in frequency) with corresponding orthonormal codewords. The resulting data stream can then be multiplied (correlated) at the receiver with each codeword in turn to retrieve each virtual channel.

A simple example of data layering at the transmitter is shown in Figure 11-2. The use of very small arbitrarily chosen codes and an on-off keying modulation scheme make this example a trivial case, but a didactic one. To combine three virtual channels, A, B, and C, the data value, or symbol, from each channel is modulated and then multiplied by a codeword. The results are then summed into a single data stream. This stream can then be sent across the communication channel. At the receiver, the digitized input stream is split into three copies, one copy for each reception pass. Each copy is then multiplied by one of the codewords. The output of this operation then has the zero code elements removed and is demodulated to produce the corresponding virtual channel data.

The choice of codewords has a profound impact on the functionality of the system. Codewords must form an orthonormal basis in order to minimize ambiguity and maximize the likelihood of correctly detecting each symbol at the receiver. For the codewords to be orthonormal they must satisfy the following properties:

$$\sum_{i,\forall j\neq i} \langle \phi_i, \phi_j \rangle = 0 \tag{11.1}$$

and

$$\langle \phi_i, \phi_i \rangle = \text{const}, \forall i \tag{11.2}$$

where each ϕ represents a codeword. Property 11.1 guarantees the multiplication by each codeword at the receiver produces a unique and independent output. Property 11.2 ensures that the transmission energy is evenly distributed across all virtual channels.

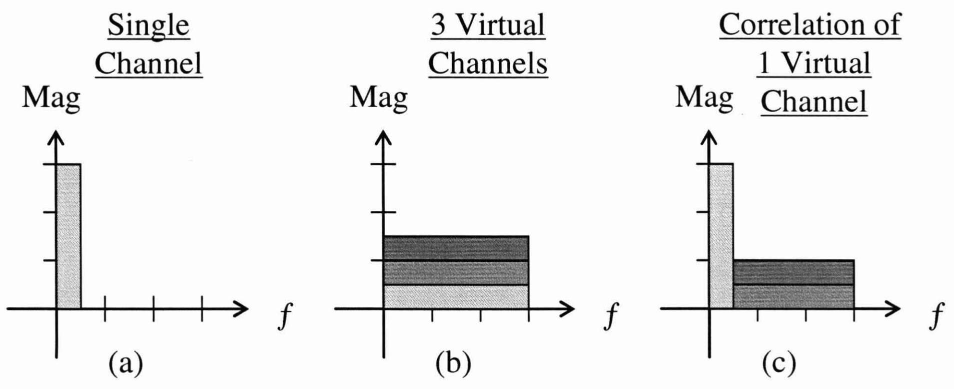


Figure 11-3: Data layering frequency spreading. Figure (a) represents the original spectrum of a single data channel. Figure (b) represents the spreading of each virtual channel's bandwidth, as well as the layering of the energy from each virtual channel to form the final spectrum of the data. Figure (c) shows the despreading resulting from correlating (multiplying) the data stream with the desired virtual channel's codeword.

The price for the increased throughput of data layering is increased bandwidth. The most efficient way of layering N virtual channels is by choosing codewords which form an N dimensional orthonormal basis. Multiplying a data stream by a codeword corresponds to convolving the data and codeword in the frequency domain. Thus, the frequency characteristics of the resulting waveform is spread to the bandwidth of the codeword (see Figure 11-3).

Data layering uses codewords as a mechanism for imposing orthogonality on the virtual data channels. The freedom to choose and dynamically modify these codewords provides significant fine grain control of the transmitted signal. For instance, the number and length of codewords can be scaled to match the desired data throughput.

One can think of data layering as a superset of CDMA and many of the multiple access schemes used today. These multiple access schemes are simply more rigid instances of data layering with a fixed set of codes. Specific classes of codes can create the functionality of CDMA, TDMA, or FDMA. For instance, a set of codewords which have a relatively flat spectrum, are very long, and result in a chip rate significantly higher than the data rate, would result in data layering very similar to CDMA. A set of codewords which emulated sinusoids at varying frequencies would emulate FDMA. And a set which had non-overlapping shifted contiguous blocks of ones would resemble TDMA. Severely limiting the flexibility of these codes is a necessity for systems in which the codes are distributed across an unknown number of devices. For this case, the best that can be done is to choose codes which will allow for a large, possibly fixed, number of devices.

Using multiple passes to implement data layering for a single transmit-receive link, however, allows for great flexibility in choice of codeword set. Since only one link is involved, only two devices, the transmitter and receiver, need coordinate codeword changes and the effects of changes are easily observable and modifiable. The codeword set can then be designed to optimally match the current conditions. The basis type, codeword length, number of codewords, and codeword frequency characteristics, can all be dynamically adjusted to tune system performance. The number and length of orthogonal codewords used can be adjusted to match the number of virtual channels required. Also, the length of the codewords determines the amount of frequency spreading in the system and can be modified to produce a narrower or wider channel as needed. The frequency content of the codewords, and hence the combined output spectrum, can be dynamically shaped to match the channel. An example of this is explored further in Section 11.1.2.

