Raw Fabric Hardware Implementation and Characterization

by

Albert Sun

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

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

[June 2006]
May 2006

Author .... [illegible]
Department of Electrical Engineering and Computer Science
May 9, 2006

Certified by .... [illegible]
Anant Agarwal
Professor
Thesis Supervisor

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

Raw Fabric Hardware Implementation and Characterization

by

Albert Sun

Submitted to the Department of Electrical Engineering and Computer Science
on May 9, 2006, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The Raw architecture is scalable, improving performance not by pushing the limits of clock frequency, but by spreading computation across numerous simple, replicated tiles. The first Raw processors fabricated have 16 RISC processor tiles that share the workload. The Raw Fabric system extends Raw's scalability by weaving together multiple 16-tile Raw processors.

The Raw Fabric is a modular and scalable system comprised of two board types: one to house 4 Raw processors (Processor board) and one to handle communications (I/O board). The design is modular because it breaks down the system into smaller parts, and it is scalable because these modules may be combined to create large Fabrics. The ultimate goal is to produce a Raw Fabric with 16 Processor boards (equivalently, 64 Raw processors or 1024 tiles), though the current largest Fabric system includes one Processor board and 3 I/O boards.

This thesis walks through the important design and implementation challenges and documents how they were solved. The most basic challenge faced was to design a system flexible enough to accommodate a variety of Fabric sizes. Next, the distribution of vital signals such as power and clock provides a problem unique to the Fabric system because of the possible size of the final product. Finally, the astounding number of signal wires running between boards presents a unique challenge in finding parts and designing the mechanical aspects. The intent of this thesis is to provide the reader with an idea of the considerations necessary for designing and implementing a system of this magnitude and level of flexibility.

Thesis Supervisor: Anant Agarwal
Title: Professor

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Interest in parallel architectures has increased steadily as the limits of physics become more and more constraining at higher clock frequencies. As superscalar microprocessors become more and more complicated, the propagation delay of electrons along wires is becoming a larger and larger constraint. Parallel architectures aim to improve performance not by pushing electrons closer and closer to light-speed, but rather by spreading the computation across multiple computational units [9].

Raw is one such parallel architecture that was designed and implemented at MIT. Raw focuses on maintaining the simplicity of the identical individual tiles. This simplicity allows for extremely easy scalability: to increase performance, simply replicate more tiles. Instead of increasing the complexity of the microprocessor, the software compilers carry the burden of optimizing the application to the increased number of computation tiles. In this model, the hardware is fully exposed to the compiler so that the compiler may make best use of the resources [1].

1.1 The Raw Processor

The Raw processor was designed and implemented at MIT to demonstrate the power of scalability afforded by the design of the Raw architecture. The processor has 16 tiles that simultaneously execute Reduced Instruction Set Computer (RISC) instructions. Each tile has its own register set, instruction cache, and data cache. The processors are modeled after the MIPS 4000 processor with a 5-stage pipeline. These processors are simple enough that 16 of them fit on a single reasonably-sized die.

Greater detail on the design and implementation of the Raw processor is covered in Chapter 2: Background.

1.2 The Raw Fabric

This thesis focuses on the design, implementation, and characterization of the hardware in the Raw Fabric system. The Raw Fabric system wires together multiple Raw processors to produce the equivalent of a larger Raw processor.

1.2.1 Motivation

The motivation behind building the Raw Fabric is two-fold. First, the Fabric provides a platform to demonstrate the ease and power of scalability in the Raw architecture. Second, the Fabric provides a test platform on which further research on various algorithms may be done.

Scalability Goal

The scalability of the Raw architecture involves many factors beyond the simple argument that computation may be distributed. For example, as the system grows larger, the memory system may suffer, or the latency of the point-to-point messaging system (see Section 2.1.4) may become prohibitive. On the hardware front, reset or clock skew becomes a larger consideration as more tiles are added.

The Raw Fabric provides a hardware implementation of the architecture so that the strengths and weaknesses of the Raw architecture's scalability may be assessed. Without a hardware implementation, it is easy to fall into the pitfall of making unrealistic assumptions.

Test Platform Goal

The Raw Fabric will allow researchers to profile the performance of algorithms as the number of tiles grows larger. Previously, researchers utilizing the Raw processors could show the performance behavior as the number of tiles increases to as many as 16 tiles. The Raw Fabric will open new doors by increasing this number to as many as 1024 tiles.

Building this hardware test platform also has advantages over simply simulating a large processor in software. First, the hardware runs much faster than simulation. Second, hardware allows for the measurement of variables difficult to measure in simulation, such as power consumption. Lastly, results from experiments run on hardware are more credible than in simulation, because the simulator may have some assumptions that are unrealistic in the physical world.

1.2.2 Design and Implementation Challenges

The design and implementation of the Raw Fabric Hardware system presents a set of challenges unique to this system.

Modular Design

Chapter 3 discusses the first challenge, how the Raw processors and various other components would be laid out on one or more boards. As with any difficult decision, each choice of board layout has its own advantages and disadvantages. A single large board system has the costly disadvantage of being difficult to repair. However, using more than one board leads to the need to connect the boards together, which leads to another gamut of challenges. In the end, the significant repair cost overwhelms the nuisance introduced by connectors.

Even after the choice of a multiple board design over a single large board, the actual design of the modular system is still a momentous task. In Section 3.4, the choice to use two different types of boards – one to house the Raw processors (the Processor Board) and one to provide I/O (the I/O board) – is justified. Section 3.5

provides a glimpse of the considerations required to make the system general enough such that the I/O board may be attached onto any side of a Processor board.

Power Distribution

There are two major challenges related to power distribution in the Raw Fabric system.

One challenge presented by the use of a multiple board system is that each board needs to be powered somehow. However, using a separate power supply for each board is infeasible because an inordinate number of supplies would be required.

In addition, power distribution in this system is unique because of the momentous total power requirement. A full 16 Processor Board Fabric contains 64 Raw processors, which equates to roughly 1280 Watts of power.

Chapter 4 explores these challenges in greater detail and presents our solutions.

Reset Distribution

Just like power distribution, the need for reset distribution arises from the choice to use multiple boards. Reset distribution is also discussed in greater detail in Chapter 4.

Clock Distribution and Synchronization

Much like power distribution and reset distribution, the need for clock distribution is attributed to the choice to use multiple boards.

