Verification and Configuration of a Run-time Reconfigurable Custom Computing Integrated Circuit for DSP Applications

by

Mark F. Cherbaka

Thesis submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

APPROVED:

Peter M. Athanas, Chairman

A. Lynn Abbott

Joseph G. Tront

July 8, 1996

Blacksburg, Virginia

Keywords: CCM, DSP, Configuration, Verification

Verification and Configuration of a Run-time Reconfigurable Custom Computing Integrated Circuit for DSP Applications

by

Mark F. Cherbaka

Committee Chairman: Peter M. Athanas

Electrical Engineering

(ABSTRACT)

In recent years, interest in the area of custom computing machines (CCMs) has been on a steady increase. Much of the activity surrounding CCMs has centered around Field-Programmable Gate Array (FPGA) technology and rapid prototyping applications. While higher performance has been a concern in some applications, the solutions are limited by the relatively small FPGA bandwidth, density and throughput. This leads to area, speed, power, and application-specific constraints. In recent months, an integrated circuit known as the VT Colt has been developed to address some high performance digital signal processing (DSP) problems that conventional processors, CCMs, and ASICs cannot do under the space and power constraints. The Colt chip takes a data-flow approach to processing and is highly reconfigurable to suit the many computationally demanding tasks that new DSP applications present. A significant portion of the development of the Colt chip is architectural justification, functional verification, and configurability. This thesis explores verification of the Colt chip at various levels of development including mapping arithmetic computations and DSP algorithms that the Colt architecture was designed to solve.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Peter Athanas, for giving me the opportunity to work on an exciting project in my area of interest and for his support and expertise throughout the project. I would also like to thank Dr. Joseph Tront and Dr. Lynn Abbott not only for being on my committee, but for their support throughout my graduate career. Thanks also to Ray Bittner for his leadership and support on this project.

I am thankful to God for the opportunities and abilities He has given me. I would like to thank my mom and my brother Phil for their dedication to me, their support, and their belief in me as a student and as a person. This work is dedicated to the memory of my father, who believed in higher education and was willing to make any sacrifice to give me the opportunity to learn.

Finally, countless thanks to Stephanie Martin, Brendan O'Connor, Kevin Paar, Henry Green, and all my friends without whom this may have been possible, but infinitely less positive.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

The Virginia Tech Colt integrated circuit is a new chip that uses a data-flow architecture and run-time reconfigurability to achieve high performance while minimizing power and size [6]. These features help designers using the chip to reduce system size, power, and cost while maximizing flexibility and reliability. While designing a chip with these goals in mind, a balance must be made between different criteria and compromises may have to be made where possible in speed, area, power, performance, cost, technology, or approach. The important issue at the end is that the chip meets the specifications for these criteria while functionality is uncompromised. For the Colt chip, verification means the ability to correctly program, to obtain correct data, and to reconfigure it. Functional verification of the chip through various design stages and configuration for custom applications will be explored in this thesis.

1.1 Motivation

1.1.1 Verification

Since a properly functional chip is the desired end result, motivation for verification may seem obvious. The main problem, however, is how to go about the verification process. Since time and

tools are limited, only a finite number of tests can be performed. These tests must be done to test maximum functionality while ensuring reliable performance.

In designing the Colt chip, early design efforts took a top-down approach using the behavioral hardware language, VHDL. Due to reasons such as time constraints and the area-driven need for a custom layout the Colt chip was constructed and verified partially from a bottom-up approach [6]. Verification was done on the design and layout of the actual chip. Since this was the case, there was no need for behavioral simulations for verification.

Two basic types of simulations were used to test the Colt: analog and digital. Analog simulations have the advantage of accurately modeling the transistors and interconnections, taking into account the transistor's operating point, voltages, currents, capacitance, and charge sharing. Digital simulations give a basic behavioral model to test functionality at a logical level. While it is preferable to have the most accurate simulation, time and computing constraints prohibit the exclusive use of analog simulations. Analog simulations were used for critical paths, lower-level building blocks, power consumption estimates, and any parts of the chip where functionality was critical or reliability of the circuit design was in question. Digital simulations were used extensively elsewhere.

One of the first potential problems identified in the design of the Colt chip was the size of the chip. The die size was limited to five millimeters on a side with a 0.8 micron HP CMOS process [31]. Assuming a design that is optimal in size, there are several ways to reduce the area of a chip. An obvious way is to shrink the process. This was not feasible for two reasons: cost, and future growth. Future versions of the chip are planned to be at least five times the size of the Colt. If the design

was going to remain viable, area would have to be saved in other ways. Other solutions were to reduce the functionality or the routing on the chip. These areas had little room for improvement. Most of the chip was already to be a custom layout [31] and the functionality had been dictated by the design requirements. The final way to reduce the die area was to cut down on the number of transistors through circuit design methods. This includes n-pass technology and pseudo-nmos circuits. Although these methods save space and simplify the design over their complementary counterparts, they introduce problems with speed, power, and reliability. It is important to test these circuits for correct logic, function, power, and speed in a simulated real-world environment.

There are other areas where functionality, timing, speed, or power of a circuit are very important. For example, the clock generation circuit was tested extensively for reliability and timing. In such cases, both analog and digital simulations are performed.

A big part of the verification effort on the Colt chip involved more than just verifying the actual layout, circuits, or design. There was also a need for a proof-of-concept. Of importance is the chip's reconfigurability and ability to provide a solution for digital signal processing (DSP) and other problems it was designed to address [6].

1.1.2 Configuration

When dealing with real-time applications, the amount of processing that can be done on a given data stream is under stringent time restraints. In digital signal processing, more calculations performed in this time frame leads to more information that can be processed. For communication applications, this means greater bandwidth, more users, and a more reliable connection. These are

the criteria driving the need for higher performance computing solutions the Colt chip addresses.

Currently, many DSP applications rely on high-performance DSP processors such as the SHARC [2] or other ASICs [21] for processing power. There are several reasons why alternative solutions may be sought. First, while conventional processors may be adequate for most applications today, aggressive algorithms for future needs require a different solution. Particularly, there is a need for higher arithmetic performance. Although this need can be solved by using a parallel architecture [23], simply stringing several conventional processors together is not always feasible due to size and power constraints. Furthermore, scalability issues would have to be addressed. These include memory, communication, and efficiency. For example, does each processor have local memory or is there one shared memory? How will the processors be connected? This could be done in any way from a full crossbar to a ring with design trade-offs. How the processors communicate will also have to be addressed. Control can be given to one processor or can be distributed. When these issues are resolved, there will still be problems with distributing tasks so that processors are not idle. Significant changes at the software and hardware level are required to scale conventional processors. In contrast, the Colt architecture is highly scalable with little change in algorithms. Changes in hardware are small at the chip level and virtually nonexistent at the system level: simply adding boards will increase the processing capacity. Computations lend themselves to grow across more resources; and memory and control are virtually unaffected by scaling.

The second reason for alternative solutions is a physical one. Although CMOS process technologies continue to shrink, increasing processor and memory speeds, the performance gains are not keeping pace with the demand. Although the shrinking pace of CMOS processes has over-

come technical obstacles in the last twenty years, the rate is expected to slow down within the next ten years due to economic issues related to the fabrication facilities that will support these processes [35]. Clearly, there is a need for dramatically new architecture and approaches to solving these computational problems.

1.2 Approach and Contributions

The Colt is a large and complicated integrated circuit that strikes a balance between many variables at the chip and system level. The work presented in this thesis centered around an aggressive time frame to verify the functionality of the Colt. This involved designing and mapping algorithms for this kind of architecture for the first time. To reach this goal, every level of design from the transistor to the whole chip had to be verified for concept, functionality, and timing. As a result of the verfication efforts, errors were identified and solved, resources were redesigned, and programming was streamlined. Timing and programming specifications are presented here. Furthermore, this research provided a positive proof-of-concept for the Colt chip and showed its effectiveness in solving arithmetic and DSP problems.

Most of the development work done on the Colt was done with the aid of computer-aided engineering (CAE) tools. Some of these tools were commercial products and others were developed in-house. The work presented in this thesis began with a search for appropriate verification tools as well as exercises in their interface and use. As a result of these efforts, many errors in the design and implementation of the Colt were identified and corrected. Timing issues were also identified and specified; furthermore, potential power problems were eliminated. Programming procedures are

identified for the first time and algorithms show the usefulness of the Colt to solve computational problems. With the work and conclusions presented here, correct and efficient operation of the Colt chip can be expected.

1.3 Organization

An understanding of the Colt architecture is necessary for understanding of verification and configuration procedures. Chapter 2 gives a background of the Colt chip and platform. An overview of the design and functional capabilities is given. Since the Colt configuration is closely related to the architecture and verification, an overview of the configuration procedures and limitations is required. Chapter 3 presents a detailed description of configuration procedures.

Chapter 4 focuses on the cycle of verification and redesign in the course of the development of the Colt chip. The procedures for verification and design will be reviewed. Background of testing procedures and tools used will also be presented.

Chapter 5 covers the basic mathematical calculations that the Colt chip will need to perform and how the chip was programmed to solve these problems. In addition, some DSP algorithms for Virginia Tech's GLOMO project will be reviewed. The solutions and results as implemented on the Colt chip will be presented along with a comparison of more conventional solutions.

The Colt chip is just the beginning of a possible new class of solutions for DSP problems. With this in mind, the Colt chip is an important test chip that is expected to provide immediate solutions to research and commercial applications. It is also expected to be tested to show areas for improvement before larger chips with the same architecture are developed in the future. Chapter 6

will outline some areas that are still open for future work.

Additional information on the details of verification procedures on the tools used, including problems and solutions, are presented in Appendix A.

Chapter 2

Colt Architecture

2.1 Overview

Before verification can begin, knowledge of the architecture and operation of the chip and its components is required. Verification can identify problems that require redesign of one or many components at any level of hierarchy. This chapter provides a detailed background of the chip architecture.

The Colt chip uses a general mesh architecture that is highly reconfigurable for both static computational structures and dynamic run-time dependent structures. Although it is used in the Colt chip, this architecture can be used in other chips and applications. This thesis focuses on the general architecture but uses the Colt chip as an example. The architecture and configuration methods can easily be extended to other chips, the most notable being a larger version of the Colt chip called the Stallion. The Colt is a smaller version of the Stallion that is being used for design verification purposes; however, it does not fully exploit the capabilities of this kind of architecture. For ease in reference, this architecture will be referred to as the Colt architecture and all references to the chip will be to the Colt chip unless otherwise noted.

The Colt architecture falls under a relatively new class of computer hardware called cus-

tom computing machines (CCMs). These machines attempt to provide software-like configuration with hardware-like performance, providing a compromise between full custom ASICs and general-purpose processors. CCMs tend to have several key features in common. Among these are reconfigurable logic or processing elements (PEs), reconfigurable communication resources, and reconfigurable I/O [32]. The prominent CCM architectures today are based on a class of chips called field-programmable gate arrays (FPGAs) which use programmable logic blocks as processing elements [44]. FPGAs offer the advantage of being highly reconfigurable. Any logic or interconnection resource is almost independently reconfigurable. One of the disadvantages of this is that the data path through each PE is only one bit wide. The Colt architecture differs significantly in that it supports word-length processing and communication between the various PEs. While this may reduce independent bit-wide configurability, it makes the Colt very efficient for processing word-length data.

CCMs fall into a class of computing machinery separate from the various prevalent architectures used for general-purpose processors [18](Table 2.1). Multiple data paths are supported; however, instructions cannot be classified as single or multiple streams. This is mainly because there are no explicit instructions. The configuration of the chip determines the instructions that are performed on the data. For the Colt, there are no instruction fetches nor is there a program counter. Data flows in through input ports, goes through a crossbar and mesh and exits through output ports. The ports and crossbar are exclusively routing resources, while the mesh adds processing resources that manipulate the data streams. The Colt chip has no significant on-chip memory and therefore requires external memory to load and store data streams.

Table 2.1: Major classes of computer architecture.

SISD	Single Instruction, Single Data
SIMD	Single Instruction, Multiple Data
MISD	Multiple Instruction, Single Data
MIMD	Multiple Instruction, Multiple Data

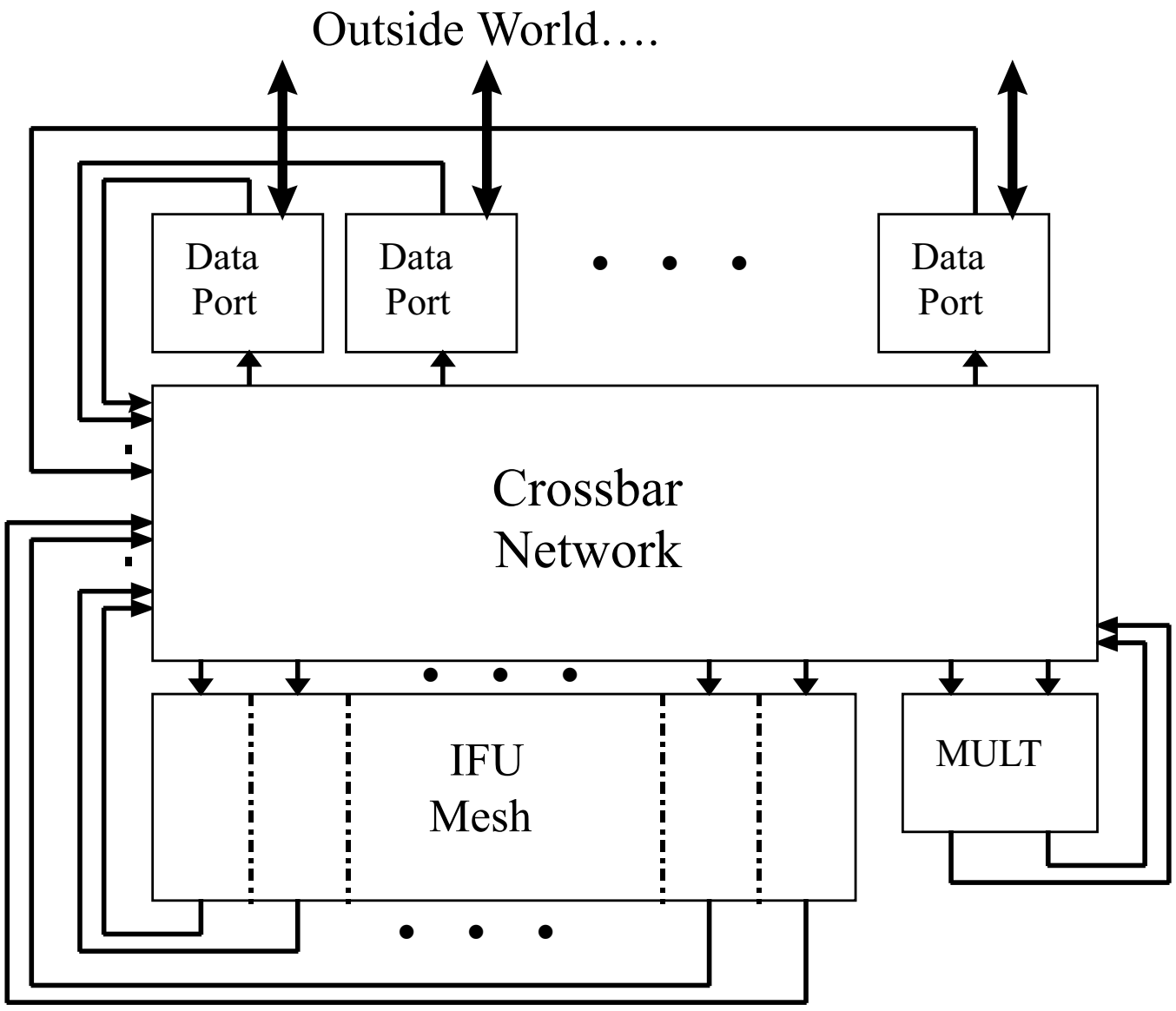

Figure 2.1: Colt chip architecture.

Figure 2.1 shows a high level view of the Colt architecture. There are a number of I/O data ports, a mesh of PEs called interconnected functional units (IFUs), and a crossbar that connects the mesh and the data ports. The number of data ports on the Colt chip is six, and there are sixteen IFUs arranged in a four by four mesh. Also included in the architecture is a special-purpose array multiply unit that will perform a 16-bit by 16-bit multiply each clock cycle with a two cycle latency.

The Colt architecture uses a data flow approach to processing. There are two basic modes in which each unit within the Colt operates: data and programming. During the programming mode, programming information goes into parts or all of the chip and configures it for processing. During the data mode, data simply passes through the chip in a pipelined manner [22]. The data or programming information can be represented as a stream. A stream enters the chip and goes through a number of resources where data is processed or programming is used to configure the resource. The path it takes through the chip can be statically or dynamically determined. It is similar to worm-hole routing [23]. The Colt chip is run-time reconfigurable, meaning that parts of it can be in data mode while other parts are being reconfigured in programming mode.

2.2 Mesh

Central to the computing power of the Colt Architecture is an array of IFUs arranged in a mesh [23]. This mesh not only contains the processing power of the chip, but also gives the Colt chip its data flow nature. The mesh supports communication between the IFUs by providing data path and control signal connections. Each node of the mesh is one IFU that holds bus routing and

a Functional Unit (FU). It is in an FU that any processing of the data takes place.