Theoretically, any codeword which provides an orthonormal basis is available and can be used. This means that practical limitations on the codewords are due to externally imposed constraints and reflect the limitations of other parts of the system. The main limiting factors in codeword choice are the amount of frequency spreading which can be accommodated by the operating environment, the maximum data conversion rate of the system, and the linearity of the analog circuits needed as an interface to transduction.

11.1.2 Colored Data Layering

The ability to choose the codes used for orthogonalization creates the opportunity to pick codes which have especially nice properties. *Colored data layering* is a technique which takes advantage of this fact. In colored data layering, the orthogonal codewords are picked such that their frequency characteristics maximize the usage efficiency of the current channel (see Figure 11-4). The spectrum of the output waveform, when all the colored data layers have been added together, then conforms to this current channel. In this way, codewords optimally make use of the available bandwidth. Data layering is maximized where there is little noise or interference and minimized where the channel is worst. The result is a channel usage model similar to that of the water-filling algorithm [28].

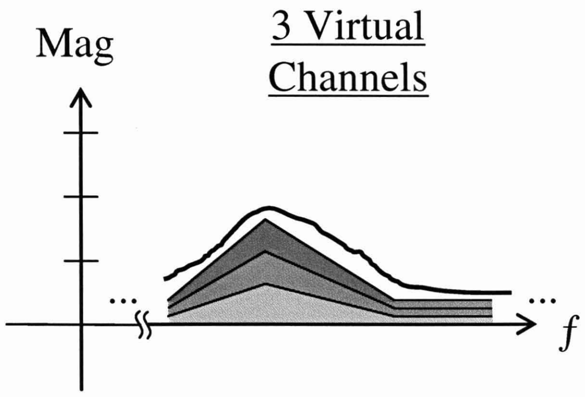


Figure 11-4: Colored data layering frequency spectrum. Codewords are chosen to shape the data frequency spectrum appropriately.

11.1.3 Frequency Bits

Multipass could be used to implement a wide-band dynamic non-contiguous OFDM system. The whole spectrum (all that is available from the ADC) could be broken down into small frequency sections, or *frequency bits*. The combining function could then be used which orthognalizes by frequency bit by assigning a set of frequency bits to each pass. A pass of the multipass system would then filter out the frequency bits of interest. The output data from each pass would then correspond to the information in a set of frequency bits of interest to form the desired system output.

Unlike OFDM, where the frequency sections assigned to each link must form a subsection of a predefined channel, in this multipass scheme, frequency bits can fall anywhere within the range of the data converter. This has the advantage that frequency bits with similar SNR characteristics can be assigned to each pass, thus allowing a single consistent modulation scheme to be used across all frequency bits in a pass. The assignment of frequency bits, and the modulation used for each pass can then be dynamically adjusted to take into account the changing channel characteristics over time (see Figure 11-5).

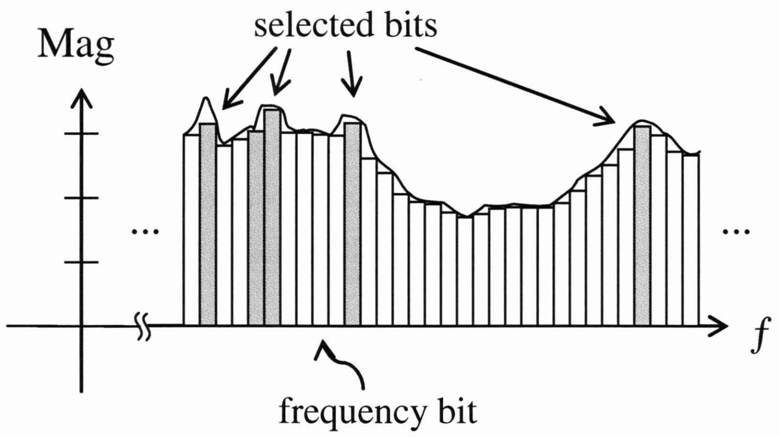


Figure 11-5: The spectrum is broken up into frequency bits. The shaded regions represent frequency bits dynamically assigned to a particular communication link. Frequency bits could be selected for a given pass based on having a similar SNR available in each frequency bit, allowing each pass to use a consistent modulation scheme across all its frequency bits. Frequency bit assignment could be dynamically adjusted to spectrum changes over time.

11.2 Alternate Channel and Radio Models

A simple noise model, AWGN, was used for the multipass system simulation presented here. While this model is appropriate for a simple evaluation of a multipass system, the application of more complex and realistic channel models is a reasonable next step. The use of a Raleigh and other more ambitious channel models [26, 25] would simulate the system operating under more realistic conditions with effects like multi-path present. This would demonstrate the impact of such suboptimal, but unavoidable, conditions on a multipass system relative to a single pass one.