On the other hand, the need for clock synchronization would probably be pressing whether a single board or multiple board design was chosen, since processors on opposite ends of the Fabric are likely to be physically distant from one another. This distance leads to the presence of clock skew, since the clock signal takes time to traverse the wire to the farther processor. Therefore, a method to synchronize the clocks being fed to all Raw processors is required.

These issues and their solutions are discussed in Chapter 4.

Connectors

The use of multiple boards leads to the very real mechanical considerations of connectors. Our system design presents the unique challenge of creating a very large number of connections in many different places. Section 4.2.1 describes how the problem of making the multitude of connections is solved, and it also provides a tour through the new problems that arose from the solution.

1.2.3 Results

The Raw Fabric system was successfully implemented and characterized. The implementation of the Fabric is described in Chapter 5. Various key electrical characteristics are highlighted in Chapter 7.

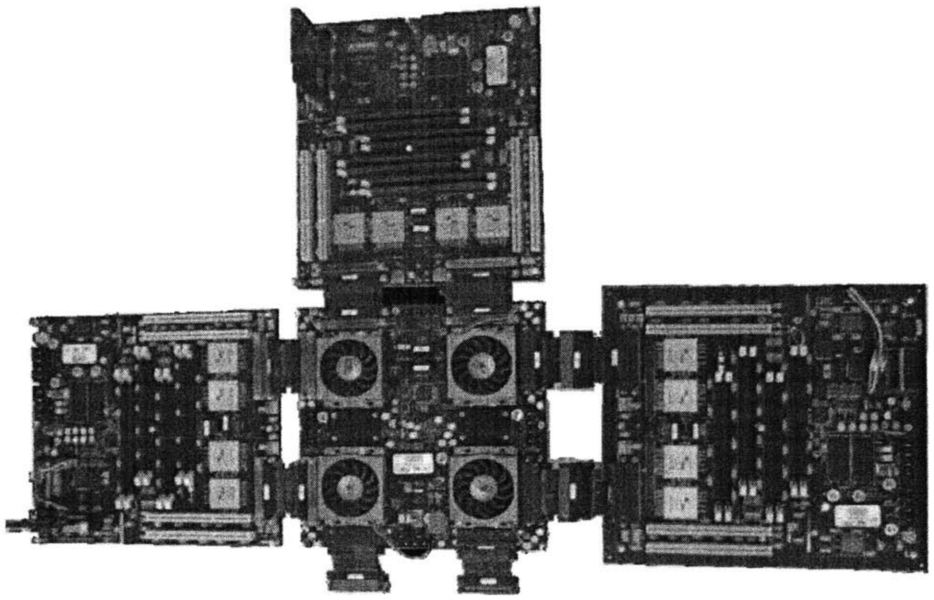
The Raw Fabric system succeeded in providing a test platform for applications to be run. The implementation also provided many lessons. Chapter 6 explains these lessons so that one who intends to build a similar hardware system may learn from our experiences.

Figure 1-1 shows a small Raw Fabric that has been implemented at MIT. The center board is a Processor board, which houses 4 Raw processors (covered by the 4 prominent orange heat sinks in the picture). The 3 peripheral boards are I/O boards responsible with communicating with the outside world.

1.3 Thesis Overview

The remaining sections of this thesis are broken down into eight sections. First, Chapter 2 paints the background picture, providing a rough sketch of the Raw processor and Raw Handheld system history. Next, Chapter 3 outlines the high-level design goals and overarching design structure of the Raw Fabric. With the design goals in mind, Chapter 4 describes various trade-offs made in the design of the Fabric system. Chapter 5 provides the nitty-gritty implementation details in the Raw Fabric hardware. Chapter 6 provides insight into lessons learned from the experience of building

Figure 1-1: Photograph of Implemented Raw Fabric



a hardware system of this size. Chapter 7 summarizes a few key electrical characteristics of the hardware system. Finally, Chapter 8 provides a summary of work done and a view of future work to ensue.

1.4 Collaboration

The Raw Fabric Hardware project is a collaborative work with Jonathan Eastep, Jason Miller, and Haydn Nelson. Jonathan Eastep, Haydn Nelson, and I handled much of the hardware implementation, debugging, and design revisions. Jason Miller was largely responsible for the initial design of the system, and he provided valuable advice during the later implementation stages.

Chapter 2

Background

This chapter provides a whirlwind tour of the foundation upon which the Raw Fabric system rests. The intent of this background material is to familiarize the reader with topics relevant to the Raw Fabric system. This chapter by no means provides a comprehensive description of the Raw Processor or Raw Handheld System. For more details on the Raw architecture, please consult *The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs* [10].

2.1 The Raw Processor

The Raw Processor is a tiled processor developed at MIT in the Computer Architecture Group (CAG).

2.1.1 Tiled-Processor Architectures

As power consumption and wire delay become more and more limiting on today's superscalar microprocessors, microprocessor architects are faced with the challenge of finding new approaches to improve performance [9].

The idea behind any tile architecture is replication. Instead of a single extremely fast processor core, multiple cores distribute the workload.

The possibility of overcoming the power consumption and wire delay hurdles has spurred on many tiled architecture research undertakings, including VIRAM [3] at UC-Berkeley, Smart Memories [7] at Stanford, TRIPS [8] at UT-Austin, Raw [12] and SCALE [4] at MIT. The focus of these research initiatves has been to offer better versatility - ability to run a broader range of applications effectively - rather than strictly better performance on desktop workloads [9].

2.1.2 Compilers

The Raw system operates under the assumption that there exists a compiler that can distribute the computation in a manner that maximizes efficiency. Many research topics have been explored, including work on scheduling [5, 6] and work on memory management [2]. However, the details of these compilers is beyond the scope of this thesis.

2.1.3 Inter-tile communication

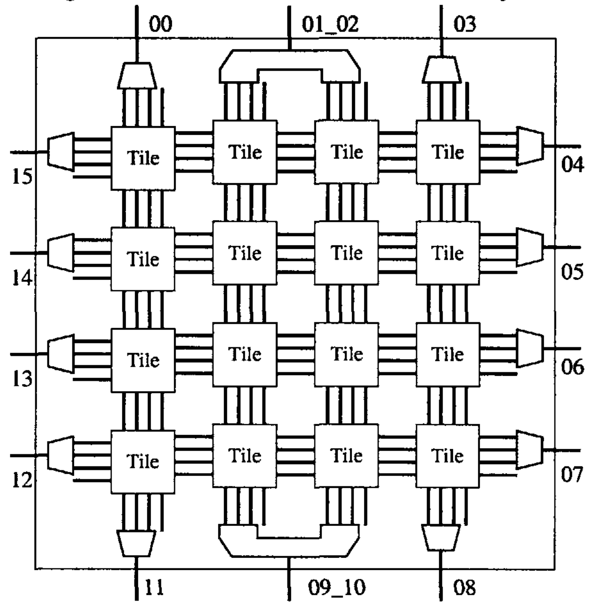
An important feature of any tile architecture is the availability of communication between tiles. The tiles require fast communication between tiles in order to coordinate activities and to share data.