This mesh architecture is similar to a systolic array [12]. The main difference between the Colt architecture and most other systolic array architectures lies in the configurability of individual PEs. In a systolic array, each PE performs the same function on a series of data streams, hence it falls under SIMD architecture [23]. The Colt's configurability as a CCM allows each of its PEs (a single FU) to perform a different function. Programmable interconnections and full word-length data paths further distance the Colt from systolic arrays.

2.2.1 Functional Unit

The functional unit (FU) is the heart of the Colt architecture. It contains the core of the data path that includes registers, an Arithmetic Logic Unit (ALU), a barrel shifter, a conditional unit, and additional logic. Also contained in the FU are the configuration registers that hold programming information for an entire FU and IFU.

Because the FU is a critical component and for several other key reasons, most of the verification efforts centered around this unit. First, a majority of the building blocks of the chip were contained in the FU. Second, the configuration process was largely dictated by how the FU was programmed. Third, functional verification centered around the FU: it had to support all desired functionality; and fourth, the critical path, which passes through the FU, had to be minimized. A full understanding of the functionality and configuration of the FU is required.

Since the FU's main purpose is to process the data, the first part to consider is the data path. The data path through the entire Colt chip is sixteen bits wide. Through the FU, there is an

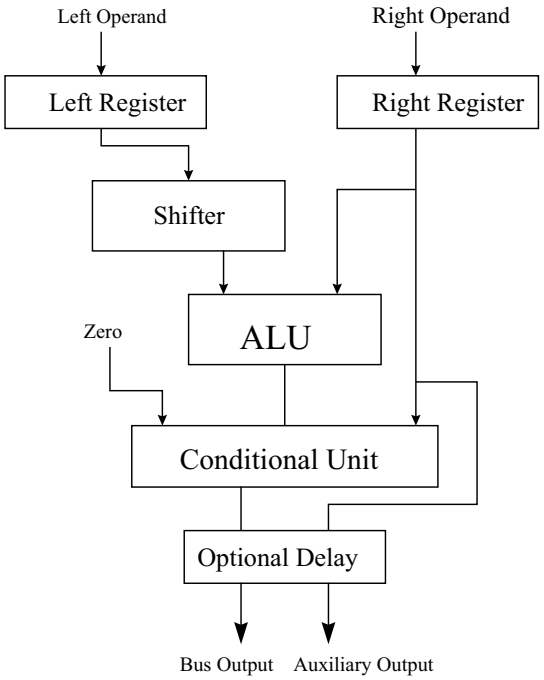

Figure 2.2: Data path through FU.

additional 17th bit that indicates valid data when set (the valid bit). Figure 2.2 shows the data path through the FU. It passes through a shifter, an ALU, a conditional unit, and an optional delay register.

Data Path

The FU has two inputs, left and right, and two outputs, a bus output and an auxiliary output. Each of the inputs is first latched into a register. The left input register employs some logic to allow several different options for latching new data. Table 2.2 shows the different modes and their respective actions. The left path then passes through the barrel shifter while the right goes directly to the ALU. The barrel shifter can be configured to do one of five shifts: shift left by

Table 2.2: Left input register latch modes

Mode	Operation
0	Loads any data word
1	Loads a valid zero if FN = 1
2	Loads only valid data words
3	Holds constant loaded at configuration time

zero, one, two, three, four, or shift right by one. When shifting in more than one, multiple bits are supplied by the right input. These capabilities are available for use in fixed and floating point operations. All shifting configurations are programmed statically, but the shifter can perform the shift conditionally. The output of the shifter also feeds into the ALU.

The ALU is the part of the datapath where computations are performed. The ALU is highly reconfigurable and has three data inputs, three programmable inputs, and two data outputs. The data inputs consist of two operands, X and Y, and a carry in, C(in), while the data outputs consist of the output, Result, and a carry-out, C(out). The three programmable inputs, propagate (P), generate (G), and result (R), are used with the input and carry signals in each bit of the ALU to allow a variety of functions. Figure 2.3 shows one bit of the ALU. The three terms P,G, and R are stored in the configuration registers and are multiplexed using the input and carry signals to generate the output and carry out. This allows the ALU to be programmed for a variety of functions including addition, subtraction, pass-through, and all logic functions. Programming of the ALU will be outlined in Chapter 3.

The next element in the datapath, the conditional unit, allows a mechanism for decision making. One of three inputs to this unit is routed to the output: the ALU output, the latched right input,

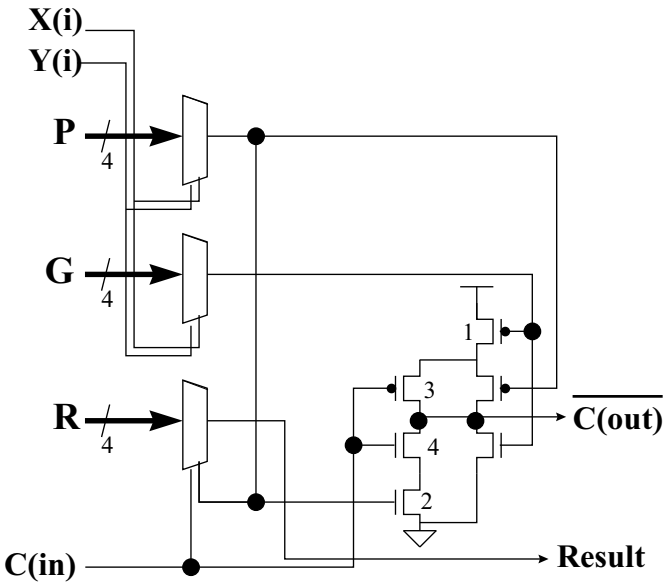

Figure 2.3: ALU bit-slice.

Table 2.3: FU conditional modes.

Mode	FN	Conditional Output
0	0	zero
0	1	ALU output
1	0	Right input register
1	1	ALU output

or zero. Two signals determine which input is selected. One signal, the conditional flag (FN) is determined at run-time. The other signal is a mode bit determined when the path is programmed. Table 2.3 shows the possible selections in the two modes.

The last element of the FU datapath is a delay unit with an optional output register, used to program a one clock cycle delay into the path. This delay can be used to align operations. Like other portions of the datapath, it is also used in programming. The output from the delay unit is the bus output from the FU. The auxiliary output of the FU is the output of the right input latch.

Table 2.4: Bus output valid bit.

Mode	Valid Bit Calculation
0	FNOut
1	FNOut AND left valid bit
2	FNOut AND right valid bit
3	left valid bit AND right valid bit

Whenever the bus output is delayed, the auxiliary output is also delayed.

Control Flags

Although much of the data flow control is programmed into the chip statically, a mechanism for run-time data control is needed. Also a method to input, output, or preserve shift or carry information between FUs is needed. In addition to the datapath is one valid bit and three control flags: a shift flag (FS), a carry flag (FC), and a conditional flag (FN). The valid bit enters and exits at the inputs and outputs with the data, but is used and manipulated inside the FU. Since data is always flowing through the Colt, the valid bit serves the purpose of indicating valid processing and programming data. The three flags can be generated external or internal to the FU. External to the FU, the flags have the same connection capabilities as the main datapath, allowing them a high degree of flexibility. Inside the FU, the flags have a wide range of uses in control of the data.

The auxiliary output valid bit is simply the auxiliary input valid bit in data mode. The bus output valid bit can be programmed to be one of four possibilities listed in Table 2.4. This can be used as a way to conditionally control the validity of data coming out of an FU based on either the validity of the input data or an external signal.

The three control flags FX, where X represents S, C, or N, exist in three forms in the FU:

Table 2.5: FU Flags: Input and output signals

Flag	Input Possibilities (True or Inverted)	Output Possibilities
Shift (S)	Constant FSOut Cond Sign FSIn	FSOut
Carry (C)	Constant FSOut FN FCIn	FCOut
Conditional (N)	Constant FSOut FNOut FNIn	Right Valid Right Sign Bit Right Bits 4-0 FNIn ALU Sign Bit FCOut NOR of Right Bits 15-14,13, or 12

external inputs, internal signals, external outputs. The names for the three forms are, respectively: F*X*in, F*X*, and F*X*out. All three control flags can be generated internally or taken as external input from another FU. Internally, the shift flag, FS, is an input into the shifter. It shifts into the LSB in a left shift or the MSB in a right shift. The carry flag, FC, is a carry input to the ALU. The conditional flag, FN, is used to select the input to the conditional unit; however, FN can also be used to determine the bus output valid bit or to internally generate FC or FS. Each flag has a variety of possible input and output possibilities that give the Colt a high degree of run-time control, allowing it to be a more general-purpose chip. The input and output possibilities for all the flags are shown in Table 2.5.

Configuration Registers

During programming, data that first enters the FU goes into a series of latches and registers to be stored as programming data for the FU and the IFU. This data determines the modes and logic decisions that make up the programming. There are just over six configuration registers in the FU that hold this data

FU State Machine

Programming and data control throughout the FU is determined by an internal state machine. This state machine determines the sequence of events for various programming and data signals and is a critical part of the FU.

2.2.2 Interconnection Functional Unit

The Interconnection Functional Unit (IFU) provides a means for interconnecting data and control paths of multiple FUs. It does this mainly through the support of connection buses for the data and flags, providing a mechanism that determines the connections and flow within the mesh. The two main ways one FU can connect to another FU is through a local bus or through the skip bus.

Local Bus

The first connection mechanism supported in the IFU is a local bus connection. Local bus connections allow any two neighboring FU's to communicate. Any FU Bus Output can be configured

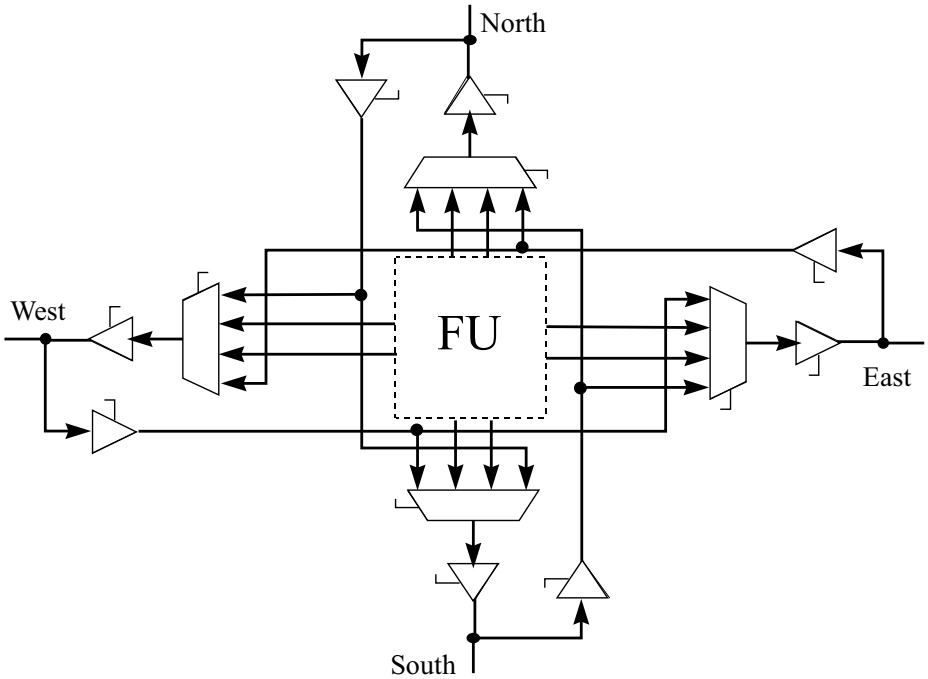

Figure 2.4: IFU Connections.

as an input to any of the neighboring FUs immediately to the North, South, East, or West. There are four local bus connection outputs from the IFU to each of the neighboring IFUs. For each direction there is a separate valid bit along with the common bus output of the FU. The valid bit is used to conditionally pass programming data. Furthermore, there are four inputs to the IFU from each of the neighboring IFU local buses. Any of these four inputs can be either the left or right input to that FU. In addition to providing local bus support for the datapath, the IFU provides identical independent local connections for all three control flags.

Skip Bus

The skip bus provides a more flexible means of communication between FUs that are one or more IFUs apart from each other. Unlike the local bus which has dedicated unidirectional buses in and out of the IFU in each direction, the skip bus has a single configurable bidirectional bus in each direction. The internal connection between the four skip buses is also configurable. Figure 2.4 shows skip and local buses of the IFU. The IFU skip bus output in each direction can be programmed from one of four possible inputs: the IFU skip bus directly to the right (compass right), the IFU skip bus directly opposite (compass opposite), the FU auxiliary output, or the FU bus output. Furthermore, that direction output must be enabled. This flexibility allows the skip bus to be used to pass the auxiliary output, the bus output, to bend a data path without using the FU, or to skip over an FU. The skip bus can also be partitioned along any row or column and used separately by different FUs.

Like the local connections, the skip capability is identical for all the control flags in the IFU. This flexibility is designed to increase efficiency of algorithms mapped to the Colt by removing the burden of connectivity from the FUs, whose resources can be used for data processing. This will become more evident in chapter 5.

IFU State Machine

The IFU also has a state machine that initializes the connections at reset and allows programming to reach the FU. This state machine is critical for proper operation of the IFU, and like other state machines in the Colt chip was an important part of the verification process.

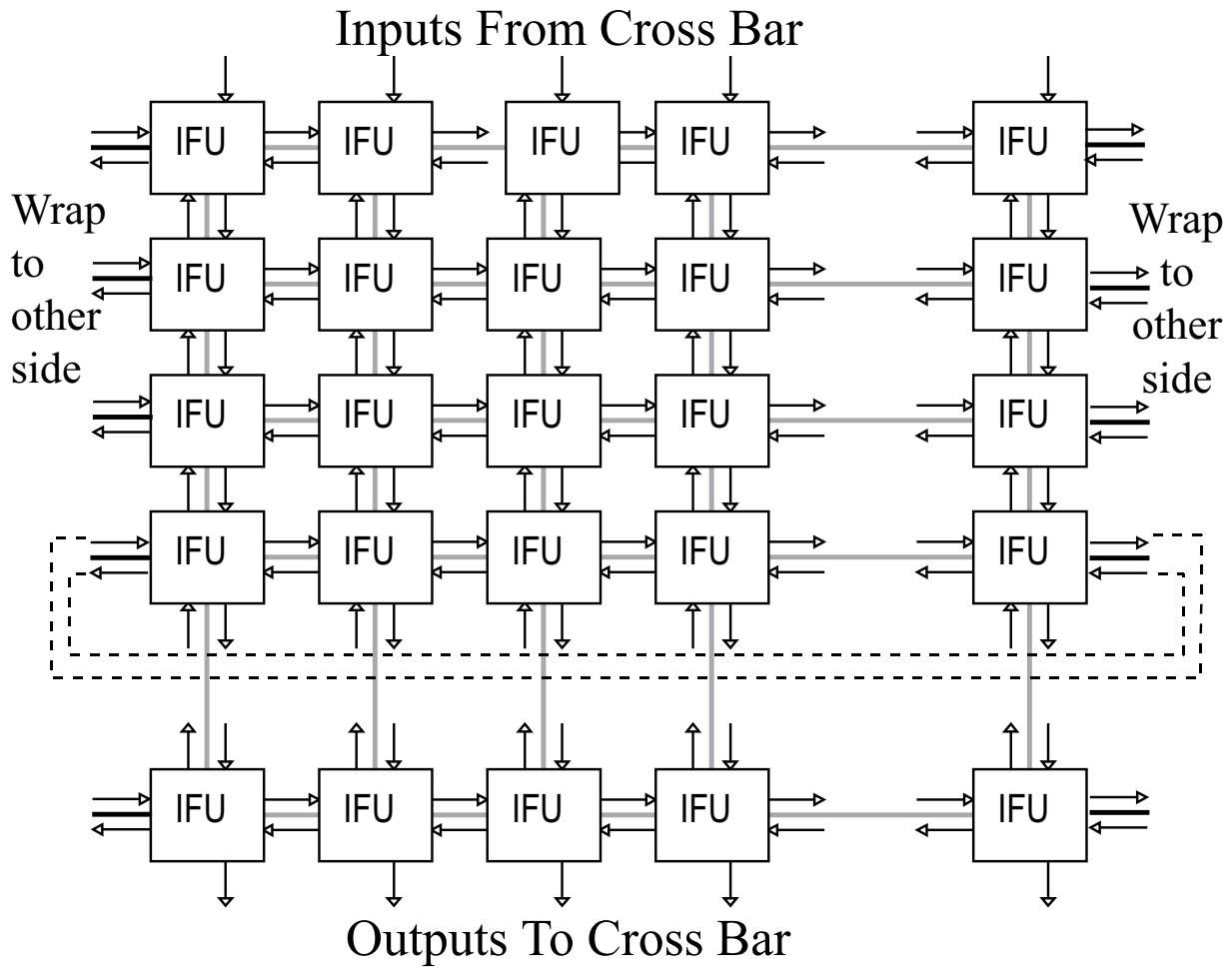

Figure 2.5: Mesh of IFUs.