A closer examination of the effects of system noise (from non-idealities such as converter resolution, system linearity, power amplifier saturation, etc.) and the limitations of the system in conjunction with the multipass combining function used would be revealing.

In addition, an investigation into the suitability and ramifications of using the multipass approach with other communication system configurations, e.g. WiMAX [127, 128] or 802.11n [129], is a logical extension of this work.

11.3 Implementation on Alternate Platforms

While the multipass system presented here focused on implementation using a general-purpose tiled architecture, the multipass algorithm is applicable to other parallel systems. The multipass algorithm is a means of using spatial resources to reduce temporal constraints and is appropriate for systems which are application-specific, as well as those which are general-purpose.

11.3.1 Other Parallel GPP Systems

The multipass system addressed here made use of a specific tiled architecture, the Raw processor. The system itself is not Raw specific, however, beyond the intra-function optimizations. Other tiled architectures, and in fact other parallel processing solutions, could take advantage of the multipass approach to implement communication processing.

The architectural features available on a given platform dictate the level of parallelism attainable for a communication system, and the corresponding temporal requirements. The specific implementation platform capabilities determine the applicability of a multipass communication system approach. Specifically, the abilities discussed in Chapter 4 are key to successful implementation and use of a multipass, or potentially any, high data rate communication system. The use of other tiled processors (like those mentioned in Section 10.3) and other parallel processing systems like CMPs (especially those making use of tightly coupled DSP cores) could provide a fruitful implementation environment for multipass communication systems.

11.3.2 ASICs

The multipass algorithm, splitting up the communication workload into multiple portions transmitted in parallel, could easily be applied in an ASIC solution. Multiple passes could be statically incorporated into the receiver design. While this would negate the ability to dynamically adjust the number and possibly the type of passes, it would reduce the processing load required of each pass. This reduced processing load could be translated into lower power consumption (through higher V_t devices and slower clocking speed or through the application of asynchronous clocking techniques) and/or simpler design requirements (through the lower critical path latency and ability to reuse function designs across passes). The smaller area per function might make such a solution appealing in a system where communication is a constant high priority function.

11.3.3 FPGAs

The hybrid nature of FPGAs make the multipass algorithm particularly appealing in this realm. FPGAs provide a modicum of the reconfigurability of GPPs with a much improved processing rate. The ability to implement ASIC functionality while retaining the ability to reconfigure the system for debugging, altering, or upgrading the system with software design tools leads to a decrease in the design time and effort of FPGAs over ASICs. FPGAs, however, while faster than GPPs (on the order of 2–3$\times$ faster [130]), are consistently slower than ASICs (on the order of 2–3$\times$ slower [130]). The application of the multipass algorithm in communication systems based on FPGAs would allow these systems to utilize the benefits of FPGAs while regaining much of the performance capabilities of an ASIC solution.

11.4 Adaptable Systems

A significant benefit of digital radios, and software radios in general, is the ability to dynamically modify the communication system. The use of software to implement the system on a GPP allows the system to be adapted based on current conditions, a feat which is not allowed by the static nature of hardware design. The ability to modify the number and types of passes in a multipass system is just a single instance of the flexibility of a software radio based approach. There is ample opportunity for research into software-based systems which are dynamically adaptable in other ways. Just a few possible areas for additional adaptation in the current system are highlighted below:

- The algorithms used can be modified based on the current state of the channel. For example, the packet detection function in the receive chain (Figure 7-2) can be adjusted based on the amount of noise and interference observed on the channel. A simple algorithm based on a direct measurement of the receiver energy can be used to estimate the start of a packet in low noise conditions. If the channel noise were to increase, however, and the number of false positives rise too high, the algorithm could be modified to a more complex, but less error prone packet detection scheme like the double sliding window algorithm.

- The protocol can be altered to take best advantage of the current conditions. For example, in a situation with significant amounts of multi-path, the cyclic prefix (CP) might be extended to allow a longer settling time between symbols.

- In a multipass system, the combining function could be changed to optimize the system for the current conditions. If the current data to be sent contained streams with differing priorities, then a superposition combining function might be appropriate. If all of the data were equally important, then a code-based combining function might work better for the current channel.

Appendix A

802.11a Specification Values

Some of the specifications values for the 802.11a baseband are presented below. The major system parameters are shown in Table A.1. Table A.2 displays the available data rates, along with the data rate specific parameter values. Some overall system values, such as timing and frequency allocations, are enumerated in Table A.3.

802.11a Major Parameters	
Information Data Rate	*6, 9, 12, 18, 24, 36, 48, and 54 Mbit/s (6, 12, and 24 Mbit/s are mandatory)*
Modulation	BPSK OFDM QPSK OFDM 16-QAM OFDM 64-QAM OFDM
Error Correcting Code	$K = 7$ (64 states) convolutional code
Coding Rate	1/2, 2/3, 3/4
Number of subcarriers	52
OFDM symbol duration	4.0 μs
Guard Interval	0.8 μs (T_{GI})
Occupied Bandwidth	16.6 MHz

Table A.1: Major Parameters of the 802.11a OFDM PHY [24].