The Raw processor architecture includes four networks: two static networks and two dynamic networks. The two static networks provide communication between two static neighboring tiles. On the other hand, the dynamic networks provide communication between any two tiles (which may be decided at runtime).

All four networks have the form of a grid, where each tile is connected to its north, east, south, and west neighbors (see Figure 2-1). Tiles on the edge of the board are connected to ports off the chip, where their neighbors would have been.

Though the previous description is sufficient to understand the design of the Fabric, there are many details needed to actually use the Raw architecture effectively. These details may be found in in the Raw Specification [11].

Figure 2-1: Raw Tile and Port Layout



2.1.4 Off-Chip Communication

This section describes the aspects considered when choosing how the Raw processor would communicate to the outside world through pins.

Point to Point Interconnection Network Channels

The Raw processor uses point to point interconnection channels to communicate between tiles and with the outside world. Each tile communicates with up to 4 neighbors on 4 different networks: 2 static networks, a memory dynamic network, and a general dynamic network.

Furthermore, the data between each pair of points is transmitted on a parallel bus. This design decision allows the entire chip to use a single common clock. The single clock and parallel buses result in a system that is simpler to design, since extra synchronization and conversion (to/from serial) logic is not required.

Package Limitations

The Raw processor was fabricated using the IBM SA-27E ASIC process, and it was packaged in a 1657-pin CCGA package. This package has 1080 HSTL_I signal pins.

Pin Requirements

Unfortunately, the parallel nature of the Raw processor's networks means that many pins are needed. Each 32-bit port requires the 32 bits for data, and also 2 bits to encode the network the data resides on, and 3 bits for an acknowledge signal, for each direction. Given there are 16 ports and 2 direction, approximately (40*16*2) = 1280 pins are required for I/O communication through the package's pins.

Port Multiplexing

Since the pins would not all fit in the package, it was necessary to find a solution to save a few pins. In the end, the decision was made to multiplex a few of the ports together. Therefore, two pairs of ports on the north and two on the south (in both directions) share a 32-bit data bus, as shown in Figure 2-1. This choice degrades the performance through these ports when both of the ports are being used heavily, but it maintains a simple, logical design.

Only Three Networks Leave the Chip

In addition, though the Raw processors have 4 networks, the memory dynamic, general dynamic, static 1, and static 2, only the first 3 connect off the chip. This is another pin-saving choice: with 3 networks, 2 wires can encode the ready signal (00 represents no activity). In addition, we save a pin in the acknowledge signals (acknowledge signals are not multiplexed, for performance).

2.2 The Raw Handheld System

The Raw Handheld system was the first hardware system designed and built for use with the Raw processor. The system was designed to show off the capabilities of the Raw processor. The system's name comes from its original purpose to demonstrate the applications that *could be possible* using a Raw processor on a hand-held system. The system itself is not small enough to be held in a hand.

2.2.1 Functions

The handheld system looks and acts much like a modern PC motherboard. The system's main function is to allow the Raw processor to communicate with the outside world in a variety of ways. When the handheld system was designed, one of the most important goals was to allow for maximal expansion and experimentation. The handheld system's host of functions include, but are not limited to:

1. Memory
2. USB
3. PCI
4. Serial RS-232
5. Audio
6. A/D and D/A converters
7. Keyboard and Mouse
8. Connector for Text-only LCD

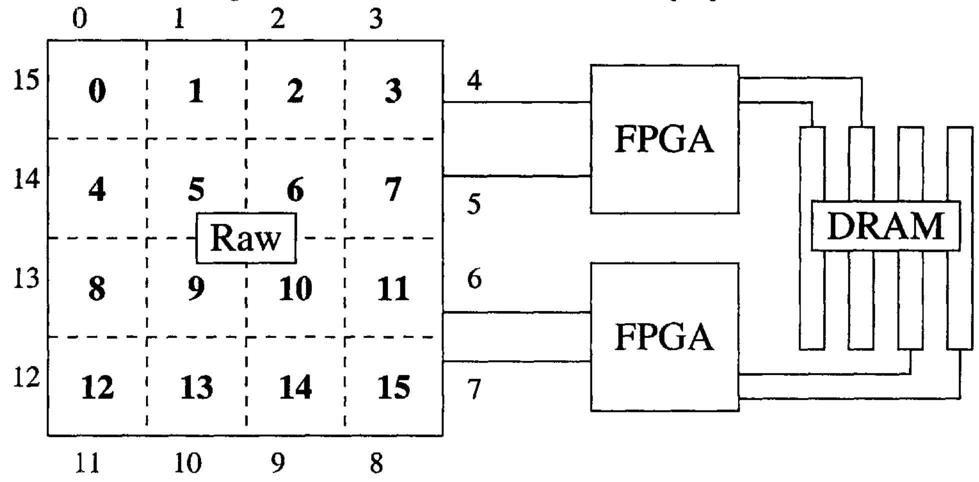
The functions that bear the most relevance to the Raw Fabric system are the Memory and USB subsystems.

Raw Handheld Memory System

The Raw Handheld board has four industry standard 168-pin SDRAM DIMM slots for system memory. By default, these slots are populated by 512MB sticks, for a total of 2GB system memory.

Figure 2-2 shows the overall structure of the Raw Handheld memory system. The DRAM slots are connected to two FPGAs, which in turn are connected to ports 4 through 7 on the Raw processor.

Figure 2-2: Raw Handheld Memory system



The FPGAs must be loaded with firmware that handles memory requests from Raw and translates them into the protocol the DRAM understands. In other words, the firmware must implement a memory controller.