The IFU Mesh

A number of IFUs are connected together in an array to form a mesh. The mesh provides connection between neighboring IFUs, but the skip buses give signals the ability to connect between IFUs that are not direct neighbors. For this architecture, the skip bus can pass signals to other FUs an arbitrary distance away, in theory. Physical restrictions do put a limit on this distance, and verification results will determine this limit.

The Colt chip uses a mesh of four rows of four columns each for a total of sixteen IFUs and hence, sixteen FUs. The critical parts of the mesh are the outside perimeter of IFUs. Ideally, a full connection of the mesh from all sides through the crossbar is preferred, but physical limitations prevent a crossbar and data port sizes that will support such a level of connectivity.

There are two inputs into each IFU in the top row and one output from each IFU in the bottom row. The data flows in through the top of the mesh and out through the bottom. The sides of the mesh retain some functionality by wrapping around the sides, much like the Iliac mesh [23] (Figure 2.5). The top of the mesh has local and skip inputs while the bottom, due to programming constrains, only supports local output. The local inputs from the bottom and local outputs in the top of the mesh are not connected. The inputs to the mesh come from the crossbar and the outputs feed back into the crossbar.

2.3 Crossbar

The crossbar provides high level interconnection capabilities between the mesh and ports. This includes connection capabilities between the mesh and itself. These capabilities make the chip architecture extremely flexible, increasing the number of possible configurations and allowing the resources to be used more efficiently.

Since any data or programming information that goes to the rest of the chip must first pass through the crossbar, the crossbar is designed to support data and programming information. The crossbar is an array of input and output buses and program lines that are connected through reconfigurable crossbar nodes. These nodes can be programmed to connect an input bus to an

output bus or to leave them unconnected. If connected, the program input is also connected to the program output. This is used to pass on programming information during chip configuration.

The crossbar on the Colt chip is a twelve by sixteen crossbar. The twelve inputs are: the six data ports, four mesh outputs, and two multiplier outputs. The sixteen outputs are: the six data ports, eight mesh inputs, and two multiplier inputs. Since the crossbar is reconfigurable, any of the inputs can be connected to any of the outputs with the exception of the data port inputs to data port outputs; furthermore, the crossbar supports broadcasting so that any one input can be connected to all eight mesh inputs. Figure 2.1 shows this connectivity.

The flexibility of the crossbar can allow a mesh to be used more efficiently. Suppose for example that an algorithm requires a six by two mesh. The crossbar can be configured to route the output of two columns of the Colt chip mesh outputs to the inputs of the two previously unused columns. This creates a virtual six by two mesh. The multiplier can also be accessed in the middle of a stream. Having the data ports as part of the crossbar can allow the mesh to be split and used independently. For example, a three-input one-output algorithm on a three by three mesh and a one-input one-output algorithm on a three by one mesh can independently coexist on the same chip.

2.4 Ports

The data ports provide the sole means of off-chip communication for the Colt. It is where data and programming streams enter and exit. The data ports not only provide a communication path, but a memory interface as well. Each data port is reconfigurable as an input or an output. There

are two sides to a data port. The first is the interface to memory and the outside world. The opposite side is the interface to the crossbar, and hence the rest of the chip. When configured as an input, the port passes data from the memory to the chip. When configured as an output, the port passes data from the chip to the memory. Since programming streams may come in or out through the data ports, programming support is provided in both directions.

2.4.1 Memory Interface

The data ports are designed to communicate with a memory controller or another Colt chip. When attached to a memory controller, there is a series of synchronization signals that control the flow of data to or from memory. These signals are the transmit and receive pins. When the data port is used as an input, the transmit pin is asserted by memory when it is writing valid data to the port. The receive pin is asserted by the data port as a ready signal.

Since the Colt chip is data driven and has no on-chip memory, the above is the only mechanism for controlling flow through the chip. When the data port is used as an output, the transmit pin is used as the valid bit by the data port. The receive pin can be de-asserted by memory to stop the flow of data at the *input* to the chip when used in loop-back mode.

Since each data port has its own memory controller, there is a concern that data intended to flow into the chip synchronously may be unsynchronized should one of the interfaces temporarily fail or should there be invalid data in the memory. There is a mechanism within the data ports that allows them to be operated in one of three modes: synchronized, loop-back, or normal. In synchronized mode, any combination of ports can be required to synchronize their memories. If

one of these ports receives invalid data from its memory, all ports de-assert their receive lines and wait until valid data arrives at the port which first received invalid data. This insures that no valid data is lost and that all data is synchronized. The loop-back mode allows memory to de-assert the receive line on an output port which causes the input ports to de-assert their receive lines to control the flow at the input.

Programming information coming from the memory side can either program the data port, or can be passed along as data would. The only difference would be the assertion of the program signal on the chip side of the port. Programming information not intended for the data port coming from the chip side can assert the program signal on the memory side.

2.4.2 Chip Interface

The chip interface of the data port provides input and output paths to the rest of the chip through the crossbar. Unlike the memory side of the data port, there are two unidirectional buses on the chip side. The first, used when the data port is configured as an input, is an output bus that feeds the input side of the crossbar. The second, used when the port is configured as an output, is an input bus fed by the output side of the crossbar. This arrangement for the Colt chip can be seen in Figure 2.1. In addition, there are signals representing the valid bits for the respective buses. Both buses also have program signals that can be used to indicate programming information in both directions.

Other data ports in the chip communicate with the data port through the chip interface as well. Each data port has a single output synchronization signal that feeds back to itself along with the

synchronization signals from the other data ports.

As stated before, the Colt chip has six data ports that can each be configured as an output or an input. Hence, there are six inputs to the crossbar and six outputs from the crossbar that are dedicated to the data ports. For proper functionality of the chip, it is important that the crossbar connections as well as the data port directions be configured properly. Data ports clock data into a register and hence have a one clock cycle delay through them.

2.5 Performance

To meet the demanding goals of the DSP applications, the Colt architecture is being targeted for a clock speed of 50MHz. Modern 0.5 micron processes can successfully run at speeds of 100MHz. The high degree of connectivity in the Colt architecture introduces some capacitance and RLC issues that can limit this target speed. Although pipelining can minimize these effects, it does so at the expense of chip area. Furthermore, due to cost considerations the Colt chip will be fabricated in a relatively slower and less efficient 0.8 micron process. This brings the Colt chip speeds down significantly while leaving open the possibilities for significant improvements in the future.

The initial run of the Colt chip has been designed and tested to perform at the target clock speed. This is done with no pipelining between FUs and with minimal signal buffering in the IFUs. Furthermore, there is virtually no limit to the skip bus range within the three by four mesh. With sixteen FUs and one multiplier, the Colt chip can still perform at a theoretical maximum 850 million operations per second (MegaOps). More importantly, DSP algorithms can realistically be programmed to output one result every clock cycle, or 50 Million calculations per second. This is

a sufficient rate for the applications currently being implemented.

2.6 The Colt Platform

Although the Colt architecture is targeted at the chip level, the stream approach to programming and data is easily extended to the system level. This means that a number of Colt or Colt-type chips can be strung together in series, parallel, with or without memory to create a variety of different high level platforms that maximize the capability of the Colt chips. There is currently work underway [20] to develop such a platform using the Colt chips and the Colt architecture.

Chapter 3

Configuration Background

Configuration and verification of the Colt chip and architecture are closely related topics. Configuration of the colt chip is essential to verify programming procedure and chip functionality. While the process of verification can find errors at circuit, logic, or architectural levels that require redesign at that level, this level of verification highlights problems that require redesign of parts of the configuration mechanism. This redesign may change the capabilities or the configuration procedure of the chip.

Understanding of the configuration procedures is also crucial for implementing algorithms on the Colt chip. A description of the limitations and capabilities of the chip at a detailed level are essential for efficient and correct configurations. This chapter outlines the the configuration mechanism in the Colt architecture and gives an overview of tools at various levels that aid this configuration.

3.1 Reconfigurability

Reconfigurability is one of the staples of a custom computing machine (CCM). Configuration mechanisms vary widely, but in general, an increase in the level of configurability raises the complexity and size of the chip. Most chips used in reconfigurable platforms including [44, 1] use

dedicated programming paths to statically configure the chip resources. This programming puts the entire chip into a programming mode. The Colt uses the actual data path along with minimal hardware support to configure the resources. This gives it an area and power advantage over other designs.

The other advantage of the Colt architecture is its run-time reconfigurability. Run-time reconfigurable chips can be partially reconfigured while other resources remain unaffected and able to continue processing. The Colt architecture allows multiple paths into or through a chip. Any one or more of these paths can be reconfigured while leaving the others processing data. Furthermore, paths that are reconfigured can immediately begin to process data independently of when other resources in other paths are ready. Run-time reconfigurability gives the programmer a wider degree of flexibility by allowing a more efficient use of resources within the chip.

Run-time reconfigurable chips have been recently developed and are starting to see some popularity [43], however they are usually based on a previous FPGA architecture that is not targeted at signal processing. Furthermore, these chips still have a higher degree of dedicated programming support and suffer from narrow data paths.

3.1.1 Streams

A stream is a set of words that are organized together as a group and arranged in a consecutive order. A stream can be comprised of data, programming information, or both. Since the Colt lacks on-chip memory, any data that enters the chip will clock through the entire chip in a finite amount of time. Data will clock in and out at the rate of one word per port per clock cycle. Since the Colt

is designed to do a few repetitive computations on many data items, it is natural to configure the chip to perform computations on streams of data. The lack of an extensive dedicated programming mechanism dictates that data and programming information for the Colt take the same basic path through the chip; therefore, a series of programming words can enter as a programming stream.

Streams can be further broken down into packets, sections of a stream containing programming information. Packet lengths are not fixed in the Colt architecture. Each of the following requires one programming packet: data port, crossbar node, and FU. The packet lengths vary for each of these, and in the case of the FU, they can vary depending on configuration. Packet lengths are always between one and eight words long. The word width for any data or programming word is the same as the width of the data path through the chip, sixteen bits.

3.1.2 Addressing

A single packet may pass through many different chip components before reaching its intended destination. In the meantime, those components are in a programming state. To keep undesired configurations from occurring, each component has its own address. The first word of a packet contains an address field that, when matched, will configure a component. Otherwise, the programming information is passed on. Address fields are six bits wide. There is also a special address used to broadcast and simultaneously program many components at once. This address consists of all ones. Crossbar addresses start at one, data ports at 24, and IFUs at 32.

3.1.3 Configuration Registers

Although there is no data memory within the Colt chip, there does need to be a minimal number of registers to store the configuration for each of the chip components. These registers are called configuration registers. When a component is configured, these registers are loaded with any programming information required to hold a configuration for that particular component. The crossbar nodes require no registers while the data ports require one each.

The FU and IFU configurations are stored in six or seven registers within the FU. These registers are accessed off the right input register. The FU state machine determines when the FU is being configured and asserts the load line of these registers. The path out of these registers loops back to the left input register through the IFU so that an optional constant can be programmed and held there.

3.1.4 Configuration Process

The process of configuring the Colt chip involves a flow-driven method. Packets are sent through a path within the chip to program respective components on the way. Each path may have one or more packets in its stream. The path may terminate within the chip, split up, or come out of the chip through a data port and continue to program other chips. More than one path may be used for configuration. In fact, all data ports can be used simultaneously as program inputs.

During configuration, the program pin on the data port is held active. This subsequently drives the program line of each component in the path to an active state as the first packet reaches that component. This is required to enable components to distinguish between data and programming

information. When the last word of the last packet has gone through, the program pin is de-asserted and data is sent through.

In order to program a component, the address must match, the program line must be asserted, and the programming data must be valid. Furthermore, the start-of-packet marker must be set correctly. If the program line is asserted and the packet address does not match the component, the component will pass the packet on in the direction it is already configured to pass data. The only exception to this is the FU, which can be programmed to pass programming information and data separately.

Data Ports

The data port must be configured before any other parts of the chip can be accessed. The data port packet consists of only one word. The bits are broken down in Table 3.1. The main bits here are the direction and mode bits. Bits eleven through six represent a mask used in the synchronized mode. A one in a position disables that respective port from being synchronized.

The data port can be configured externally or internally. If internally, then the chip program line must be asserted and the packet must come in on the chip input bus. If externally, the program pin is asserted and the packet comes in on the external pin bus.

Crossbar

The crossbar is configured by each individual crossbar node. These nodes, shown in Figure 3.1, can be configured to make a connection between the input and output buses and the input and

Table 3.1: Data port configuration word.

Bits	Function
15	Must be 1
14	Direction
13	Loop-Back
12	Sync Mode
11:6	Sync Mask
5:0	Port Address

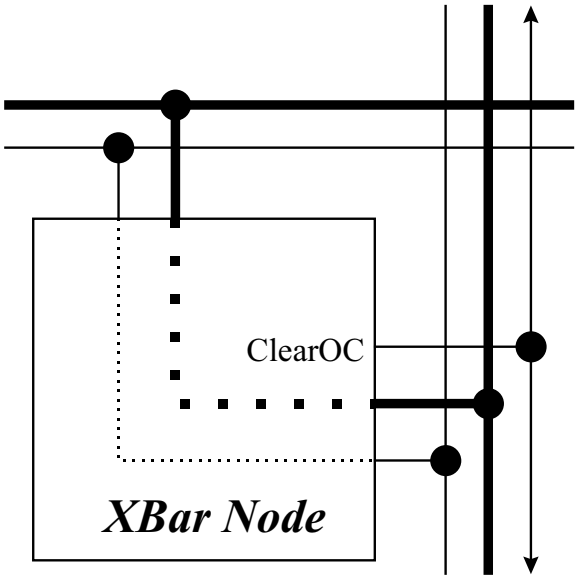

Figure 3.1: Crossbar node.

output program lines.

Since the crossbar inputs are fixed, and only one of them at a time can be configured to drive a certain output, there is no need for different addresses between rows as long as configuration is done carefully. Each column has its own address. When the one word crossbar packet comes in, the crossbar node will check the fields in Table 3.2 for a match and will enable the output bus and program line. To ensure that only one node is driving the output of a column at a time, there is

Table 3.2: Crossbar node programming word.

Bits	Function
16	Valid Bit must be a one
15	Must be 1
5:0	Column Address for Connection

a common bidirectional clear signal (ClearOC) that, when asserted by the driving node, will clear all other nodes in the column from driving the output.

IFU and FU Configuration

All configuration information for a single IFU and FU is stored within the respective FU. Since the mesh is composed of IFUs, mesh configuration can be simplified to just a series of FU configurations.

Table 3.3 shows all eight programming words for the FU. When the chip is reset, the IFU state machine allows programming information to reach the right FU input from any direction. The first word, the start-of-packet marker, is immediately read in to the FU from the right input register and sets up certain configurations within the FU and the FU state machine. The state machine then signals the configuration registers to load. The next seven words then take a path from the output of the right input register, through the six configuration registers, through the bus output of the FU, through the input ring of the IFU and back into the left input register of the FU. The path is taken through the left input register to program the optional constant there. Section 5.1.1 explains how this programming path came about as a result of verification. Figure 3.2 shows the programming path taken in the FU along with the configuration registers.

Table 3.3: FU and IFU programming words.

Word	Bits	Function
0	15	Must be 1
	14:6	Program Forwarding, Valid Bit Calculation
	5:0	IFU Address
1	15	Must be 0
	14:0	Left Operand Constant
2	15	Must be 0
	14:0	IFU Data Control Bits for FU Input and Skip Bus
3	15	Must be 0
	14:0	IFU Flag Control Bits for FU Input and Skip Bus
4	15	Must be 0
	14:0	IFU Flag Control Bits for FU Skip Bus
5	15	Must be 0
	14:10	FU Shift Configuration
	9:0	IFU Flag Control Bits for FU Skip Bus
6	15	Must be 0
	14:0	FU Flag Input/Output Selection Conditional Mode Select
7	15	Must be 0
	14:12	Left operand input selection
	11:0	ALU Programming (P, G, and R)

Once programming is finished for an FU, it continues to pass packets until its program line is de-asserted. The direction that packets are passed is determined by the first programming word. Packets can be passed to any combination of the four local neighbors. This flexibility allows streams to be split or routed in any manner in the mesh. The advantages of this include the ability to program an entire chip or series of chips with only one serial programming stream, and the ability to reconfigure portions of the chip at run-time while only affecting a minimal number of resources.

When the program line is de-asserted, valid data may be sent through for processing. The path that data takes from an FU is determined by the programming information in that and surrounding IFUs.

Programming information that leaves the mesh enters one of four rows in the crossbar. These packets can program the crossbar and can then be sent back through parts of the mesh, through the multipliers, or back to one of the data ports.