802.11a Data Rate Values					
Data rate (MBits/s)	*Modulation*	*Coding rate (R)*	*Coded bits per subcarrier (N_{BPSC})*	*Coded bits per OFDM symbol (N_{CBPS})*	*Data bits per OFDM symbol (N_{DBPS})*
6	BPSK	1/2	1	48	24
9	BPSK	3/4	1	48	36
12	QPSK	1/2	2	96	48
18	QPSK	3/4	2	96	72
24	16-QAM	1/2	4	192	96
36	16-QAM	3/4	4	192	144
48	64-QAM	2/3	6	288	192
54	64-QAM	3/4	6	288	216

Table A.2: 802.11a Data Rate Values [24].

802.11a System Values	
Parameter	*Value*
Data Length	1–4095 Bytes
N_{SD}: Number of data subcarriers	48
N_{SP}: Number of pilot subcarriers	4
N_{ST}: Number of subcarriers, total	52 ($N_{SD} + N_{SP}$)
Δ_F: Subcarrier frequency spacing	0.3125MHz ($= \frac{20\text{MHz}}{64}$)
T_{FFT}: IFFT/FFT period	3.2μs ($\frac{1}{\Delta_F}$)
$T_{PREAMBLE}$: PLCP preamble duraction	16μs ($T_{SHORT} + T_{LONG}$)
T_{SIGNAL}: Duration of the SIGNAL BPSK-OFDM symbol	4.0μs ($T_{GI} + T_{FFT}$)
T_{GI}: Guard Interval (GI) duration	0.8μs ($\frac{T_{FFT}}{4}$)
T_{GI2}: Training symbol GI duration	1.6μs ($\frac{T_{FFT}}{2}$)
T_{SYM}: Symbol interval	4μs ($T_{GI} + T_{FFT}$)
T_{SHORT}: Short training sequence duration	8μs ($10 \times \frac{T_{FFT}}{4}$)
T_{LONG}: Long training sequence duration	8μs ($T_{GI2} + 2 \times \frac{T_{FFT}}{4}$)

Table A.3: 802.11a System Values [24].

Bibliography

[1] Intel Corporation. Handheld and handsets: Developing compelling multimedia applications with new wireless platform, Retreived November 15, 2005. `http://www.intel.com/design/pca/intro/wirelessmobility.htm`.

[2] Wade Roush. Social machines. *Technology Review*, 108(8):45–53, August 2005.

[3] Intel Corporation. The wireless landscape of tomorrow: The mobile future is within reach, Retreived November 15, 2005. `http://www.intel.com/personal/wireless/landscape.htm`.

[4] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Jae-Wook Lee, Paul Johnson, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. The Raw microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE Micro*, Mar/Apr 2002.

[5] Geoff Koch. Discovering multi-core: Extending the benefits of moores law. *Technology@Intel Magazine*, Jul 2005. `http://intel.com/technology/computing/multi-core/index.htm`.

[6] Multi-core processors–the next evolution in computing. Technical report, Advanced Micro Devices, Inc., 2005. `http://multicore.amd.com/WhitePapers/Multi-Core_Processors_WhitePaper.pdf`.

[7] J.A. Kahle, M.N. Day, H.P. Hofstee, C.R. Johns, T.R. Maeurer, and D. Shippy. Introduction to the cell multiprocessor. *IBM Journal of Research & Development*, 49(4/5):589–604, Jul/Sep 2005.

[8] Hisashige Ando, Nestoras Tzartzanis, and William W. Walker. A case study: Power and performance improvement of a chip multiprocessor for transaction processing. *IEEE Transactions on VLSI*, 13(7):865–868, Jul 2005.

[9] Doug Burger, Stephen W. Keckler, Kathryn S. McKinley, Mike Dahlin, Lizy K. John, Calvin Lin, Charles R. Moore, James Burrill, Robert G. McDonald, William Yoder, and the TRIPS Team. Scaling to the end of silicon with edge architectures. *IEEE Computer*, 37(7):44–55, Jul 2004.

[10] Benny Bing and Nikil Jayant. A cellphone for all standards. *IEEE Spectrum*, pages 34–39, May 2002.

[11] Sandeep Junnarkar. A dizzying array of options for using the web on cellphones. *New York Times*, June 23, 2005.

[12] David Pogue. Beyond wi-fi: Laptop heaven but a price. *New York Times*, June 23, 2005.

[13] David Pogue. The awkward smart phone grows up. *New York Times*, June 29, 2005.

[14] Sony Corporation. About psp, Retreived November 15, 2005. `http://www.us.playstation.com/psp.aspx?id=abouthightlight`.

[15] Ltd. Nintendo Company. Overview nintendo ds, Retreived November 15, 2005. `http://www.nintendo.com/overviewds`.

[16] Intel Corporation. Intel centrino mobile technology product information – overview, Retreived Jan 18, 2006. `http://www.intel.com/products/centrino/index.htm`.