Figure 2-3: Raw Handheld Memory Controllers

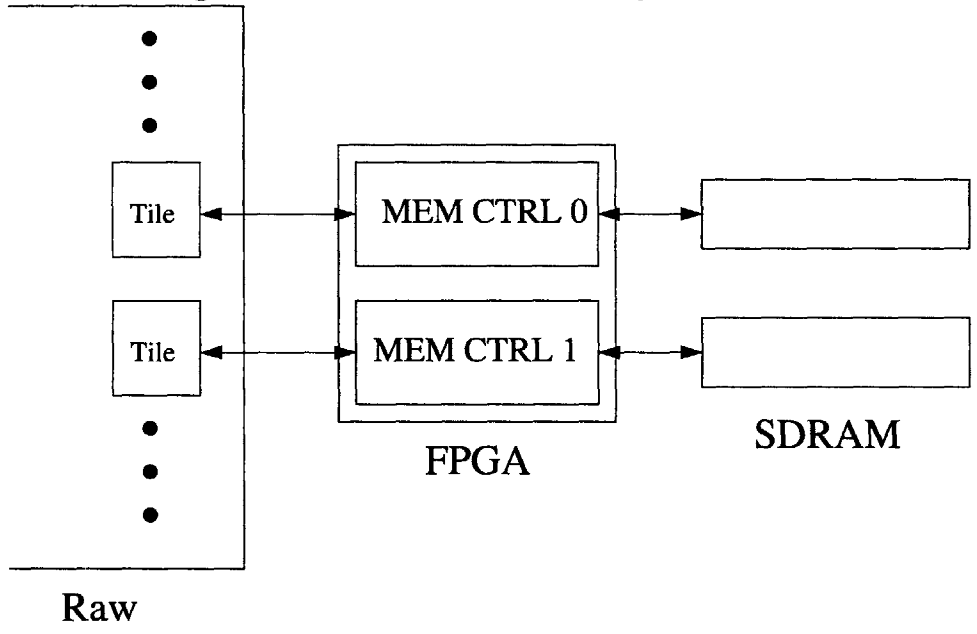


Figure 2-3 shows that there is a memory controller for each port on Raw. Each memory controller interfaces with one stick of SDRAM. Thus, there are a total of four memory controllers in the Raw Handheld system.

Figure 2-4 provides a high-level block diagram of a single memory controller. It shows the path data traverses as memory store and load requests are made.

The first 3 blocks provide buffering and clock domain conversion. First, the memory request arrives at the memory controller from Raw on the memory dynamic network (mdn). The speed gasket is responsible for sorting data according to network and providing some buffer space for each netowrk. The Network Input Block (NIB) element of the memory controller reads this data from the speed gasket first-in-first-out buffer (FIFO). Next, the data is stored in an asynchronous FIFO, in order to connect from Raw's clock domain to the memory's clock domain. This asynchronous FIFO allows the memory and Raw to run at different clock frequencies.

The next stop on the data path is the codec block, which handles the decoding of and encoding to Raw's communication protocol. The header decode unit communicates with the SDRAMC unit, which is generates and receives control signals for the SDRAM. The rest of the logic in the codec unit handles the packaging and depackaging of data. This processing is necessary because data to and from Raw is communicated in 8-word packets.

If the memory request was a load, then an 8 data word mdn packet will be sent back to Raw. When data leaves the codec module, it arrives in an asynchronous FIFO, which switches clock domain back to Raw's. Next, the data is read into the Network Output Block (NOB). From the NOB, it passes through the speed gasket and back into Raw.

Raw Handheld USB System

USB is the method by which the Raw processor interacts with a host computer. In order to communicate to a host computer, the Raw processor simply sends a static network, mdn, or gdn message through an off-chip port that is connected to an FPGA with USB controller firmware. On the Handheld system, sending messages from Raw to ports 12 through 15 will end up at the host computer.

Figure 2-5 shows the overall structure of the Raw USB system. A USB cable connects the host computer to the USB Plugin card on the Raw Handheld board. This plugin board houses the SX2 USB controller, which decodes the USB packets.

Figure 2-4: Raw Handheld Memory Controller Firmware Block Diagram

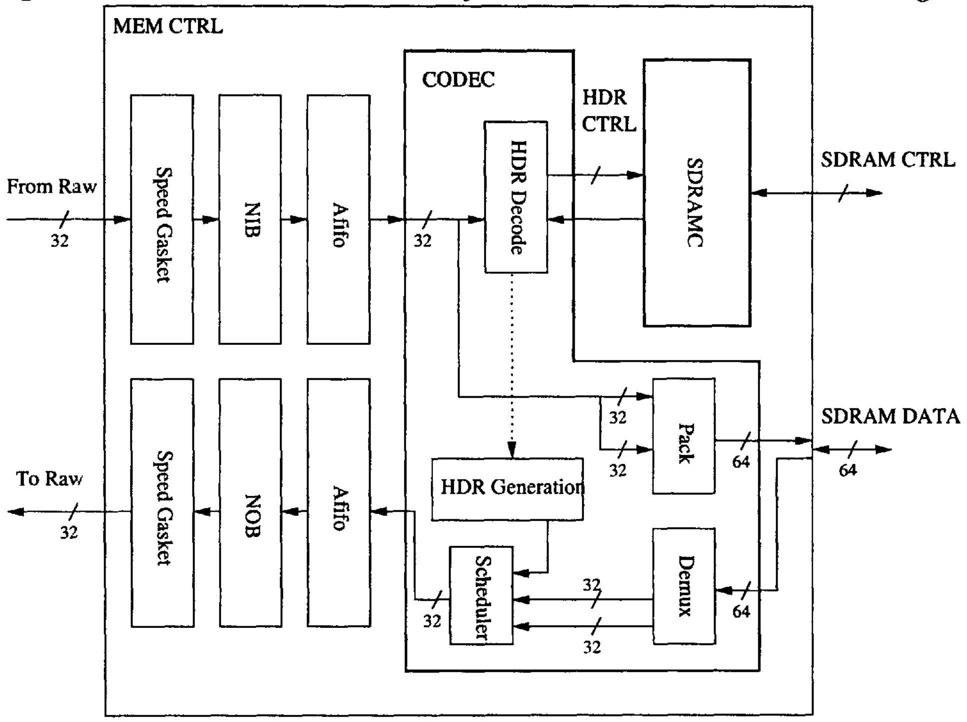
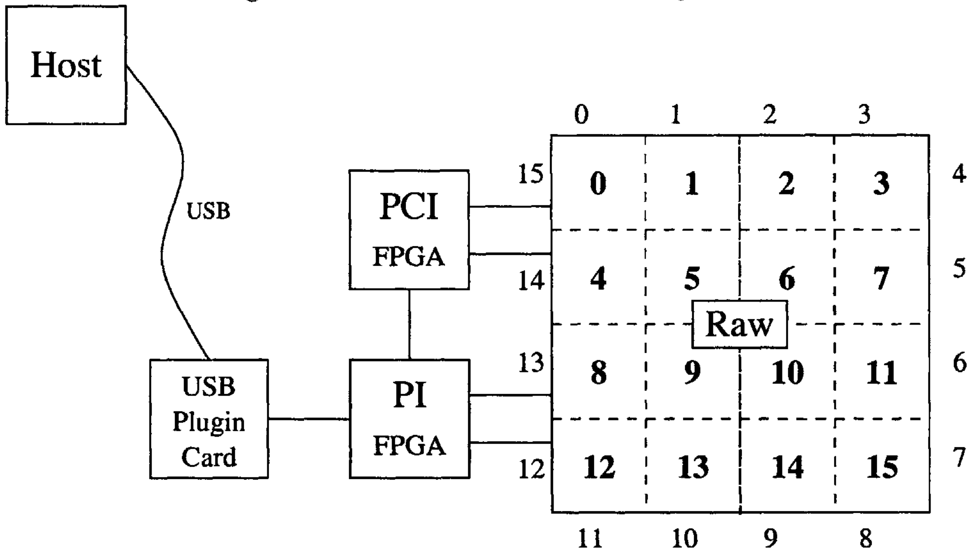