3.1.5 Reconfiguration

Once the program pin is de-asserted, valid data may be sent through a configured path in the chip. Resources that are not on this path may or may not be configured. These resources can be reconfigured through unused data ports in the manner described above. Paths that are processing data will remain unaffected as long as their resources are not programmed. The key to this ability is the fact that when a program pin is asserted, resources within the path go into a programming mode as the packets are passed to them; therefore, only the resources in the path are in the program

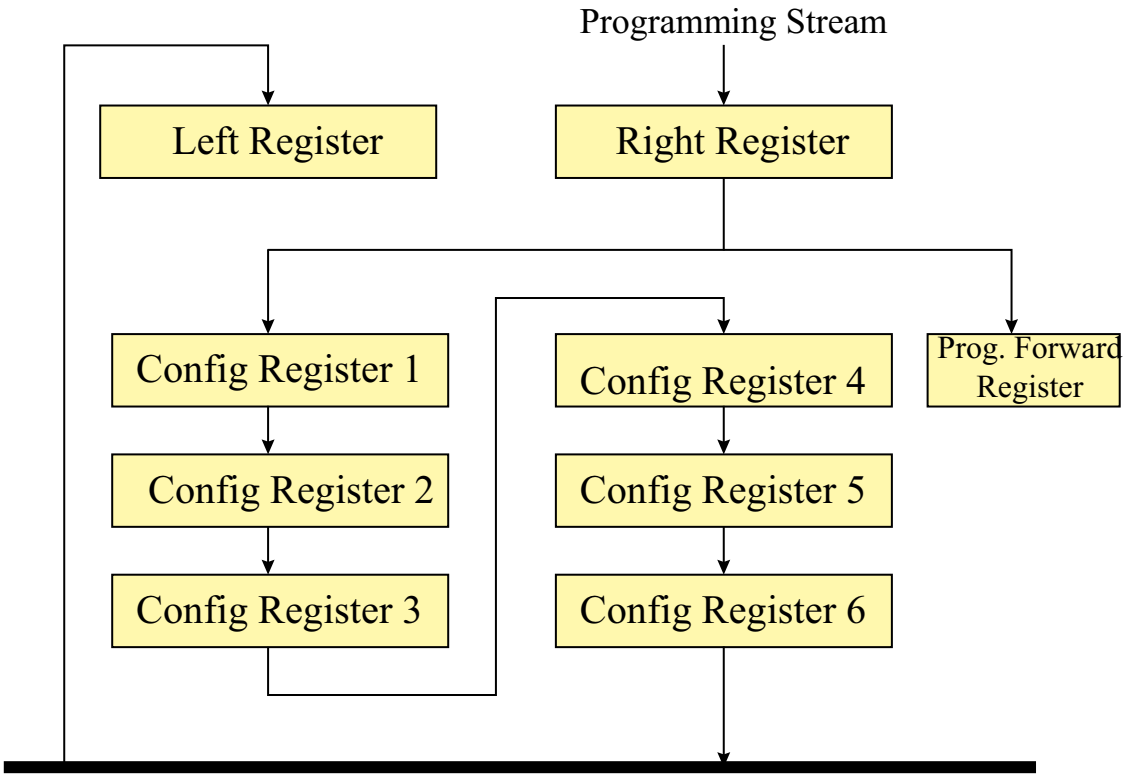

Figure 3.2: Programming path through the FU.

mode. Other parts of the chip can be in program or data mode. In contrast, other CCM chips put the entire chip into a program mode regardless of which resources are affected.

The high degree of configurability makes some special-case situations possible. For example, within the mesh, certain FUs in one path can be reconfigured from another path. This will affect the data stream for the first path, but can be used to reduce the programming latency between data sets if properly done. If there is a split in the data path, a new programming stream can reconfigure the resource used by one side of the split while leaving the other untouched and uninterrupted. Care must be taken in programming such situations, especially in timing the beginning of data or programming streams.

The number of configuration possibilities for even one Colt chip is quite large. The programmer is given a wide range of flexibility, but is also responsible for keeping the configuration error-free, especially during reconfiguration. Some configuration examples will be presented in Chapter 5.

3.2 Configuration Tools

Without the aid of specialized tools, configuration of a chip of this complexity is difficult and tedious at best. Throughout the development of the Colt chip, tools have been developed to aid in the configuration and verification process [6]. These tools have been developed as a hierarchy to allow for a various level of abstraction in verification. This also allows parts of the tools to be reused in the future and allows for ease in maintainability.

There are two main issues related to maintainability: relocation of configuration bits and changes in the configuration procedure and hardware. The bits in the configuration registers are

hard-wired to their respective control logic and units. These registers come off the main data path, so the locations of the bits are fixed in the programming words. During layout, it was determined that some of these bits required long routing due to their inefficient placement within the configurations registers; therefore, some bits were swapped around to reduce routing and area resources. If configuration information and test representations are hard-coded as ones and zeros, they would have to be changed manually each time a configuration bit changes. If they are represented at an abstract level, then only a change in the tool would have to be made, leaving the tests unaffected.

3.2.1 Simulation Files

The lowest level of programming requires signal stimulation: ones and zeros at the input pins. This is the desired end result from any tools or interfaces since this is the language of any computer hardware. For purposes of simulation or actual hardware verification, a set of test vectors are needed. For digital simulation purposes, these vectors are used in the simulation input files and are therefore the end output. For analog simulations, these vectors need to be converted to a different format.

3.2.2 Piece-Wise Linear Tool

Piece-wise linear (PWL) is a tool that converts a clocked group of vectors or signals into piece-wise linear formats to support various simulators. The supported formats include VHDL [3], SPICE [40], and STL, a common input language used in tools made by Cadence Design Systems [10]. These are the input formats used by the simulators throughout the development of the Colt. Other

```
# Initialize
# Path[0]
#     FU
#         ConfigReg[0]
LeftOperand = 1000100001100000
CLOCK
#         ConfigReg[1]
LeftOperand = 0000000000000000
CLOCK
#         ConfigReg[2]
LeftOperand = 0000000001011000
CLOCK
#         ConfigReg[3]
LeftOperand = 0000000000000000
CLOCK
#         ConfigReg[4]
LeftOperand = 0000000000000000
CLOCK
#         ConfigReg[5]
LeftOperand = 0000000000000000
CLOCK
#         ConfigReg[6]
LeftOperand = 0000000000010000
CLOCK
#         ConfigReg[7]
LeftOperand = 0000000000000000
CLOCK
```

Figure 3.3: Sample PWL code.

formats can be added to the PWL tool relatively easily. PWL reads a file that consists of lists of vectors or signals represented in binary format. Figure 3.3 shows an example of a PWL file used to test the FU.

3.2.3 Data Flow Compiler

The Data Flow Compiler (DFC) provides a level of abstraction above PWL and binary numbers. Signals can be logically grouped into units they represent, such as an FU. Furthermore, bit positions are replaced with actual signal names. Signal values can be represented in binary numbers or in valid text fields the signal can take on. DFC also supports path programming, allowing units in a path to be specified. As Figure 3.4 shows, the input to the DFC is text or text and numbers as described above. DFC generates PWL as output files.

3.2.4 Colt Graphical User Interface

Due to its array nature, configurations on the Colt chip are difficult to visualize and program even at the DFC level. The Colt Graphical User Interface (GUI) was developed to add a visual aid to the programming (Figure 3.5). The GUI uses graphical representations to help a programmer configure the mesh, the crossbar, and the data ports. Configurations in whole or part can be copied and saved, providing a higher degree of maintainability for Colt algorithms. The Colt GUI generates DFC files as output.

All three tools work together to aid the verification process. The Colt GUI provides the user interface, DFC provides the abstraction, and PWL provides the format support. Any changes in

```
DeclareFU BasicFU

      LeftInputSelect = Normal
      Lf = LoadAll
      P = Add
      G = Add
      R = Add
      OutputDelay = Disable
      FCInpSel = Zero
      FCInv = Enable
      FNInpSel = Zero
      FNInv = Enable
      ValidBitMux = FNOut
      FSInpSel = Zero
      FSInv = Disable
      CondMode = ALUOrZero
      FNOutSel = ALUSign
      ShiftCondInpSel = Zero
      ShiftCondInv = Disable
      ShiftType = ShiftLeft1

EndFU
```

Figure 3.4: Sample DFC code.

the simulators require only updates in PWL, while changes in the configuration bit ordering will only affect DFC. Changes in the configuration procedure or hardware may require changes to both DFC and the GUI; however, PWL would remain unchanged.

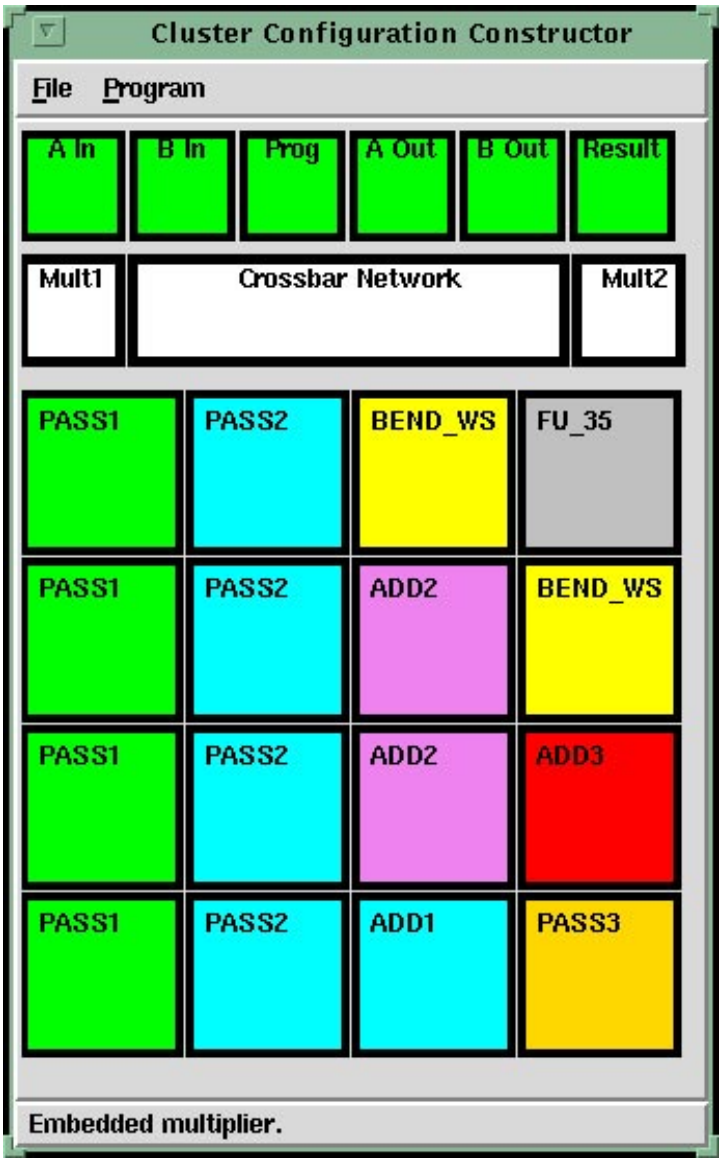

Figure 3.5: The Colt GUI.

Chapter 4

Verification

A significant effort has been undertaken to scrutinize the Colt chip for correctness. High complexity yields high performance, it also increases the possibility of failure at any level. Due to its increased functionality, failures in design or layout are possible. To insure proper operation of the Colt, verification must be performed at every level, be as exhaustive as possible and cover as many input and output combinations as possible. This is not a trivial task.

Verification involves designing a test, simulating it on a design, collecting data, and comparing this data to the expected result. Testing is then a subset of verification, but the two are closely related and the two terms used interchangeably. This verification, or testing, actually embodies several levels. Functional verification is concerned with logical and functional correctness of the designs of components under test. Temporal verification is concerned with timing issues. Of primary interest is preserving functional correctness at high clock speeds. Other verification procedures fall under general testing of the chip and power considerations. The testing is done at all stages of chip development: algorithm design, logic design, circuit design, layout, and silicon. This chapter focuses mainly on the first three stages.

4.1 Algorithms

The main purpose of the Colt chip is to obtain mathematical and logical results from a series of data streams. Just as important as the design of the chip are the designs of the algorithms that will drive it. These algorithms work within the architecture and hence provide a proof of concept for the architecture. If certain key functions cannot be efficiently mapped, the verification-design cycle takes over and changes must be made to the architecture.

Chapter 5 presents some algorithms that have been mapped to the Colt. These include arithmetic as well as DSP algorithms.

4.2 Functional Verification

Verification at the functional and logic level primarily requires digital testing. At this level, verification is concerned with correct functional results without regard to timing issues except for how it affects the logic. Testing digital functionality requires a digital logic simulator.

4.2.1 Digital Simulator

The simulator used for the Colt was the Verilog-XL simulator in the Cadence Design Systems (CDS) version 4.3.4. This simulator uses Verilog models at any level of hierarchy. This gives the user a high degree of flexibility, a wide array of choices within the simulation, and allows the integration of custom models into the simulation. This is especially useful for simulating special cases.

Some of the highlights of digital simulation include resistive models, signal delays, custom

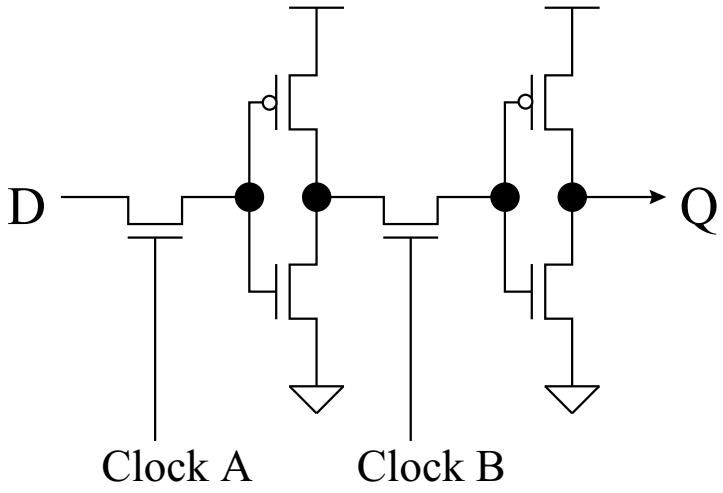

Figure 4.1: Pass transistor D-latch.

Verilog model blocks, and special case digital models that need analog properties. One of the advantages to using digital simulations was speed. Even a chip-wide simulation took only a few minutes and results could be viewed graphically within seconds.

4.3 Temporal Verification

Functional verification tests the circuit logic for correctness. It does not account for timing errors or incorrect results that may be the product of high capacitance, excess resistance, or other electrical properties; therefore functional verification is not sufficient. Temporal verification is required to ensure timing constraints are met. Furthermore, circuit design of critical paths and problem areas must be performed to increase the speed along these delay-critical paths.

As mentioned in Chapter 1, the Colt uses many space-saving techniques that have more stringent design constraints than complementary logic. The most widely used method is the use of pass transistor logic. One area where this saves a large amount of area is in latches. A conventional

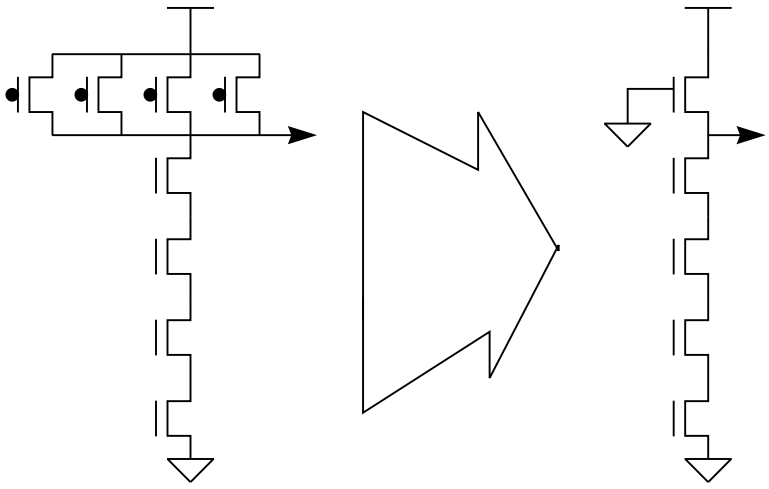

Figure 4.2: Area improvement for pseudo-nmos.

D-latch requires a minimum of four gates at four transistors per gate. As Figure 4.1 shows, the use of pass transistor logic for the D flip flop in place of complementary logic for the D flip-flop can cut the number of transistors in half or better. In a mesh of one hundred FUs with seventeen-bit wide data paths, this can amount to a significant area savings. The Colt architecture uses n-pass transistor logic. Although a logic zero can be passed with ease, a logic one will be degraded through an n-pass transistor. For this reason, CMOS inverters or buffers are periodically placed in the path to boost degraded signals. Placement needs to be determined by analog simulations.

Another method used to reduce transistor count up to 50 percent over CMOS logic is to use pseudo-nmos logic. Figure 4.2 shows this reduction. There are two concerns when using this technology. First, care must be exercised in the design of the pmos transistor size to insure it can be pulled down through the nmos tree. Second, the use of pseudo-nmos logic creates a constant current load between power and ground, increasing power consumption. The use of this technology must be carefully considered to offer a balance between area and power consumptions.

4.3.1 Analog Simulator

The analog simulator used for the colt is a SPICE-compatible CDS simulator called Spectre. This simulator can use SPICE or Spectre models. Spectre gives several advantages over SPICE including faster DC operating point convergence [10]. This is an important advantage when simulating large-scale circuits such as the FU or the IFU.