[17] Inc. Palm. Palm – products – smartphones, Retreived Jan 18, 2006. `http://www.palm.com/us/products/smartphones/`.

[18] Hewlett-Packard Company. Handheld pcs at a glance, Retreived Jan 18, 2006. `http://h10010.www1.hp.com/wwpc/us/en/sm/WF02d/215348-64929-215381.html`.

[19] Professor Robert Gallager. 6.450 Principles of Digital Communication – I, Fall 2002. `http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-450Principles-of-Digital-Communication---IFall2002/CourseHome/index.htm`, Fall 2002.

[20] Professor David Forney. 6.451 Principles of Digital Communication – II, Spring 2001. `http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-451Spring-2003/CourseHome/index.htm`, Spring 2001.

[21] A. Viterbi. Convolutional codes and their performance in communcation systems. *IEEE Transactions on Communications*, 19:751–772, Oct 1971.

[22] G. David Forney. Maximum-likelihood seqeunce estimation of digital dequences in the presence of intersymbol interference. *IEEE Transactions on Information Theory*, 18(3):363–378, May 1972.

[23] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing.* Prentice Hall, 1989.

[24] IEEE Std 802.11a 1999. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications: High speed physical layer in the 5GHz band, Sep 1999.

[25] Edward A. Lee and David G. Messerschmidt. *Digital Communication.* Kluwer Academic Publishers, Boston, second edition, 1994.

[26] John G. Proakis. *Digital Communications.* McGraw-Hill, New York, fourth edition, 2001.

[27] Behzad Razavi. *RF Microelectronics.* Prentice Hall, 1997.

[28] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley & Sons, New York, 1991.

[29] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill Book Company, New York, 1965.

[30] Juha Heiskala and John Terry. *OFDM Wireless LANs: A Theoretical and Practical Guide.* SAMS, 2002.

[31] C.W. Helstrom. *Statistical Theory of Signal Detection.* Pergamon, New York, 1960.

[32] Theodore S. Rappaport. *Wireless Communications.* Prentice Hall, 1996.

[33] Gary M. Miller. *Modern Electronic Communication.* Prentice Hall, New Jersey, sixth edition, 1999.

[34] Leo Setian. *Practical Communication Antennas with Wireless Applications.* Prentice Hall PTR, New Jersey, 1998.

[35] David H. Staelin, Ann W. Morgenthaler, and Jin Au Kong. *Electromagnetic Waves.* Prentice Hall, New Jersey, 1994.

[36] Richard van Nee and Ramjee Prasad. *OFDM for Wireless Multimedia Communications.* Artech House Publishers, 2000.

[37] Hiroshi Harada and Ramjee Prasad. *Simulation and Software Radio for Mobile Communications.* Artech House, Boston, 2002.

[38] Sridhar Vembu, Sergio Verdu, and Yossef Steinberg. The source-channel separation theorem revisted. *IEEE Transactions on Information Theory*, 41(1):44–54, Jan 1995.

[39] G. David Forney and Gottfried Ungerboeck. Modulation and coding for linear gaussian channels. *IEEE Transactions on Information Theory*, 44(6):2384–2415, Oct 1998.

[40] Lewis Embree Franks. *Signal Theory.* Prentice-Hall, Englewood Cliffs, N.J., 1969.

[41] William Siebert. *Circuits, Signals, and Systems.* McGraw-Hill, New York, 1986.

[42] Alan V. Oppenheim and Alan S. Willsky. *Signals and Systems.* Prentice Hall, second edition, 1996.

[43] J. Mitola III. Software radios: Survey, critical evaluation and future directions. *Aerospace and Electronic Systems Magazine, IEEE*, 8(4):25–36, Apr 1993.

[44] Joseph Mitola, Vanu Bose, Barry M. Leiner, Thierry Turletti, David Tennenhouse, and A. Bush. Guest editorial - software radios. *IEEE Journal on Selected Areas in Communications*, 17(4):509–513, April 1999.

[45] Joseph Mitola III. Software radio architecture: A mathematical perspective. *IEEE Journal on Selectred Areas in Communications*, 17(4):514–538, April 1999.

[46] R.I. Lackey and D.W. Upmal. Speakeasy: the military software radio. *IEEE Communications Magazine*, 33(5):56–61, May 1995.

[47] Peter G. Cook and Wayne Bonser. Architectural overview of the SPEAKeasy system. *IEEE Journal on Selectred Areas in Communications*, 17(4):514–538, April 1999.

[48] James E. Gunn, Kenneth S. Baron, and William Ruczczyk. A low-power DSP core-based software radio architecture. *IEEE Journal on Selectred Areas in Communications*, 17(4):574–590, April 1999.

[49] Alison Brown and Barry Wolt. Digital L-band receiver architecture with direct RF sampling. In *IEEE Position Location and Navigation Symposium*, pages 209–216, 1994.

[50] J.J. Patti, R.M. Husnay, and J. Pintar. A smart software radio: concept development and demonstration. *Selected Areas in Communications, IEEE Journal on*, 17(4):621–649, Apr 1999.