Figure 2-5: Raw Handheld USB system



The USB plugin card is connected to the PI FPGA. There is a routing network known as the "test harness" that sorts packets between the PI and PCI FPGAs (the name was apparently a temporary name that was never changed). The PI and PCI FPGAs connect to Raw.

Figure 2-6: Raw Handheld USB Firmware PI FPGA Firmware Block Diagram

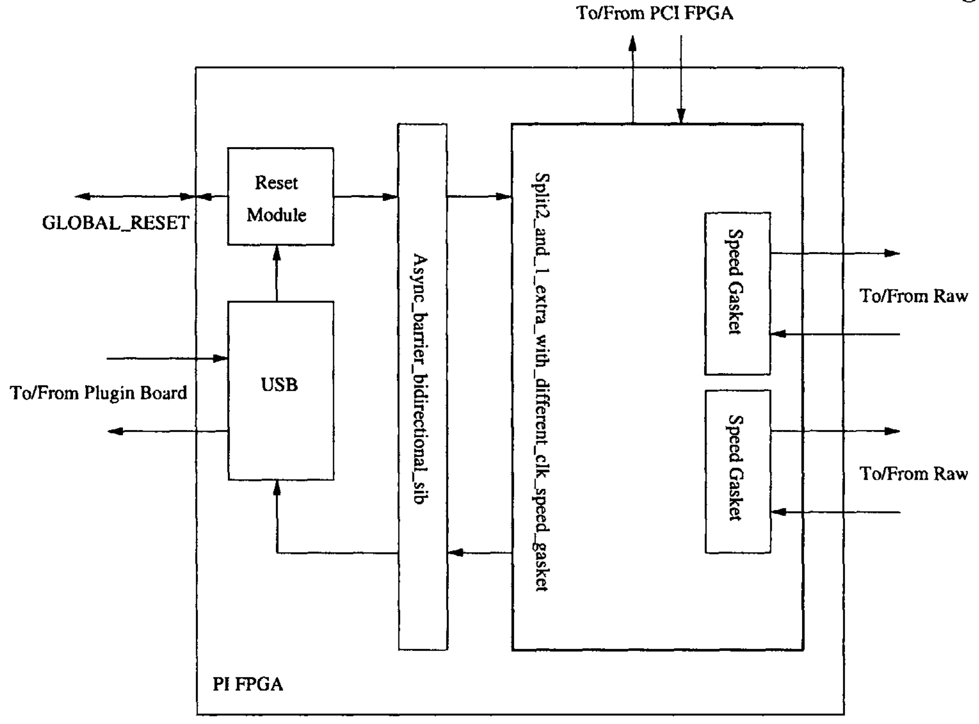


Figure 2-6 provides more detail on the internal organization of the USB firmware on the PI FPGA. The data that arrives from the host computer through the SX2 plugin board is interpreted and translated into header/data pairs by the USB module. The data then passes through a reset module, which picks out special packets (identifiable through a special header) from the data stream that instructs the PI FPGA to initiate a reset on the Handheld board. This allows the host computer to reset the board remotely before streaming in boot code.

Next, an asynchronous barrier connects from USB's clock domain to the test harness's clock domain. The test harness routes packets so that they arrive at the correct port on Raw. The speed gasket interfaces with Raw.

2.2.2 BTL simulation environment

The BTL simulation environment allows programmers to simulate their code before running it on hardware. A remarkable feature of the BTL simulation environment is that it includes an accurate Verilog model of the Raw processor, so the simulation results execution results match execution results on the hardware.

The BTL simulation environment supports simulation of larger fabrics like the Raw Fabric system. However, the simulation environment excludes many of the hardware restrictions present in the Raw Fabric hardware.

2.2.3 Firmware Development

Firmware development in the handheld was performed using the Veribuild suite organized by David Wentzlaff. Veribuild automates the firmware compilation process by using GNU Makefiles to describe projects.

This system also allows for modular design of components. The Xilinx XC2V3000 FPGAs chosen were intentionally large, to be able to perform many different tasks. Modular development allowed firmware development to proceed in an organized, parallel manner. For example, the speed gasket and memory controller submodules could be developed in parallel. Furthermore, the speed gasket, which contains many source files, could be reused easily in many independent projects whenever interfacing with Raw. One major advantage of modular design is that when a change to the speed gasket is made, it is automatically made in all of the independent projects that reference the speed gasket module.

Chapter 3

Design

This chapter describes the system design of the Raw Fabric. It addresses high-level issues considered when first laying out the structure of the system, whereas the next chapter includes much more specific design considerations that result from design choices in this chapter.

3.1 Fabric Size

The goal of the Raw Fabric system is to connect multiple Raw processors in such a way that they emulate the behavior of a single Raw processor with more tiles. By constructing the equivalent of a 64-tile Raw processor or even a 1024-tile Raw processor, we are allowed a sneak preview at a future implementation of the Raw processor. The goal of the Fabric system is to be modular enough to accommodate Fabric sizes from 64 tiles to 1024 tiles, or equivalently, 4 Raw processors to 64 Raw processors.