4.4 Approach

There are many advanced and complicated methods for verifying VLSI designs and layouts [27, 8]. Tool and time constraints allow only functional and temporal testing of the Colt chip. Even at this level, exhaustive testing is virtually impossible due to the complexity of the circuit. Circuits can be broken down into sub-circuits and tested individually to reduce the complexity while retaining an acceptable level of reliability [26].

An effective way to approach verification of sub-circuits is to take a bottom-up approach to testing. Due to the fact that the design was implemented partially in a bottom-up approach, verification of components followed this implementation. Digital and analog simulations at a functional level were performed at each level of design starting at the gate level.

4.4.1 Mesh Verification

Mesh Verification began with low-level building block circuits of the FU. Low level gates, multiplexers, decoders, and latches were tested at the analog and digital levels. Basic problems associated with pass transistor logic were identified at this time.

The voltage drop through an n-pass transistor should be the same as the turn-on voltage as long as the gate voltage was near full value [42]. Simulations showed a significantly larger drop when the gate voltage was lower than five volts. This degradation of the signal was unacceptable; therefore it was determined that output from pass logic must be buffered before being used as input for other pass logic blocks.

Pseudo-nmos circuits were tested for power consumption. A pull-up transistor in an average circuit drew anywhere from 40 to 60 μamps. This was acceptable for some applications, but proved too much for repeated blocks. For example, there are around 200 address comparators. If each of them drew 46 μA, the total would be close to 10 mA; hence, pseudo-nmos address comparators were scrapped in favor of fully complementary ones.

Dynamic latches were used exclusively throughout the Colt chip. While saving space, these latches require precise clocking and cannot be operated under a certain minimum frequency. These latches also presented a unique problem for digital simulation. A charge storage model was developed to allow correct digital simulation. The charge storage model was attached to the latch at the input to each inverter (see Figure 4.1). The first clock (Clock A) passes the input to the inner node where charge stored on the drain and source capacitances holds the value until the second clock (Clock B) passes it out.

Due to concerns over timing of the latches and over clock skew problems, a clock generation circuit is used throughout the Colt chip. This circuit has critical timing constraints to insure proper operation of the latch. There must be between one and three nanoseconds of time between the two clock strobes when they are both low. This allows the value to be latched instead of passed

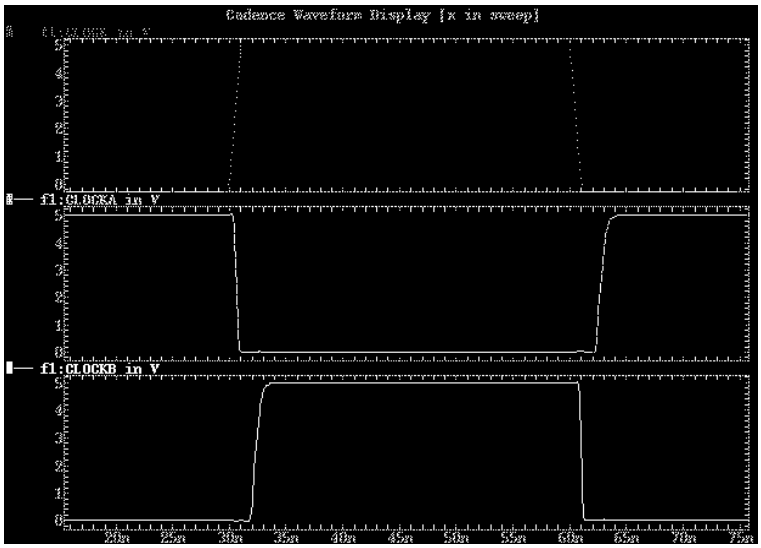

Figure 4.3: Simulation results of clock generation circuit.

Clock input is on top with Clock A (middle) Clock B shown below.

Table 4.1: Latch properties.

Property	Value
Propagation Delay T_prop	2.5ns
Set-up Time T_su	0.3ns
Hold Time T_hold	0.5ns
Cycle Time T_c	3.3ns

through immediately. Figure 4.3 shows the simulations results of the clock generation circuit with the proper delays. The delays in the clock determine timing properties for the latches in the design. These are shown in Table 4.1.

ALU Critical Path

Major data path elements were to be tested next. For the ALU, the critical path was to be reduced. With all transistors minimum size, the delay through the ALU was 12 nanoseconds. IFU

Table 4.2: ALU transistor sizing and delay.

Combination Number	Tx width in λ				Critical Path Delay (ns)
	1(P)	2(N)	3(P)	4(N)	
1	4	4	4	4	11.9
2	16	16	8	8	6.6
3	8	8	8	8	7.8
4	8	8	16	16	8.6
5	16	16	16	16	6.6
6	32	32	16	16	5.7
7	32	32	8	8	6.2
8	64	32	16	16	5.6
9	32	16	16	8	6.1

and other network delay would comprise a significant portion of the total target delay for one clock cycle of 20ns. Delay through the ALU was therefore unacceptable and needed to be reduced to under 10ns.

The primary means of decreasing the delay of the critical path through the ALU was to iteratively identify this path and then to reduce it through transistor sizing and layout techniques. The critical path consisted of the first set of multiplexers followed by the carry logic through the Manchester chain (Figure 4.4). The speed-up through the multiplexers was found to be small and costly, but sizing of the transistor widths in and around the critical path produced significant speed-up due to reduced channel resistances [42]. Table 4.2 shows ALU delays for various combinations of transistor sizes corresponding to Figure 4.4. A trade-off between speed and area resulted in the selection of the third combination, a 7.8ns worst-case delay.

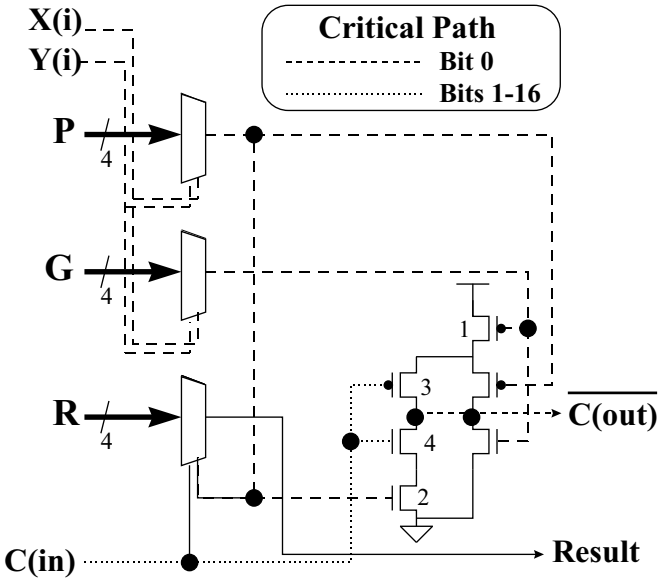

Figure 4.4: Bit-slice of ALU critical path.

Signal Degradation

Due to the use of n-pass logic, signal levels through the data path dropped significantly while going through the shifter, ALU, and conditional unit. Voltage levels for a logic one were around 3.5 volts at the bus output of the FU. These signals each drive up to 8 other inputs. The IFU was redesigned to buffer these signals with large "donut" or "round" inverters. Inside the FU, the signal degradation was not enough to require buffers internal to the data path.

Speed

Flexibility of the Colt architecture allows IFUs to be strung together to operate on data paths wider than 16 bits. While data signals are pipelined, control flags are not. Although shift and conditional flags can be passed from one IFU to another, they cannot be propagated through an

indefinite number of IFUs. For this reason, they are not part of the critical path. The carry signal, however, can be part of the critical path and hence can increase the worst-case delay in the Colt chip. For example, suppose that a 64-bit addition is to be performed across four IFUs. The worst case delay for this operation can be up to four times what it normally is for a single IFU since the carry signal may have to propagate through four, not one, IFUs. Since the clock will not support this delay, erroneous data will result. Verification identified this problem and the carry signal was then pipelined to control the delay and retain the designed clock speed.

Capacitance

Excess capacitance due to transistors or wiring can cause signals to propagate slowly. This is of greatest concern in the IFU where long wires are prevalent. Since the architecture allows for signals to skip across an arbitrary number of IFUs, this capacitance can be large if a signal is to skip across an entire mesh. Back annotation and calculation of capacitances in the layout were used to accurately simulate delays across more than one IFU. Figure 4.5 shows a circuit used to model delays in the skip busses across up to four IFUs. The capacitances were calculated using the worst-case layout dimensions of the wires. For example, one part of the bus is 2500 lambda long, so the total area is 2500 by 4 lambda, or 1600 μM^2. According to MOSIS [38] the worst-case capacitance for metal routing is 31 aF/μM^2 with an additional 21 aF/side for fringe capacitances. For a three-part bus, this gives a worst-case capacitance of 0.65pF. After connection capacitances and transistors were added, an accurate model of the bus was simulated for timing considerations. The corresponding delays from simulations are shown in Table 4.3. Although a mesh can be

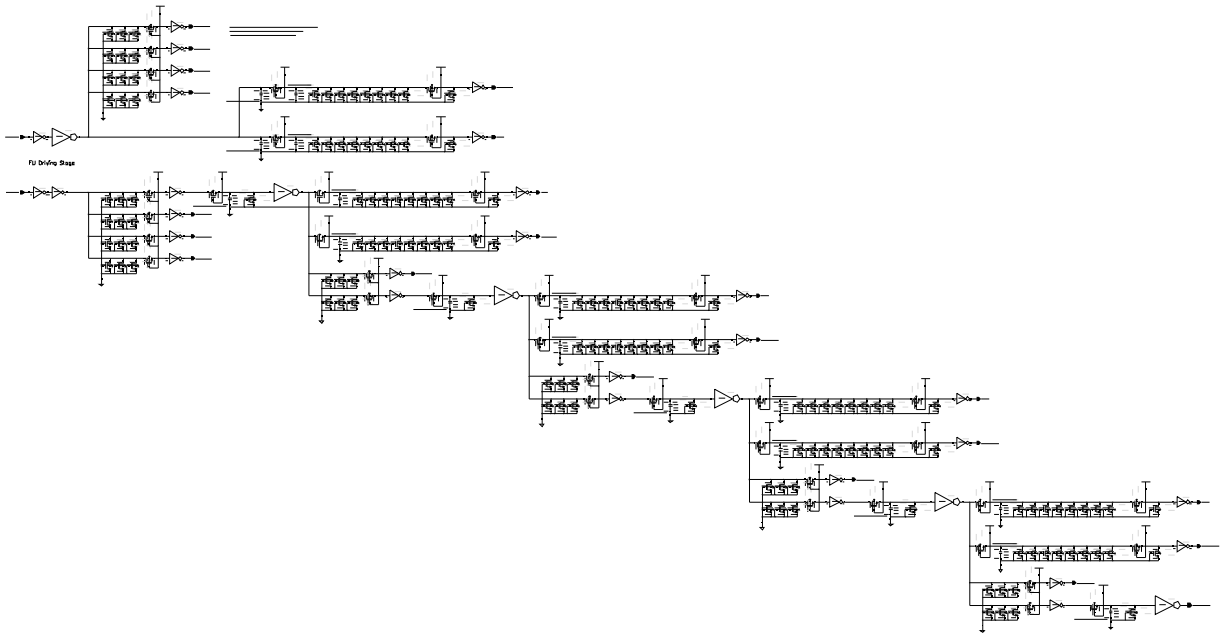

Figure 4.5: Skip and local bus model.

programmed to allow signals to skip over an arbitrary number of IFUs, a 30ns clock dictates that for the Colt chip, the reliable limit is four IFUs.

4.4.2 Crossbar

Crossbar verification centers primarily around the crossbar node, since the crossbar is comprised of an array of these units. The biggest problem encountered in the crossbar was the ability for

Table 4.3: Bus Model simulation results.

Output	Delay from Input (ns)
Local bus	4.2
Skip bus 1	5.8
Skip bus 2	6.7
Skip bus 3	7.5
Skip bus 4	8.3

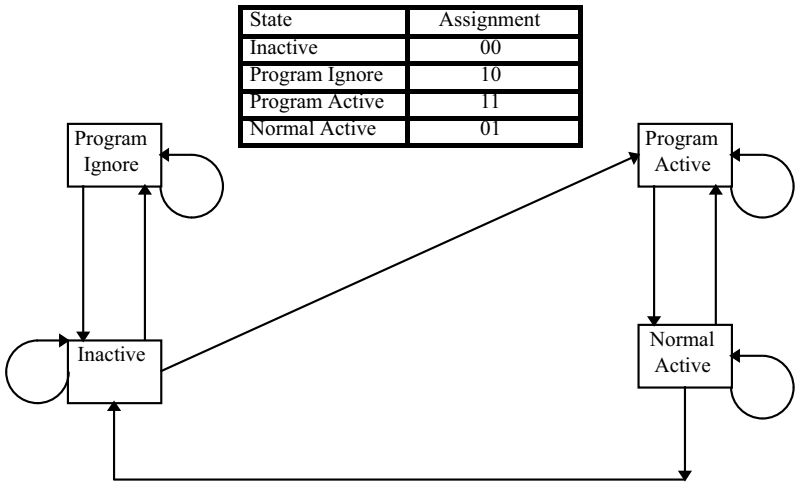

Figure 4.6: Crossbar node state machine design.

the clear signal to be driven both internal and external to the node. Figure 4.6 shows the possible states for the crossbar node. One of the difficulties associated with the crossbar node is the fact that they communicate and therefore depend on each other. Verification ironed out communication issues in early versions of the nodes.

The other issue for the crossbar centers around capacitance. Similar to the IFU simulations, the crossbar wiring capacitance was also calculated and simulated. For the crossbar, each wire is 2500 lambda long. Using the same calculations used in Section 4.4.1, the input and output busses were found to have a worst-case capacitance of 540 fF associated with each wire. This carries a corresponding propagation delay of about 5.8ns.

The other error found that required significant redesign of the crossbar centered more around programming. State three in figure 4.6 was added after verification showed that it was possible to inadvertently program multiple nodes in the crossbar. This will be discussed in Chapter 5.

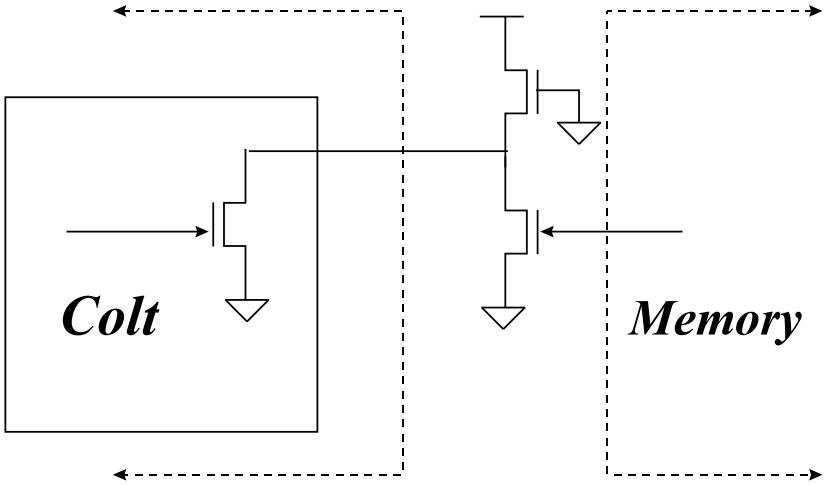

Figure 4.7: Data port memory interface model.

4.4.3 Data Ports

The data ports were the last major components of the chip to be tested; however, they proved to be one of the most difficult. Due to the high configurability of the ports, their interface to memory, and their ability to pass data or programming information in either direction, the data ports are much more complicated than they may seem.

To properly verify the data ports, there was a need for an interface with a memory unit or a mock memory unit. Careful manipulation of the inputs along with a model could serve this purpose, or the unit could be a Verilog behavioral model. Figure 4.7 shows the hardware model. The control signals attached to the data port are bidirectional, and are open-drain as to be driven internal or external to the chip. The pull-down circuit in the figure models how the memory unit would control this signals.

The data port state machine proved to be large and difficult to work with. One of the design

compromises made to ease the verification-design cycle was to model the state machine logic with a Verilog behavioral model. This made diagnostics and changes on the logic possible with a reasonable amount of effort. Furthermore, these equations will make it easier to use tools for automatic layout of this state machine [31].

The data port configurations in both directions were tested from both directions. Synchronization modes as well as all configurations were tested. The data ports were not tested together until the chip as a whole was tested. The next chapter goes into detail on full-chip testing and configurations.