[51] David L. Tennenhouse and Vanu G. Bose. The SpectrumWare approach to wireless signal processing. *Wireless Network Journal*, 2(1), 1996.

[52] Thierry Turletti and David L. Tennenhouse. Estimating the computational requirements of a software gsm base station. In *Proceedings of ICC'97*, June 1997.

[53] Vanu Bose, Mike Ismert, Matt Welborn, and John Guttag. Virtual radios. *IEEE Journal on Selected Areas in Communications*, 17(4), April 1999.

[54] Free Software Foundation. Gnu radio, Retreived November 16, 2005. `http://www.gnu.org/software/gnuradio/`.

[55] Vanu G. Bose. *Design and Implementation of Software Radios Using a General Purpose Processor*. PhD thesis, Massachusetts Institute of Technology, 1999.

[56] Michael Kanellos. Intel expands core concept for chips. *CNET News.com*, December 17, 2004. `http://news.zdnet.com/2100-9584_22-5494714.html`.

[57] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.

[58] Monica Lam. Software pipelining: an effective scheduling technique for vliw machines. In *Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, pages 318 – 328, 1988.

[59] Dean Tullsen, Susan Eggers, and Henry Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of the 22rd Annual International Symposium on Computer Architecture*, Jun 1995.

[60] A. Peleg and U. Weiser. Dynamic flow instruction cache memory organized around trace segments independent of virtual address line. US Patent number 5,381,533, Intel Corporation, 1994.

[61] E. Rotenberg, S. Bennett, and J. Smith. Trace cache: a low latency approach to high bandwidth instruction fetching. In *Proceedings of the 29th Annual ACM/IEEE International Sym- posium on Microarchitecture*, Paris, 1996.

[62] D.W. Wall. Limits of instruction-level parallelism. In *Internation Conference on Architectural Support for Programming Languages and Operating Systems*, 1991.

[63] Patrick P. Gelsinger. Microprocessors for the new millenium - challenges, opportunities and new frontiers. In *ISSCC 2001 Digest of Technical Papers*, San Francisco, CA, Feb 2001.

[64] Paul DeMone. The incredible shrinking cpu. *Real World Technology*, 2004. `http://www.realworldtech.com/page.cfm?ArticleID=RWT062004172947`.

[65] Anant Agarwal, Ricardo Bianchini, David Chaiken, Frederic T. Chong, Kirk L. Johnson, David Kranz, John Kubiatowicz, Beng-Hong Lim, Ken Mackenzie, and Donald Yeung. The mit alewife machine. In *IEEE Proceedings*, 1999.

[66] The BlueGene/L Team. An overview of the BlueGene/L supercomputer. In *Supercomputing 2002 Technical Papers*, Nov 2002.

[67] J. J. Dongarra. Performance of various computers using standard linear equations software in a Fortran environment. In Walter J. Karplus, editor, *Multiprocessors and array processors: proceedings of the Third Conference on Multiprocessors and Array Processors: 14–16 January 1987, San Diego, California*, pages 15–32, San Diego, CA, USA, 1987. Society for Computer Simulation.

[68] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan-Kaufmann Publishers, Inc., San Mateo, Calfornia, 1992.

[69] Elliot Waingold, Michael Taylor, Devabhaktuni Srikrishna, Vivek Sarkar, Walter Lee, Victor Lee, Jang Kim, Matthew Frank, Peter Finch, Rajeev Barua, Jonathan Babb, Saman Amarasinghe, and Anant Agarwal. Baring it all to software: Raw machines. *IEEE Computer*, pages 86–93, September 1997.

[70] Ronny Krashinsky, Christopher Batten, Mark Hampton, Steve Gerding, Brian Pharris, Jared Casper, and Krste Asanovic. The vector-thread architecture. In *Proceedings of the 31st International Symposium on Computer Architecture (ISCA-31)*, Munich, Germany, Jun 2004.

[71] Steven Swanson, Ken Michelson, Andrew Schwerin, and Mark Oskin. Wavescalar. In *Internation Symposium on Microarchitecture*, pages 291–302, Dec 2003.

[72] John Oliver, Ravishankar Rao, Paul Sultana, Jedidiah Crandall, Erik Czernikowski, Leslie W. Jones IV, Diana Franklin, Venkatesh Akella, and Frederic T. Chong. Synchroscalar: A multiple clock domain, power-aware, tile-based embedded processor. In *Proceedings of the 31st International Symposium on Computer Architecture (ISCA-31)*, Munich, Germany, Jun 2004.

[73] Michael Taylor. The Raw processor specification. Technical report, Massachusetts Institute of Technology, 2004. `ftp://ftp.cag.csail.mit.edu/pub/raw/documents/RawSpec99.pdf`.

[74] Michael Bedford Taylor, Walter Lee, Jason Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Kim, Arvind Saraf James Psota, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ILP and streams. In *Proceedings of International Symposium on Computer Architecture*, June 2004.