3.2 Multiple Boards

In order to use a Raw processor, it is necessary to mount it on a printed circuit board (PCB). There are compelling reasons to mount the Raw processors on multiple boards rather than mounting all of them on a single board.

First, the use of multiple boards rather than a single board allows each of the boards to be smaller in size. The ultimate goal of 1024 tiles requires the use of 64 Raw processors (each Raw processor contains 16 tiles). Mounting 64 processors on one board would require a prohibitively large board. In addition, since it is expensive to remove and remount Raw chips, if a smaller board is broken, we only lose the number of Raw chips on the board. If our design is sufficiently modular, then the broken board can just be replaced.

The use of multiple boards allows for modularity of design. By breaking down the design into smaller parts, it is possible to design and debug each part at a time. This modularity results in shorter debugging times. If there is a problem with one board, it may be easily replaced by a working board.

Finally, using multiple boards provides flexibility in Fabric size not possible with a single board. By using multiple boards, it is possible to create Raw Fabrics with different sizes and shapes by using fewer or more boards. Thus, using multiple boards allows aspects of the design to be incrementally validated on a smaller Fabric before attempting to build a larger Fabric.

In the end, we decided that four Raw processors on each board would provide the correct balance between number of connections needed and physical board size.

3.3 I/O

The Raw Fabric hardware system has many functions beyond simply providing a place to mount the Raw processors. Most notably, the hardware system must provide a way for the Raw processors to interact with the outside world. In other words, the hardware system must provide Input/Output (I/O) channels for the Raw processors.

The hardware system was designed to accommodate many different I/O types in order to provide versatile functionality. First, the hardware system provides USB support to communicate with a host computer. Next, the hardware system provides a memory system, so that the Raw processors may store data in memory without paying the latency overhead of going to the host. In addition, the hardware system supports PCI slots so that expansion boards may be accessed. Finally, the hardware system provides an additional expansion connector that provides a wide bus by which the hardware system may communicate with custom hardware.

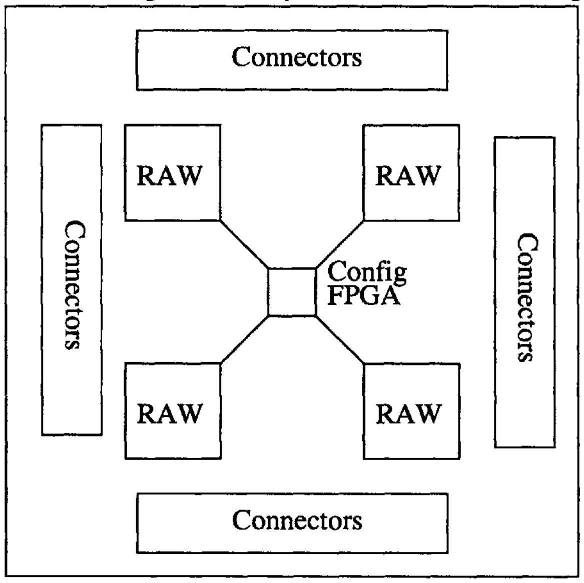
3.4 Board Types

In order to enforce modularity, two separate board types exist in the Raw Fabric hardware system. The Quad boards house the Raw processors, and the I/O boards provide I/O for the Raw processors.

3.4.1 Quad Boards

The Quad boards, also known as Processor Boards, derive their name from the fact that there are four Raw processors (for a total of 64 tiles) per board. The Quad boards contain pads to which the Raw processors are soldered. In addition, they contain traces that connect neighboring Raw processors such that the tiles are connected in roughly the same way that Raw processor tiles are wired internally. Figure 3-1 demonstrates what the Quad board looks like schematically.

Figure 3-1: High Level Quad board block diagram



3.4.2 I/O Boards

In order to accommodate the large area required by all of the I/O components, a separate board was introduced for the sole purpose of providing I/O for the Raw processors. This design allows for debugging the Quad boards and I/O boards separately.

Figure 3-2: High Level I/O board block diagram

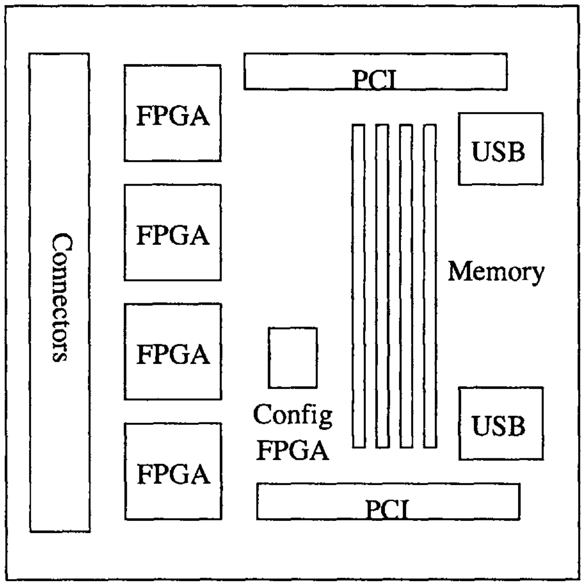


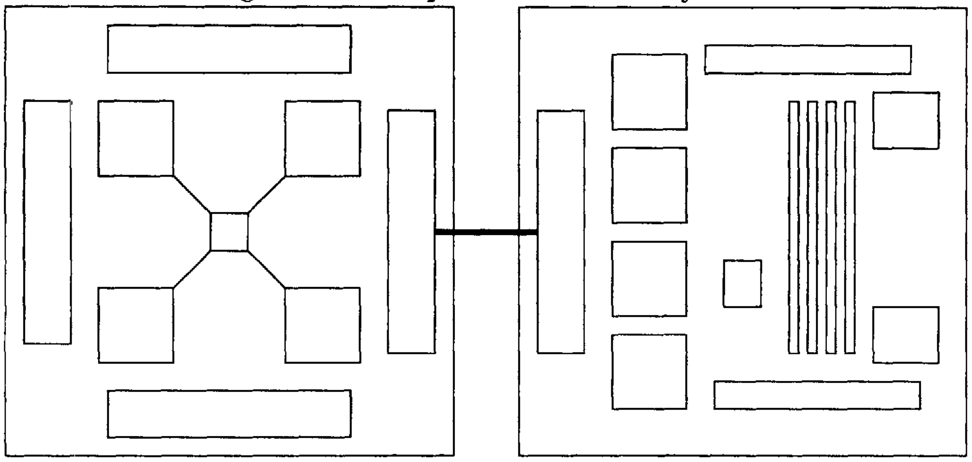
Figure 3-2 shows the features supported by the I/O board. Specifically, the I/O board has PCI connectors, memory connectors, and expansion connectors. In addition, the I/O board has a plug-in connector for a USB plug-in board.