4.5 Clock Speed

One of the main goals of temporal verification was to make sure the Colt would run at the designed speed of 50MHz. To achieve this goal, the critical path in the chip was identified and iteratively simulated and redesigned. The critical path should be found in the IFU Mesh. Delay between the cross bar and data ports was under 10ns. Delay from the bottom of the mesh to the crossbar and from the crossbar to the top of the mesh fell well under the target clock of 20ns, as were delays through the multiplier. The worst case delay was found within the mesh, as it should be. The only place that data is clocked in an IFU is at the top of the data path in the FU. The critical path therefore goes from one register to the next. The critical path, shown in Figure 4.8, goes from the left input register, through the shifter, through the first bit of the ALU and then through the Manchester carry chain (Section 4.4.1). The input to the shifter can come across either the conditional or shift flag skip busses. This can add skip delay to the critical path. Since the

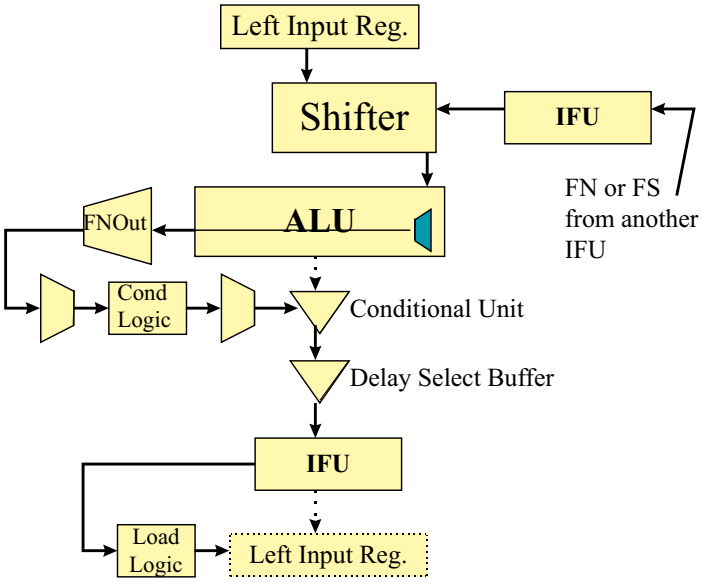

Figure 4.8: Colt chip critical path.

conditional input can depend on the ALU output, the path goes out the ALU carry-out, through the FN select logic, through the conditional unit select logic, through the conditional unit and then through the delay buffers. At this point, the path goes through the IFU which has variable delay depending on the number of skips the path takes (Table 4.3). The critical path then goes back in to the left input register via an input buffer and through the load select logic. The load select logic delay dominates the buffer delay and is therefore the critical path. Table 4.4 shows incremental delays through this path and the total delay. The clock speed is dependent on how many IFUs a signal can skip through at one time; however, if this is kept to three and the shift input is kept to neighboring FUs, the Colt can be operated at 45MHz.

Another area critical to full speed clock operation is the open-drain data ports. These were shown by [6] to operate fast enough to run at clock speed and to remain within pad boundaries for

Table 4.4: Worst-case critical path delay.

Component	Worst-case delay(ns)
Left Reg Load Sel.	0.6
Left Input Register	2.5
Shifter	0.5ns
Shift Skip	4.2
ALU	7.3
13to1 Mux	0.96
2to1 Mux	0.24
Cond. Logic	0.75
Cond. Unit	0.24
Delay select	0.24
IFU Delay	4.2
Total Delay	21.75 (46.0 MHz)
Each Data Skip	+0.8 (-1.6 MHz)
Each Shift Skip	+0.8 (-1.6 MHz)

power dissipation.

4.6 Summary

The procedures outlined in this chapter were thorough in their attempt to find potential errors in the design or implementation of parts or all of the Colt chip. The tests also produced timing information used to verify proper functionality at high clock speeds. The Colt chip design has been verified and found to be functionally and temporally correct. The VLSI implementation was checked against the design when complete. Blocks of the implementation were also netlisted and simulated using the same input files used for design [31]. These tests produced results identical to the design simulations. The Colt chip verification effort has shown the design to be ready for fabrication. Given the process data from the foundry [38], the manufactured chip should operate

close to the targeted speed of 50MHz. The speed will depend on algorithms' dependence on the skip bus.

Chapter 5

Algorithm Mapping

The highest level of verification performed on the Colt architecture were full chip simulations. These tests are not only concerned with proper functionality of the major blocks of the chip, but also provide conceptual verification. They are designed to find out if the Colt can be programmed to perform the various algorithms it is designed for. Performance and efficiency are other factors to analyze from these tests. Several key arithmetic operations that any DSP chip needs to support are mapped to the Colt architecture and the Colt chip, as are two sample DSP algorithms used by the Virginia Tech GLOMO team.

Since the Colt chip is primarily a verification chip, it does not have a full complement of resources that should be available to a chip that the Colt architecture is designed for. For example, the mesh in this type of architecture should be at least an array of eight by eight IFUs. There should also be higher I/O capability and more multipliers. However, it is still possible to learn much about the Colt performance and ability through the following configurations. When applicable, algorithms have been designed for a general mesh architecture, but have been simplified or partitioned to fit the Colt. The ability to perform these functions on the Colt architecture is of higher importance than whether or how efficiently they can be mapped to the Colt chip itself.

5.1 Full-chip Testing

The following tests served a dual purpose. On one level, they were used to test the Colt chip in both program and data modes. Verification of the data paths connecting major components, addressing, and configuration testing took place. Of interest at this level was functional verification as well as any architectural design problems.

5.1.1 Configuration

Chip configuration was tested along different paths. A few simple algorithms were changed to configure various components of the chip differently. For example, output data ports can be configured from the outside or the inside. Two variations of a configuration can program the output data port directly from the outside or as the last resource from the inside; however, the configuration information must be changed and could increase the time the chip is in programming mode. For example, if programmed from the inside the chip must be in program mode for an extra clock cycle but there would only be one programming stream. If programmed from the outside, the input port *and* output ports can be programmed simultaneously; however, two programming streams would be required.

The one configuration problem found at the chip level concerned crossbar programming. Due to the fact that inputs to the crossbar are fixed, addressing is the same across all rows. This means that each column has its own address. A problem occurs when a programming path goes through the crossbar on one row, through the mesh and back through the crossbar on another row. The programming packet for the *second* row must first go through the the first row. When it goes

through the first row, it will program a connection to another column. This can cause some serious configuration problems since configuration information will be broadcast incorrectly. There were two solutions to this problem: give each crossbar node its own address, or change the way the nodes are programmed.

Since the address space was only 6 bits, only 64 unique addresses were possible. A 12 by 16 crossbar alone would require 192 addresses. This would require new address space and a large amount of redesign. The first solution was not realistically possible. In the second solution, a wait state was added to the crossbar node state machine and each row still used the same sixteen addresses for the respective columns. Now when a node is programmed, all other nodes on that row go to an idle state where they cannot be reconfigured until the program line is de-asserted. If a programming packet for another crossbar row goes through this row first, the node for that packet cannot be programmed since it is in the inactive state. This packet is then passed by the first connection and goes on to properly configure another node. Once program is de-asserted, any node in the wait state goes back to the inactive state where it makes no connection until reconfigured. This assures that in one program stream, this problem will not occur.

Another subtle problem area involves programming of the mesh. When configuring a path through the mesh that ends (ie. IFU configuration is the last packet on the path), the last programming word is clocked into the last IFUs right input register. This word should actually be in register zero. When other programming information follows, this word is effectively pushed in normally; however, in this case, an extra "false" programming word should be added to push the last word into place.

The programming path described in Section 3.1.4 came about as a result of verification tests. The previous path allowed programming information to enter both left and right inputs. Errors were found with this method. The first seven programming words came off of the right input register while the last went to the left; however, the words on the right were one register behind due to the extra right input register. The solution was to either require an extra programming word to "push" the data through to the configuration registers, or to redesign the programming path. Since programming latency was a concern, the path was redesigned to its current configuration.

5.1.2 Data

Full-chip data mode testing was concerned with insuring the integrity of input and output data through the entire chip; specifically that the components remained in the data configuration and properly processed the data. The only problem found was with loss of data words in the mesh when the IFU skip bus was used. This problem occurs as follows: Programming packets passes through a number of IFUs. One or more of the IFUs is programmed to skip data. When the program line is de-asserted, valid data enters the first IFU, right behind the program stream. Since the program stream takes a full clock cycle to go through each skip IFU, the problem occurs when the data attempts to skip ahead as shown in Figure 5.1. The result is that for each IFU that is configured as a skip along the programming path, one valid data word is lost.

This problem is compounded in the event that the programming stream takes a longer path than the data. For example, Figure 5.2 shows a program stream that "snakes" through the mesh. The latency for this stream is all sixteen IFUs, or sixteen clock cycles. Notice, however, that the

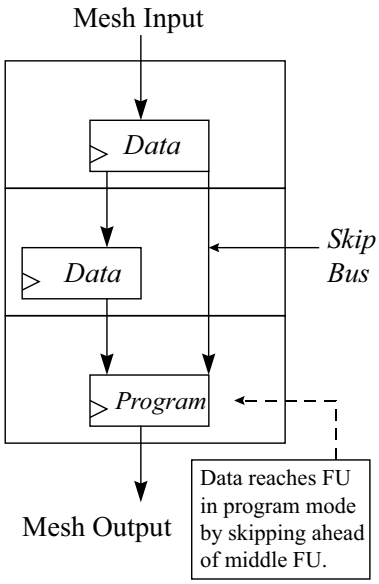

Figure 5.1: Data latency due to IFU skip.

data comes in through the top of the mesh and proceeds straight down. Until the last IFU is finished programming, some data will be lost. Therefore, up to twelve data words can be lost.

Since the Colt architecture is flow driven, the solution cannot be to simply add invalid data. A predetermined number of valid data words will be lost. It is the responsibility of the programmer or support tools to pad the beginning of the data stream with an appropriate number of valid data words to insure that no data is lost.

Run-time reconfigurability also presented some problems with timing of data and programming information that required careful planning. Section 5.3 gives a runtime reconfigurable example.

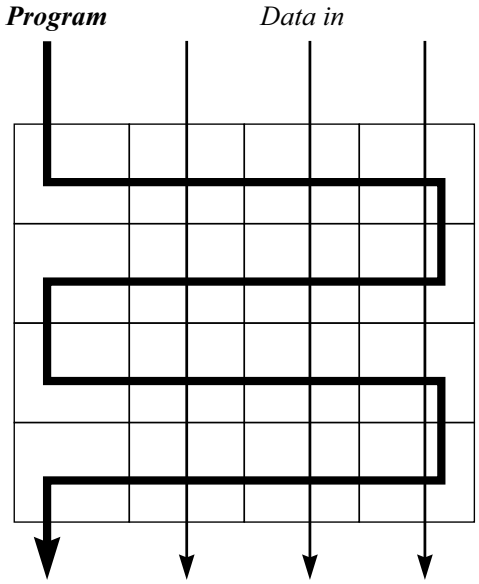

Figure 5.2: Data latency due to configuration path.

5.2 Algorithm Testing

The other purpose served by the full-chip tests was to verify the functionality of a variety of DSP algorithms. In most cases, simply mapping the algorithm was a challenge. The tests could answer many questions: Is it possible to perform this function on the Colt? Is this configuration allowing the Colt to do what it is designed to do? Are the results correct? These questions will be answered for each algorithm in the following sections.

5.2.1 Arithmetic Algorithms

The fundamentals of any DSP algorithms will involve a fair amount of arithmetic computations. These computations tend to be repetitive and simple in nature: adds and multiplies. Occasionally there will be subtractions, divisions, or other arithmetic functions. The Colt architecture was

designed to be most efficient at those computations with the highest frequency. The dedicated multipliers and reconfigurable ALUs make multiplication and addition an easy task. But in addition to this, the Colt has to support other functions and tasks reasonably.

There are many different variations on number system representations [28, 30]. Although these representations can be mapped to the Colt, it was designed primarily to support conventional binary two's complement. This will be the number system used in these examples. Furthermore, the targeted DSP algorithms use 12-bit fixed-point values; therefore, the following algorithms focus on fixed-point solutions. The colt chip supports floating point arithmetic. In most cases the algorithms would simply add shifting resources for alignment and normalization.

Addition, Subtraction, and Boolean Operations

Addition, subtraction, and boolean function are a relatively straight-forward mapping in the colt architecture, made possible by the reconfigurable ALU. Figure 4.4 shows a bit-slice of the ALU. The three programmable terms (propagate (P), generate (G), and result (R)) are used as inputs to three multiplexers. The result is selected by carry in and the P term while the carry out is selected by carry in, G, and P terms according to the equation $CarryOut_i = G + (C \cdot P)$. Any boolean or arithmetic function can be mapped to this ALU. Additions, subtractions, logical functions, pass left term, and pass right term are just some of the functions that have been implemented on the ALU. To implement a function, a truth table is made with the desired inputs and outputs. The P term can then be calculated from the carry in and desired output. The R term falls out as a result of this and the G term can be calculated from the above equation.

Table 5.1: Truth tables for ALU subtraction.

C_{in}	X	Y	R	C_{out}	P	G
0	0	0	0	0	1	0
0	0	1	1	1	0	1
0	1	0	1	0	0	0
0	1	1	0	0	1	0
1	0	0	1	1	1	0
1	0	1	0	1	0	1
1	1	0	0	0	0	0
1	1	1	1	1	1	0

C_{in}	P	R
0	0	1
0	1	0
1	0	0
1	1	1

Table 5.1 shows an example of deriving terms for a subtraction operation. For the input combinations C_{in} and the two operand inputs X and Y, the desired results are filled in for the ALU result and C_{out}. From the C_{in} and result columns, the P term can be filled in. From the other combinations and the above equations, the four-bit G terms can be filled in. Note that since the X and Y inputs are repeated, the P and G terms that depend on them are also repeated in the top table. Using the resulting P term and the carry in, the R term can be calculated as shown in the bottom table. The resulting terms for some ALU configurations are shown in Table 5.2. The three terms for subtraction can be seen in the truth tables when read from bottom to top. If only one of these functions is desired on an IFU, other data path resources can be programmed to simply pass values. Furthermore, several IFUs can be strung together to increase the effective width of the data path. Figure 5.3 shows an implementation of a 32-bit addition across two IFUs. The third

Table 5.2: ALU term configurations.

Function	P	G	R
	(in decimal)		
Addition	6	8	6
Subtract X-Y	9	2	9
Complement X	3	0	A
Complement Y	5	0	A
Pass X	C	0	A
Pass Y	A	0	A
NAND	7	0	A
AND	8	0	A
NOR	1	0	A
OR	E	0	A
XOR	6	0	A
XNOR	9	0	A
Shift X left	0	C	C
Shift Y left	0	A	C

IFU can be used if the carry out is required outside of the mesh. Since control flags are internal to the mesh, the carry flag must be shifted or added in to another operand and taken out out through a bus output at the bottom of the mesh. Sine the carry out of an FU is latched, there will be a one cycle latency for the carry in to the most significant word (MSW). The first row of FUs aligns the MSW to the least significant word's (LSW's) carry out by delaying it one cycle through the optional delay. Using the carry skip bus, the carry out of the MSW is aligned with the output by skipping ahead one row.

Division and Multiplication

Division and multiplication are some of the most time-consuming arithmetic operations performed. Multiplication also happens to be one of the most frequent operations for DSP algorithms.

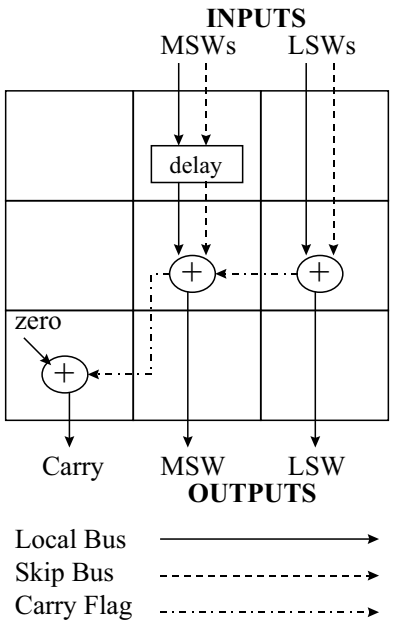

Figure 5.3: 32-bit addition mapped to the Colt.

Unlike addition, which only requires one IFU for each 16 bits, multiplication and division require at least two to three IFUs per bit. For these reasons, the Colt chip employs a dedicated multiplier while the stallion employs two. While the multiplier on the stallion will do both multiplication and division, the multiplier on the Colt does only a 16 bit by 16 bit multiply [6]. A divide algorithm was developed to cover this function and to show whether an array-type restoring algorithm can be mapped to the Colt architecture.

Many ways have been presented to increase the performance of division computations including high radix and SRT division [28, 5, 36]; however, these methods invariable require specialized hardware, communication paths, or resources that the Colt does not support. Furthermore, systolic architectures [14, 25, 12] tend to function on a bit-wide data path and configure all PEs the same. Solutions for the Colt must be novel in use of resources.

One of the reasons for the complexity of the divide function is the number of operands involved: a dividend N, a divisor D, a quotient Q, and a remainder R. These operands satisfy the equation $\frac{N}{D} = Q + R$. For the division algorithm, a sequential shift and subtract algorithm was found to be the most efficient use of resources. The quotient Q is calculated one bit at a time with q_1 being the most significant bit (MSB). The least significant bit is q_n where n is width of the operands, in this case, 16. In a given step i, the dividend is subtracted from the remainder. If the result is a positive number, q_i is a one. Otherwise, the subtraction is negated and q_i is a zero. This process iterates n times and gives a final remainder R and a quotient Q. This algorithm has been proven to give the correct results [28].