[75] Anant Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4), Oct 1991.

[76] William J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6), Jun 1990.

[77] David Steinberg. Invariant properties of the shuffle-exchange and a simplified cost-effective version of the omega network. *IEEE Trans. Computers*, 32(5):444–450, 1983.

[78] M. Taylor, J. Kim, J. Miller, F. Ghodrat, B. Greenwald, P. Johnson, W. Lee, A. Ma, N. Shnidman, D. Wentzlaff, M. Frank, S. Amarasinghe, and A. Agarwal. The raw processor: A composeable 32-bit fabric for embedded and general purpose computing. In *Proceedings of Hotchips 13*, Aug 2001.

[79] Anantha P. Chandrakasan, Samuel Sheng, and Robert W. Brodersen. Low-power cmos digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, Apr 1992.

[80] James T. Kao and Anantha P. Chandrakasan. Dual-threshold voltage techniques for low-power digital circuits. *IEEE Journal of Solid-State Circuits*, 35(7):1009–1018, Jul 2000.

[81] James Kao, Siva Narendra, and Anantha Chandrakasan. Subthreshold leakage modeling and reduction techniques. In *ICCAD 2002*, pages 141–148, Nov 2002.

[82] International Technology Roadmap for Semiconductors. Process integration, devices, and structures, 2003.

[83] Takayasu Sakurai and A. Richard Newton. Alpha-power law mosfet model and its applications to cmos inverter delay and other formulas. *IEEE Journal of Solid-State Circuits*, 25(2):584–594, Apr 1990.

[84] Takayasu Sakurai. Alpha power-law mos model. *Solid-State Circuits Society Newsletter*, 9(4):4–5, Oct 2004.

[85] Anant Agarwal. Raw computation. *Scientific American*, Aug 1999.

[86] raw architecture workstation. `http://www.cag.csail.mit.edu/raw/`.

[87] V. Agarwal, M.S. Hrishikesh, S.W. Keckler, and D. Burger. Clock rate versus ipc: The end of the road for conventional microarchitectures. In *27th International Symposium on Computer Architecture (ISCA)*, Jun 2000.

[88] R. Ho, K. Mai, and M. Horowitz. The future of wires. *Proceeding of the IEEE*, pages 490–504, Apr 2001.

[89] Michael Bedford Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Scalar operand networks: On-chip interconnect for ilp in partitioned architectures. In *Proceedings of the International Symposium on High Performance Computer Architecture*, Feb 2003.

[90] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Walter Lee, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Saman Amarasinghe, and Anant Agarwal. A 16-issue multiple-program-counter microprocessor with point-to-point scalar operand network. In *Proceedings of the IEEE International Solid-State Circuits Conference*, Feb 2003.

[91] D. Chinnery and K. Keutzer. *Closing the Gap Between ASIC and Custom.* Kluwer Academic Publishers, 2002.

[92] IEEE Std 802.11b 1999. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications: Higher-speed physical layer extension in the 2.4GHz band, Sep 1999.

[93] IEEE Std 802.11g 2003. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications: Further higher data rate extension in the 2.4GHz band, Jun 2003.

[94] J. Chuang and N. Sollenberger. Beyond 3G: wideband wireless data access based on OFDM and dynamic packet assignment. *IEEE Communications Magazine*, 38(7):78–87, July 2000.

[95] R. van Nee. A new OFDM standard for high rate wireless lan in the 5GHZ band. In *Vehicular Technology Conference, 1999*, volume 1, pages 258–262, Amsterdam, Netherlands, September 1999.

[96] Benjamin Philip Eugene Zaks Walker. A raw processor interface to an 802.11b/g rf front end. Master's thesis, Massachusetts Institute of Technology, September 2004.

[97] M.R.G. Butler, S. Armour, P.N. Fletcher, A.R. Nix, and D.R. Bull. Viterbi decoding strategies for 5GHz wireless LAN systems. In *Vehicular Technology Conference, 2001*, volume 1, pages 77–81. IEEE VTC, Oct 2001.

[98] F. Tosato and P. Bisaglia. Simplified soft-output demapper for binary interleaved COFDM with application to HIPERLAN/2. In *IEEE International Conference on Communcations, 2002*, volume 2, pages 664–668, 2002.

[99] J.G. Baek, S.H. Yoon, and J.W. Chong. Memory efficient pipelined viterbi decoder with look-ahead trace back. In *IEEE International Conference on Electronics, Circuits and Systems, 2001*, volume 2, pages 769–772, September 2001.

[100] S. Bitterlich, H. Dawid, and H. Meyr. Boosting the implementation efficiency of viterbi decoders by novel scheduling schemes. In *Global Telecommunications Conference*, volume 3, pages 1260–1264, Dec. 1992.

[101] S. Bitterlich and H. Meyr. Efficient scalable architectures for viterbi decoders. In *International Conference on Application-Specific Array Processors*, pages 89–100, Oct. 1993.