3.5 Board Configurations

The use of modular boards allows for many different board configurations. This strategy allows us to scale up the size of the Fabric incrementally.

The simplest possible board configuration consists of one Quad board and one I/O board (see Figure 3-3. The I/O board may be connected to the Quad board on any of the four sides (North, East, South, West). In fact, the I/O board should be rotated around to test that the connections on each side of the Quad board function correctly.

Figure 3-3: Simplest Raw Fabric system



The next logical step up is to add more I/O boards to the system. A system may contain one Quad board and as many as four I/O boards to allow for maximal I/O possibilities (see Figure 3-4).

Once the functionality of the simple Fabric board configuration is verified, then larger board configurations are more likely to work. The next step up is to Fabric system with two Quad boards. The most important step here is to verify that North-South tiling and East-West tiling of two Quad boards functions correctly.

After the two Quad board system is verified, arbitrary tiling is theoretically possible. The software infrastructure prefers square Fabrics (though rectangular Fabrics should theoretically work), so the logical steps are first a 2 Quad board by 2 Quad board system. Finally, a 4 Quad board by 4 Quad board system may be constructed, yielding a total of 1024 tiles in a single Fabric system (see Figure 3-5).

3.6 Board Connections

A drawback of the multiple board design is the need for connections between boards. The use of connectors is a liability because it is a notorious source of errors.

The use of connectors is particularly costly in the Fabric system because of the number of connections required. A tile on one Quad board should be able to communicate with a tile on an adjacent Quad board in exactly the same way it communicates

Figure 3-4: Raw Fabric with 1 Quad board and 4 I/O boards

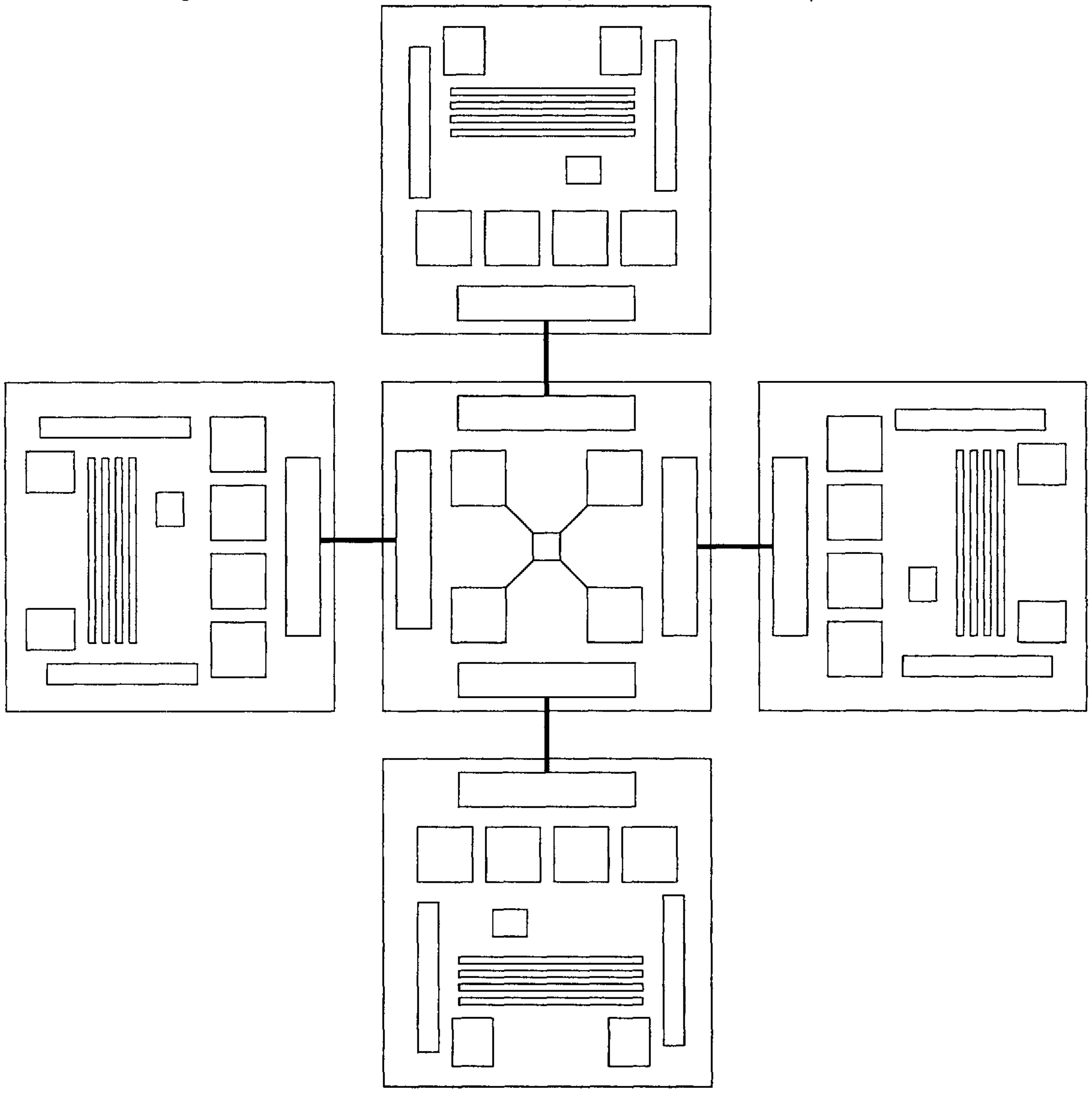
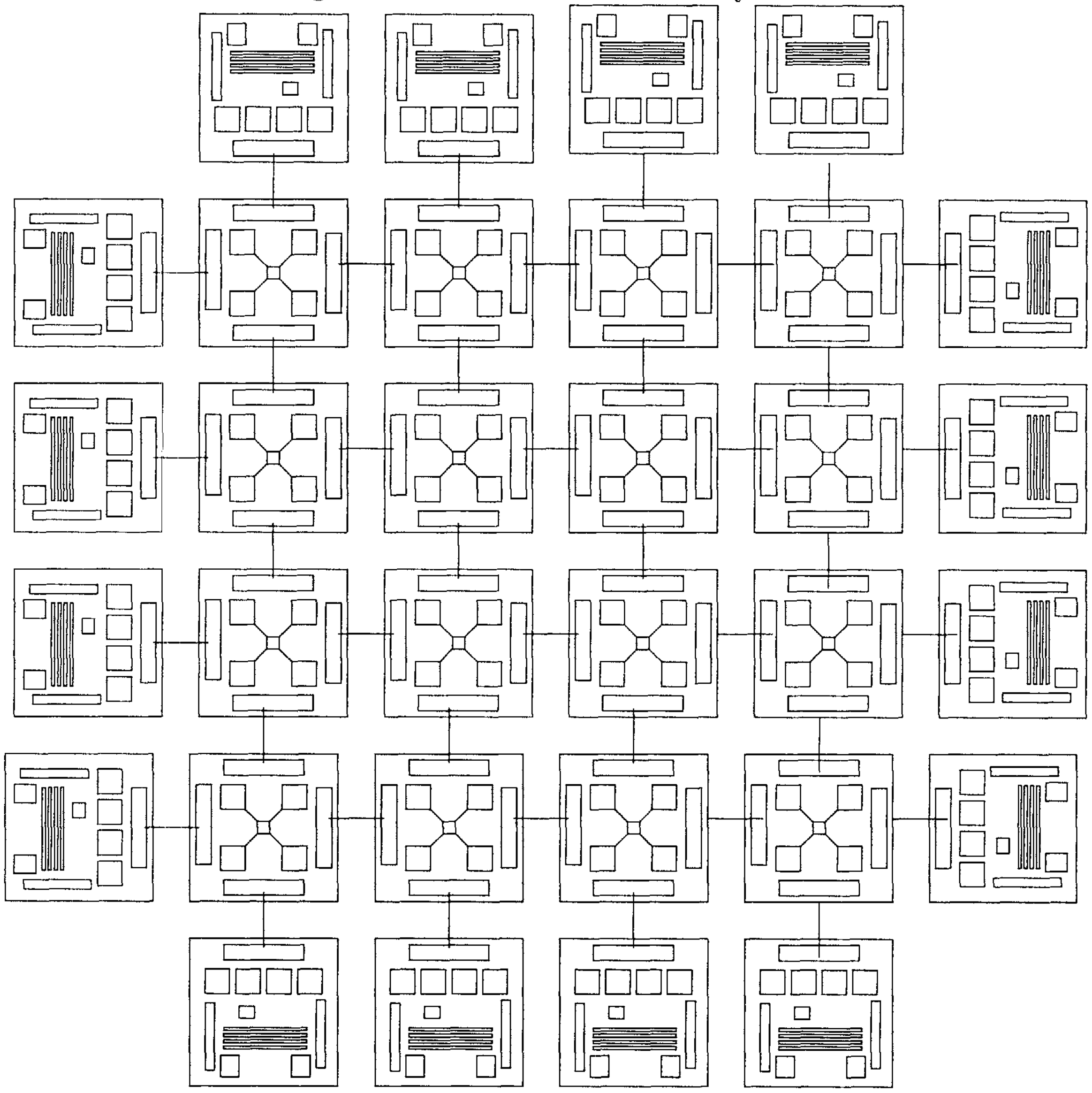