The two main variations on this algorithm are the restoring and non-restoring variety. The difference between the two is how they negate a subtraction that resulted in a negative number. The non-restoring method leaves the remainder negative and attempts to add it back at some later iteration. The restoring method actually restores the value of the remainder before subtraction. The former requires a run-time selection between addition and subtraction. Obviously the ALU does not support this feature, so an addition *and* a subtraction must be done concurrently and another IFU select between the two. The latter requires only a way at run-time to restore a value previous to the subtraction. The Colt easily supports this through the conditional unit. The sign bit of the ALU is one of the input options to the FN flag. When the conditional unit is in Mode 1, the FN flag is used to select between the ALU output or the right input register. If the remainder comes in on the right input register, the restoring method of sequential division can be easily mapped.

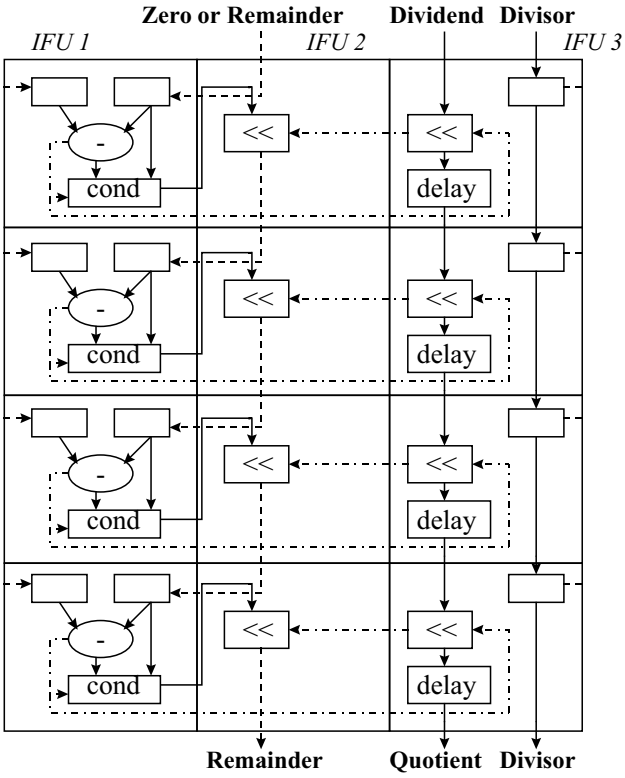

Figure 5.4: 16-bit by 16-bit division mapped to Colt.

Figure 5.4 shows the mapping of a fixed point 16-bit restoring division to the colt. There are only three different IFU configurations in this mapping. The first type performs the conditional subtraction, the second performs the shift, and the third stores the quotient. The auxiliary output of the second IFU is used to broadcast the divisor to the first IFU for subtraction. The divisor is stored across 32 bits between the second and third IFUs. The sign bit of the ALU represents q_i for that step. It gets fed around to the third IFU on that row as part of the left shift of the divisor. When the computation is finished, the second IFU column will output the remainder while the third will output the final quotient. Since the data path passes through the first and second IFUs before the next step (or row), the third IFU must utilize the optional delay for alignment purposes.

A full implementation may require additional IFUs at the beginning or end to ensure against error. For example, some check for $D \neq 0$ is in order. If negative values are used, some logic for adjustment may be in order. Since this implementation uses a 16-bit dividend in a 31-bit operand and only a 16-bit divisor, overflow is not a concern. If, however, this was used as a full 31-bit by 16-bit divider, a check for overflow conditions is needed, namely that $X \geq 2^{n-1} \cdot D$.

Square Root

Square root extraction algorithms are generally similar to division algorithms [24, 28]. Given a radicand X, the square root Q and a remainder R can be extracted iteratively. Shift and subtract algorithms similar to the division algorithm above have been presented [28]. The quotient at each step i is shifted left once, a 2^i factor added, and this sum subtracted from the left-shifted radicand. A modified division mapping can be used here. The complication arises from the term to be

subtracted. The quotient has to be shifted and added before it can be subtracted. The division algorithm required a fixed number to be subtracted. This complexity increases the number of IFUs per bit to at least 5 per bit. At this point, it may be worth the extra hardware to look at a higher-radix scheme.

High-radix square root extraction has been presented in [7, 15, 11, 24, 28]. High-radix operations attempt to reduce the computational delay by processing two bits at a time instead of one. The cost of this reduced delay is increased hardware. For a given step in the configuration for the Colt, the high-radix hardware may be more than for a conventional step; however, the interest is not in how much hardware is utilized but how many IFUs are required. The Colt was found to have the resources to reduce the number of IFUs by implementing a pseudo-radix-4 square root scheme [29].

In this algorithm, two bits are processed at a time. The equations for one step of this algorithm are

$$R(i+1) = (R(i)\&\&X(i)_{ab}) - (Q(i)\&\&01)$$

$$X(i+1) = X(i)_{c..0}\&\&00$$

$$Q(i+1) = Q(i)\&\&q_i$$

where R_i is the remainder, $X(i)_{ab}$ is the two high-order bits of X_i, $Q(i)$ is the quotient and q_i is a zero if the subtraction results in a negative number.

An example of this algorithm is shown in figure 5.5. To take the root of the number 1111 (15 decimal), first subtract 01. The first bit of the quotient is a 1. the remainder is 10. Concatenate the next two bits of the radicand to get 1011. Subtract Q concatenated with 01, or 101. Another one is concatenated to the quotient for an answer of 11 (3). The final remainder is 0110 (6).

```
   1 1   = 3 (Quotient)
√1111
-01
 1011
 -101
  110   = 6 (Remainder)
```

Figure 5.5: Square root algorithm example.

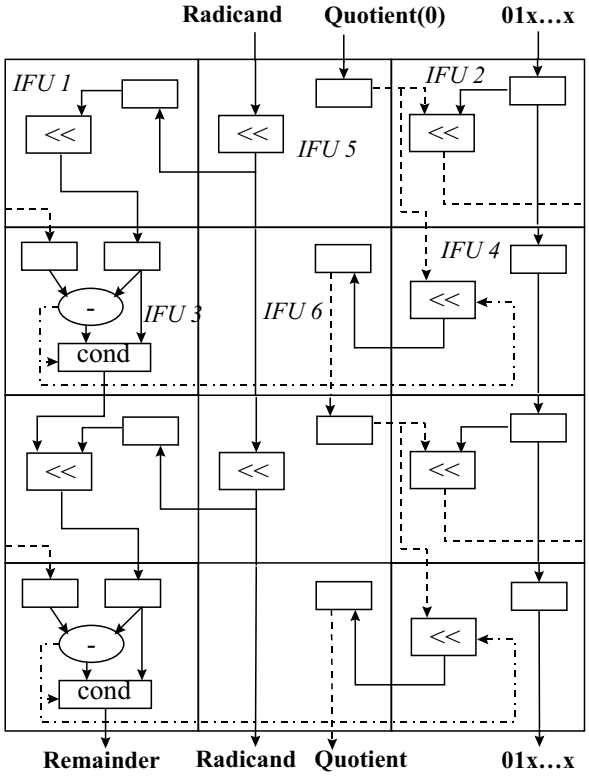

Figure 5.6: Square root on Colt

Figure 5.6 shows the mapping of one step of this algorithm to the colt. There are five different IFU configurations. The first IFU concatenates the remainder and the quotient. The second IFU is similar to the division subtract and compare IFU. IFUs 2 and 4 store the answer and generate the number for subtraction while IFU 5 in the middle column stores the radicand and supplies the two bits for concatenation in IFU one. This algorithm requires one row of three IFUs per bit in the operation. An 4-bit solution can be mapped to one Colt and a 16-bit solution can be mapped to a Stallion.

5.2.2 DSP Algorithms

Code division multiple access (CDMA) communication systems, such as the one implemented in the GLOMO project by the Virginia Tech MPRG, are becoming a popular replacement for traditional frequency and time division multiplexing systems. CDMA gives the advantage of higher bandwidth and less noise at the expense of more computations. CDMA was a main factor driving the Colt research [6]; therefore, an algorithm for this application is a good candidate for testing the Colt architecture.

Matched (FIR) Filter

Before the application is introduced, a small amount of CDMA background is necessary. In CDMA, multiple users transmit at the same time across the entire communication spectrum. A spreading code associated with each user is used to distinguish between channels. The spreading code is a fixed-length word of 1s and -1s. Each data bit is multiplied by the spreading code forming

a new vector the same length as the spreading code. This vector is then transmitted along with other users' data. The spreading codes for all users are chosen to be orthogonal so that the signals cancel and the transmission is low-level noise. On the receiving end, there must be a mechanism to decode the original data. This is done with a matched or finite impulse response (FIR) filter. The FIR filter is encoded with the spreading code of the desired channel. For a spreading code of length n, an n-tap FIR filter is required. The operation of the filter can be mathematically expressed as:

$$y(n+1) = \sum_{k=0}^{N-1} x(n-k)w(k+1)$$

where y(t) is the output, x(k) is the input, and w(k) is the transfer function or weight given by the spreading code. The digitized input signal goes serially through a series of register stages. Each register value is multiplied by a +1 or -1 according to the weight $w(k)$. All the numbers are then added up and produce the FIR filter output for that clock cycle. This is shown in figure 5.7. When a data bit is aligned to the spreading code a spike in the output of the FIR filter will occur. A negative spike represents a -1 (logic 0) and a positive spike represents a +1 (logic 1).

The CDMA mobile receiver used by the MPRG requires a 30-tap FIR filter. Thirty numbers need to be multiplied and added up per clock cycle. It may seem that 15 multiplies are required per clock cycle, but this is not the case. Since the weights of the spreading code can only be a 1 or -1, multiplication can be avoided entirely by simply subtracting numbers with negative weights in the summation while adding positive numbers. Since the spreading code weights don't vary at run-time, the FUs can be statically programmed to to add or subtract. A pipelined adder tree can take care of the additions and subtractions.

Since the data comes in one number at a time, a pipeline of registers is required. The alternate

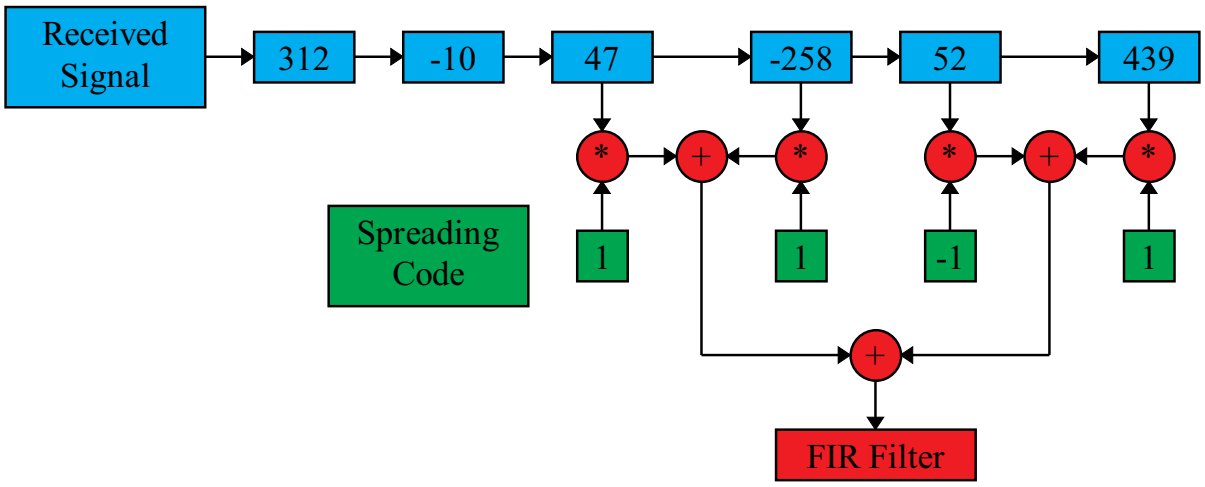

Figure 5.7: Matched 4-tap FIR filter.

way to do this is to buffer the incoming values and send them to the Colt in parallel, but not even the Stallion could handle 30 operand input bandwidth. Furthermore, this would require complex memory handling since numbers would have to be fetched from memory multiple times. A more efficient way of solving the problem is to use two columns of IFUs to pass the values down in a pipeline. The register can be tapped by the first level of adders and subtracters. This ensures a value will only have to be fetched from memory once. One of the problems with mapping this algorithm to the colt is the limited resources. More than one chip is needed. To keep the number of memory fetches required the same, the data values come out of the bottom of the pipeline unchanged and can be routed to another chip. This allows as many Colt chips as needed to be strung together to implement any size tap FIR filter. The only variable is the latency.

Figure 5.8 shows an implementation of an 8-tap FIR filter. This implementation assumes that two data values are available simultaneously. Four Colt chips can be strung together in series with one in parallel to complete a 30-tap FIR filter. Since the inputs to the FIR filter are 12 bits wide,

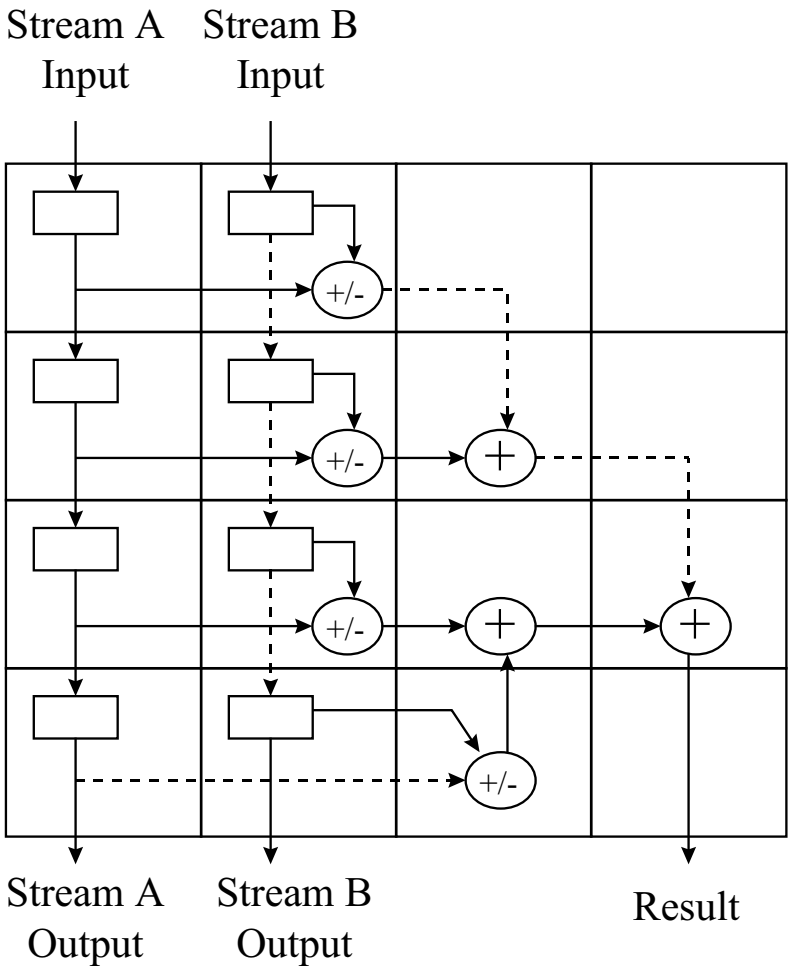

Figure 5.8: Matched 8-tap FIR filter on Colt

the same basic IFU configurations can also be mapped to two Stallion as long as four data items are available per clock cycle.

The matched filter operates at 3.84 MHz, or less than one tenth the clock speed of the Colt. Since the colt is run-time reconfigurable, with proper memory support it could be configured with four different set of weights and run the data set through it four times to produce four subsets. These can be buffered to four separate memories. The Colt could again be reconfigured to add up an element from each of these subsets at a time, producing an FIR filter result. Data would have to pass through the Colt five times. Even if programming overhead is the same as the data, a worst-case scenario gives one FIR result per 10 times through the Colt. At 50 MHz, this is an FIR filter speed of 5 MHz, well above the input rate of 3.84 MHz. As long as latency is not an issue, the FIR filter can also be mapped to one Colt chip with proper memory support.

Inherent to any pipelined architecture, there is a latency associated with these computations. The latency for the Colt is the worst case delay for data from input to output. This latency consists of delay through the Colt architecture including the data ports, crossbar, and IFU mesh. It can also consist of any buffering delays. For the FIR filter, this delay is calculated to be eight clock cycles per pass through a Colt chip. With a 50 MHz clock speed, this amounts to 0.8 μs for five passes through the chip. Since communication systems involve humans as the final link, up to several milliseconds can easily be tolerated. The latency for the Colt mapping of the FIR is on the order of one thousandths of the maximum allowable latency. If the single-chip approach is used, an additional latency for data in memory is added. For a single data item, this is 4 sets of 30 numbers (one set for each 4-tap FIR filter) or 150 clock cycles. The total latency is then 3.8 μs, well

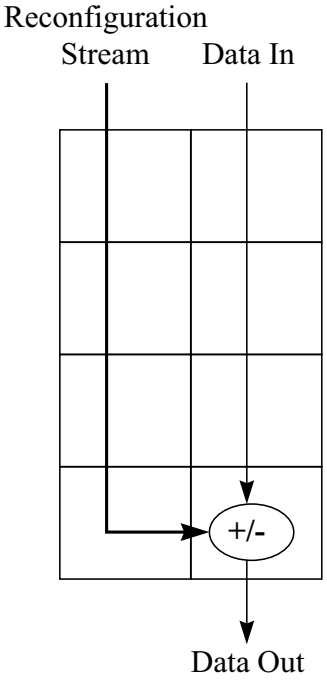

Figure 5.9: RTR Example on Colt

under a millisecond. It is obvious from these examples that latency attributed to the architecture is negligible.