[102] M. Boo, F. Arguello, J.D. Bruguera, R. Doallo, and E.L. Zapata. High-performance vlsi architecture for the viterbi algorithm. *IEEE Transactions on Communications*, 45(2):168–176, Feb 1997.

[103] G. Fettweis and H. Meyr. Parallel viterbi algorithm implementation: breaking the acs-bottleneck. *IEEE Transactions on Communications*, 37(8):785–790, Aug 1989.

[104] G. Fettweis and H. Meyr. High-rate viterbi processor: a systolic array solution. *IEEE Journal on Selected Areas in Communications*, 8(8):1520–1534, Oct 1990.

[105] M.A. Bree, D.E. Dodds, R.J. Bolton, S. Kumar, and B.L.F. Daku. A modular bit-serial architecture for large-constraint-length viterbi decoding. *IEEE Journal of Solid-State Circuits*, 27(2):184–190, Feb 1992.

[106] A.M. Michaelson and A.H. Levesque. *Error-Control Techniques for Digital Communications.* Wiley, New York, 1985.

[107] William H. Press, Saul A. Teukolsky, William T. Detterling, and Brian P. Flannery. *Numerical Recipes in C.* Cambridge University Press, New York, NY, second edition, 1992.

[108] Thomas H. Lee. *The Design of CMOS Radio-Frequency Integrated Circuits.* Cambridge University Press, January 1998.

[109] Hans Steyskal. Digital beamforming at rome laboratory. Technical report, Microwave Journal, February 1996.

[110] Gary M. Miller. *Modern Electronic Communication.* Prentice Hall, sixth edition, 1998.

[111] Theodore C. Cheston. *Radar Handbook*, chapter 7. McGraw-Hill, second edition, 1990.

[112] G. D. Golden, C. J. Foschini, R. A. Valenzuela, and P. W. Wolniansky. Detection algorithm and initial laboratory results using V-BLAST space-time communication architecture. In *Electronics Letters*, volume 35, pages 14–16. IEE Digital Library, January 1999.

[113] Govind P. Agrawal. *Fiber-Optic Communication Systems.* Wiley, third edition, 2002.

[114] Adar Shtainhart, Ronen Segal, and Aviad Tsherniak. WDM - wavelength division multiplexing. Web page accessed Nov 2005. `http://www2.rad.com/networks/1999/wdm/wdm.htm`.

[115] Vanu Bose, Alok B. Shah, and Michael Ismert. Software radios for wireless networking. In *Proceedings of Infocomm'98*, San Fransisco, CA, 1998.

[116] Free Software Foundation. Gnu radio hardware, Retreived November 16, 2005. `http://comsec.com/wiki?GnuRadioHardware`.

[117] Inc. Vanu. Vanu software radio, Retreived November 16, 2005. `http://vanu.com/technology/softwareradio.html`.

[118] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. Dally, and M. Horowitz. Smart memories: A modular reconfigurable architecture. In *Proceedings of the 27th International Symposium on Computer Architecture*, Jun 2000.

[119] Inc. Analog Devices. Blackfin processor, Retreived November 16, 2005. `http://www.analog.com/processors/processors/blackfin/index.html`.

[120] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, Apr 1967.

[121] G. Fettweis and H. Meyr. High-speed parallel viterbi decoding: algorithm and vlsi-architecture. *Communications Magazine*, 29(5):46–55, May 1991.

[122] G. Fettweis and H. Meyr. Parallel viterbi decoding by breaking the compare-select feedback bottleneck. In *IEEE International Conference on Communications*, volume 2, pages 719–723, Jun 1988.

[123] H.-D. Lin and D.G. Messerschmitt. Algorithms and architectures for concurrent viterbi decoding. In *IEEE International Conference on Communications*, volume 2, pages 836–840, Jun 1989.

[124] H.-D. Lin and D.G. Messerschmitt. Parallel viterbi decoding methods for uncontrollable and controllable sources. *IEEE Transactions on Communications*, 41(1):62–69, Jan 1993.

[125] M. Boo, F. Arguella, J.D. Bruguera, and E.L. Zapata. High-speed viterbi decoder: an efficient scheduling method to exploit the pipelining. In *Proceedings of International Conference on Application Specific Systems, Architectures and Processors, 1996*, pages 165–174, Aug 1996.

[126] K.A. Wen, T.S. Wen, and J.F. Wang. A new transform algorithm for viterbi decoding. *IEEE Transactions on Communications*, 38(6):764–772, Jun 1990.

[127] IEEE 802.16 Task Group e. IEEE 802.16 task group e. Web page accessed Jan 2006. `http://www.ieee802.org/16/tge/`.

[128] WiMAX Forum. WiMAX Forum – technology. Web page accessed Jan 2006. `http://www.wimaxforum.org/technology`.

[129] Enhanced Wireless Consortium. HT PHY specification, Dec 2005.

[130] David Wentzlaff. Architecture implications of bit-level computation in communication applications. Master's thesis, Massachusetts Institute of Technology, September 2002.