5.3 Run-time Reconfiguration

Run-time reconfiguration (RTR) allows parts of the chip to be in configuration mode while other parts are in data mode. The parts of the chip in data mode can either be processing data or can be waiting for other resources to properly reconfigure. Figure 5.9 shows a simple example of how this can happen. A data stream is being processed on the for right column of the mesh. The final computation is an addition in the last IFU. It is possible for a configuration stream to come in on another input and reconfigure the last IFU to do a subtraction instead. This will disrupt the flow

of data, but only for as many cycles as it takes to program that IFU, or eight cycles. Contrasting this to the delay if the entire chip had to be reprogrammed (36 cycles) shows a delay cut by better than 75 percent.

Chapter 6

Conclusion

This thesis has presented the Colt computing platform, specifically the Colt custom computing integrated circuit. Verification on this chip was done using analog and digital simulations of the actual design. These procedures yielded changes in the design at the layout, circuit, and architectural levels. They also reaffirmed many aspects of the design that worked without additional design. Simulations were performed on everything from basic components to the full chip. The full chip simulations also attempted to provide a proof-of-concept by mapping important arithmetic and DSP algorithms to the Colt. These algorithms also give a proof of concept for other chips employing the Colt architecture such as the Stallion.

6.1 Future Work

Since the Colt platform is still relatively new, there is much room for additional work and improvements. As the system level boards and memories are designed and simulated, the performance of the Colt in a system can be better evaluated. At this level, a behavioral model of parts or all of the Colt chip should be developed to make simulations possible in a reasonable time frame.

There are a few features of the data path that should be examined in the next chip. Many of the iterative calculations like those done in Chapter 5 require a "shift and add" or "shift and subtract"

process. Furthermore, since the Colt lends itself much more easily to restoring algorithms, the shifted value is the desired one. Mappings such as these can be made more efficient if there were a mechanism to choose the shifter output as one of the conditional unit's inputs. Splitting the optional delay between the auxiliary and bus outputs may also increase efficiency, as would an option to allow the auxiliary outputs to use the local busses.

Circuit design issues will play a role in the design of the Stallion since the mesh will be larger and hence a longer skip bus. As the processes shrink wiring capacitance will be reduced, but transistor capacitance will take a larger role. Furthermore, to be able to run at a clock speeds of 100 MHz and more, a higher degree of pipelining will be required. This degree of pipelining was not incorporated into the Colt for area considerations. One area that adds significant delay is the ALU. This should be redesigned in the stallion to shave off another few nanoseconds. Primarily, this should involve increasing the size of transistors in the first few stages of carry logic and P and G input multiplexers. Another area that would benefit the clock speed would be the conditional and shift flags. These may need to be pipelined in future chips.

Although the effort undertaken here has attempted to uncover as many problems in the Colt, the tests are not exhaustive and a small number of errors may have slipped through the process. Additional verification of any components used here in future work would decrease the chances of error. Upon completion of the layout, actual capacitances and RC effects can be more accurately modeled through back annotation of these values from the layout. Although the potential problem areas have already been modeled and simulated, additional analog simulations can be done at the chip level at that time.

6.2 Results

The examples in Chapter 5 have shown the Colt chip to be not only effective at solving computational problems but also at a much higher efficiency when compared to conventional solutions [21, 2]. The FIR filter alone takes 6 SHARC chips to perform the same computations as one Colt [6]. With a small latency, operations such as division and square root can be done at the clock speed of the chip. That results in up to 100 MegaOps on the Stallion, a much higher rate than conventional processors or DSPs can currently perform.

REFERENCES

[1] Altera Corporation, *Altera Databook,* 1993.

[2] Analog Devices, Inc., *DSP/MSP Products Reference Manual,* 1995.

[3] J. R. Armstrong, F. G. Gray, *Structured Logic Design with VHDL,* Prentice Hall, 1993.

[4] A. S. Ashur, "Systolic digit-serial multiplier," *IEE Proceedings, Circuits, Devices and Systems,* vol. 143, pp. 14-20, Feb. 1996.

[5] D. E. Atkins, "Higher-radix division using estimates of the divisor and partial remainders," *IEEE Trans. on Computers,* vol. C-17, pp. 925-934, Oct. 1968.

[6] R. Bittner, "Design and VLSI Implementation of a High Performance Run-time Reconfigurable Data Flow DSP Computing System," PhD. dissertation in progress.

[7] A. D. Booth, "A signed binary multiplication technique," *Quarterly Journal of Mechanical and Applied Mathematics 4,* pp. 236-240, 1951.

[8] R. E. Bryant. "Graph based algorithms for Boolean function representation," *IEEE Transactions on Computers,* vol. C-35, pp. 677-690, Aug. 1986.

[9] D. A. Buell and K. L. Pocek, "Custom computing machines: an introduction," *Journal of Supercomputing,* vol. 9, pp. 219-229, 1995.

[10] Cadence Design Systems, Inc., *Cadence Reference Manuals,* 1993.

[11] L. Ciminiera and P. Montuschi, "Higher radix square rooting," *IEEE Trans. on Computers 39,* pp. 1220-1231, 1987.

[12] A. L. DeCegama, *The Technology of Parallel Processing: Parallel Processing Architectures and VLSI Hardware,* Vol. 1, Prentice-Hall, 1989.

[13] S. Divadas, K. Kertzer, S. Malik, A. Wong, "Event suppresssion: Improving the efficiency of timing simulation for syncronous digital circuits," *Advanced Research in VLSI and Parallel Systems,* pp. 165-171, 1992.

[14] K. M. Elleithy, M. A. Bayoumi, "A systolic architecture for modulo multiplication," *IEEE Transactions on Circuits and Systems, Part II, Analog and Digital Signal Processing,* vol. 42, pp. 725-729, Nov. 1995.

[15] J. Fandrianto, "Algorithm for high speed shared radix 4 division and square root," *Proc. of 8th Symp. on Computer Arithmetic,* pp. 73-79, May 1987.

[16] J. M. Feldman, C. T. Retter, *Computer Architecture: A Designer's Text Based on Generic RISC,* McGraw-Hill, 1994.

[17] A. L. Fisher, H. T. Kung, K. Sarocky, "Experience with the CMU programmable systolic chip," *Microarchitecture of VLSI Computers,* pp. 209-22, 1985.

[18] M. J. Flynn, "Very high speed computing systems," *Proc. IEEE,* vol. 54, pp. 1901-1909, Dec. 1966.

[19] R. L. Geiger, P. E. Allen, N. R. Strader, *VLSI Design Techniques for Analog and Digital Circuits,* McGraw-Hill, 1990.

[20] S. Harper, "Worm-hole Reconfigurable Custom Computing System," PhD. dissertation in progress.

[21] Harris Semiconductor, *Harris Semiconductor Digital Signal Processing Databook,* 1994.

[22] J. L. Hennessy, D. A. Patterson, *Computer Architecture: A Quantitative Approach, 2nd ed.* Morgan Kaufmann, 1996.

[23] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability,* McGraw-Hill, 1993.

[24] K. Hwang, *Computer Arithmetic: Principles, architectures, and design,* Wiley, 1978.

[25] J. Jackson, D. Casasent, "Optical systolic array processor using residue arithmetic," *Applied Optics,* vol. 22, pp. 2817-21, Sept. 1983.

[26] J. Jain, J. Bitner, J. A. Abraham, D. S. Fussell, "Functional partitioning for verification and related problems," *Advanced Research in VLSI and Parallel Systems,* pp. 210-226, 1992.

[27] J. Jain, J. Bitner, D. Fussell, J. Abraham. "Probalistic verification," *ICCAD,* 1991.

[28] I. Koren, *Computer Arithmetic Algorithms,* Prentice Hall, 1993.

[29] S. Majerski, "Square-rooting algorithms for high-speed digital circuits," *IEEE Trans. on Computers,* Vol. C-34, pp. 724-733, Aug. 1985.

[30] M. M. Mano. *Computer System Architecture,* 2nd. Ed., Prentice Hall, 1982.

[31] M. Musgrove, "VLSI Design of a Run-time Reconfigurable Custom computing Integrated Circuit for DSP Applications," Master's thesis in progress.

[32] R. B. O'Connor, "Dataflow Analysis and Optimization of High Level Language Code for Hardware-Software Co-design," Masters thesis, Virginia Polytechnic Institute and State University, 1996.

[33] S. Palnitkar, *Verilog HDL: A Guide to Digital Design and Synthesis,* Prentice Hall, 1996.

[34] F. P. Preparata, "Pratical cellular dividers," *IEEE Trans. on Computers,* vol. 39, pp. 605-614, May 1990.

[35] P. Ross, "Moore's Second Law," *Forbes Magazine*, vol. 28, no. 2, pp. 98-99, Mar. 1996.

[36] J. E. Robertson, "A new class of digital division methods," *IRE Trans.,* vol. C-34, pp. 218-222, Sept. 1958.

[37] N. Shirazi, A. Walters, and P. Athanas, "Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Machines," *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines,* pp. 155-162, April 1995.

[38] W. Tanner, *The MOSIS User Manual 4.0,* University of Southern California, 1995.

[39] Y. Tsividis, *Mixed Analog-Digital VLSI Devices and Technology: an Introduction,* McGraw-Hill, 1996.

[40] P. W. Tuinenga, *SPICE: A Guide to Circuit Simulation and Analysis Using PSpice,* Prentice Hall, 1988.

[41] J. F. Wakerly, *Digital Design Principles and Practices,* Prentice Hall, 1990.

[42] N. H. E. Weste, K. Eshraghian, *Principles of CMOS VLSI Design: a Systems Perspective,* 2nd ed., Addison-Wesley, 1993.

[43] Xilinx, Inc., *XC6200 FPGA Family: Advanced Product Description,* 1995.

[44] Xilinx, Inc., *The Field Programmable Logic Data Book,* 1994.

Appendix A

Simulation Procedures

Throughout the development of the Colt chip, various tools were used. Some were successful and some were not. The first tool used was *Mentor Graphics*. This proved to be a powerful, yet difficult tool to use. The interface and data structures used for information storage were hard to work with. For these reasons, the development efforts were moved to a tool by *Viewlogic* called Powerview. This tool proved to have a convenient user interface and be productive to use; however, it provided no layout support. Furthermore, the analog simulator used was SPICE. As the design grew it became too large for this program to handle.

The need for a comprehensive tool with a good user interface that provided layout support was found in the *Cadence Design System* (CDS) tools. These tools also provided a commercial-grade analog simulator that could handle large circuits. Furthermore, the digital simulator had many advantages outlined in chapter 4. This appendix attempts to outline the procedures for running digital and analog simulations under Cadence.

A.1 Cadence Procedures

There are several start-up files that Cadence uses to initialize its environments. These include ".cdsinit," ".simrc," ".stlrc," and ".cdsplotinit". The first file should be in the user home directory.

This file should be edited to include paths to the libraries the user plans to access. The rest of the files should be found in the home directory or wherever the user plans to start Cadence from. The second file is used by the simulators. It should be edited for whatever simulation environment the user plans to run. The last two files usually do not need editing but should be available.

To start the CDS tools, one should type "icfb" at the command line. This will give you the entire tools "front-to-back". Although there are other ways to run cadence to minimize memory, care should be taken to insure that the desired tools are unaffected. There have been some simulation problems running cadence under "icds". After a few seconds, the main window, called the CIW, will appear.

There are several good references to help one get started with the CDS tools. One of these is the Iowa State University online help. The other is the CDS reference manuals. The former can be accessed from the CIW (**ISU** $\rightarrow$ **ISU Online Help**) while the latter can be accessed by typing "openbook" at the command line. Under the main openbook menu, "IC Tools" will probably provide most of the needed support.

To open a design in cadence, open the **Library Browser** under **Design Manager** in the CIW. The Library Browser allows a user to access libraries and designs. Simply click on a library, directory, or design to open it. A single design in cadence can have multiple "views". These views are representations of the same design. For example, a circuit called "test" can have a schematic, a symbol, a behavioral model, and a physical layout. These are all separate views under the design "test". For simulation purposes, the schematic or behavioral views will be of interest, although a simulation can be done from a physical view or from a symbol.

A.1.1 Digital Simulations

Digital simulation in cadence is performed under the Verilog-XL environment. To run a digital simulation on a design, the design must first be opened. Click on the middle mouse button while holding the pointer over the schematic view of the design and a menu will appear. Select **Read**. When the design is open, select **Tools** → **Simulation** → **Verilog-XL** from the toolbar menu. A window will appear asking for a simulation directory. It is a good idea to keep each simulation run in a separate directory. Furthermore, it is a good idea to keep all digital simulations in one directory. The directory specified will be added from the directory where Cadence was started. So typing "vl/test.run1" add the directory "test.run1" to the tree "cadence/vl" if Cadence was started from the directory "cadence".

The Verilog environment window will come up next. There are several menu items of interest. First of all, it is recommended that STL is used as the input language. This will allow the same input files to be used for both analog and digital simulations. To create or edit an STL file, select **Stimulus** → **STL** → **Edit Stimulus**. The STL file must be compiled, so always make sure the compile option is on when prompted. If internal signals are of interest, it is also recommended that all signals be recorded for viewing. This is done by selecting **Setup** → **Record Signals** and selecting **All Signals** from the selection box.

To run a Verilog simulation, click on the **Start interactive** symbol in the upper left corner. After netlisting and compiling, the simulator will stop and other symbols will become bold. Select the "continue" symbol. After a few seconds, the simulation will be finished. To view results, select the **View waveforms** symbol at the bottom right. Select signals in the schematic to view and

click on the **Add waveform** button in the waveform viewer.

For Verilog netlists to work on split busses, the following line must be in the .simrc file: simVerilogDropPortRange = nil.

A.1.2 Analog Simulations

To run an analog simulation, a design must be opened as described above. Select **Tools-¿Simulation → Other** from the toolbar menu. A **Simulation** field will appear on the toolbar menu. Select **Simulation → Initialize**. This will prompt for a simulation directory just as in the digital simulations. The simulator name should be selected as **u_spectre** unless another simulator is desired. A directory for analog simulations is recommended. Input files can be created or edited from the **Simulation → Stimulus → Edit STL Stimulus** menu. The first thing that needs to be done to a design is proper netlisting. Select **Simulation → Netlist/Simulate**. A window appears that has selection boxes for netlisting and simulation. Make sure only the netlist box is selected and click on "OK". This netlists the design. At the command window under the simulation directory, run the program "replacelw". This is a program that updates the length and width from lambda units to physical measurements. If the schematic already has the width and length in physical units, omit this step. Select **Simulation → Netlist/Simulate** again and make sure only the "Simulate" box is selected and click on "OK". This will simulate the design. To view results, select **Simulate → Show Waveforms**. Under the waveform window, select **Wave → Add by name**. List the actual net name of any signals to view and click on "OK".

There is one known problem with the analog simulations: Spectre will not work in a Cadence

session when Verilog has been run. If a "u_spectre.ile not found" error occurs during netlisting, simply shut down the tools and restart; in fact any time strange things happen after repeated simulations, restart the tools and try a new run directory.

A.2 GUI Procedures

Using the Colt GUI can aid significantly in programming the chip. To use the GUI, make sure the paths are correctly set and type "coltgui". The main menu appears. Under the **Configure** menu, the size of the mesh can be specified. Click on **Mesh Configuration** to bring up the *Cluster Configuration Constructor*. This tool can be used to configure any of the IFUs, data ports, or crossbar. Click on a resource to configure it. A series of menus will appear with the respective choices for configuration. Configurations can be stored or read from disk. It is a good idea to save a configuration before generating programming streams.

To generate programming streams, click on **Program-Specify Program Sequence**. A window will appear with a list of all the configured resources. Click on the resources in the order of program stream flow. Save the stream as a .dfc file. In a shell window, type "dfc [filename.dfc] [filename.pwl]" to run the data flow compiler. This will generate a PWL file. The PWL file is a list of programming words in order. These values can be inserted into the simulator input files for simulation. PWL will have an option in the future to convert a .pwl file to an STL input file for the chip.

One possible problem with the GUI may arise when generating multiple programming streams in one session. New streams may have old stream information in them. To avoid this, exit the GUI

each time a new stream is programmed.

VITA

Mark F. Cherbaka was born on June 4, 1971 in Beirut, Lebanon, and immigrated to the United States in April, 1976. Mark became a US citizen in 1990. He received the Bachelor of Science Degree in Computer Engineering with a Minor in Computer Science from Virginia Tech in December, 1993. In August, 1994 he enrolled in the graduate program at Virginia Tech to pursue a Master of Science in Electrical Engineering. His focus is on VLSI design and microprocessors. Mark will be joining IBM at Burlington, VT in August, 1996 where he will be doing circuit design for Power-PC microprocessors.