Figure 3-5: 1024 Tile Raw Fabric system



with the rest of its neighbors on the same Quad board as itself. As a result, each link between tiles across two Quad boards requires a 32-bit data bus in each direction, along with the relevant control signals. In other words, a large number of connections is required.

Thus, while there are many benefits of using a multiple board, modular design, there are also disadvantages. Section 4.2.1 delves into greater detail on the choice of connectors and its ramifications.

Chapter 4

Design Decisions

This chapter describes specific design challenges involved in creating a large Fabric system and the decisions made to resolve them. The challenges faced range from how to distribute signals reliably to how to prevent the board from bending from mechanical pressure. The scalability, performance, and reliability of the system were deemed the most important goals to keep in mind.

4.1 Signal Distribution

The scalability goal of the system provides a challenge to distributing the vital power, reset, and clock signals to each of the boards. Careful consideration was taken to make sure that the system would function reliably, no matter how small or large the board configuration.

4.1.1 Power Distribution

Power distribution presents the challenge of supplying a large amount of power to a large number of boards. Adequate power supply is a critical issue because some of the most difficult bugs to track down are power-related. Therefore, heavy emphasis was put into designing a robust power distribution system.

Supplying Power to 32 Boards

Each board in the multiple-board Fabric system must be powered from some source. Since a Fabric system may contain as many as 32 boards, it is inconvenient to power each board by its own dedicated power supply, since a user would need to switch on 32 separate power supplies to use the Fabric. In addition, 32 boards would need to be switched off and switched on in order to perform a hard reboot.

Instead, a power distribution network was introduced to supply power to each board. In this network, each Quad Board is connected to each of its four neighbors, and each I/O Board is connected to its neighboring Quad Board. Since the power signals are non-directional, any board may act as the power "source." This board is in turn powered by one or more large external power supplies.

This solution allows there to be a single power supply, which is much more convenient. Powering the Fabric on and off involves pressing a single button, and procuring the supply involves buying only one part.

Supplying Large Amount of Power

The enormous power requirements of 64 Raw processors in a single system presents a unique challenge to the Raw Fabric system design. The power usage of 64 Raw processors was estimated at 1280 watts, so it was necessary to design a system that would accommodate this much power flowing through the system.

First, it was infeasible to distribute the power through the same cables and connectors as the rest of the signals. While this is true in most systems in general, it is especially relevant in our case because the high signal density (please keep in mind there are roughly 600 connections between each pair of boards, see Section 4.2.1) coupled with the huge power requirement. Instead, much larger cables and connectors handle the distribution of power.

However, even the large power cables are limited in the amount of current they may conduct. Therefore, it was necessary to choose the distribution voltage carefully. For example, if we were to choose a distribution voltage of 5 V, then the theoretical maximum current running through the cables would be prohibitively high at 256 Amps. On the other hand, an extremely large voltage also leads to safety issues. In the end, we chose a distribution voltage of 48 V, which allows the current to conform to the connectors' current threshold rating. To help keep the current low, we also decided that if building a large Fabric, multiple power supplies would be used so that one power supply does not need to supply all of the power.

4.1.2 Reset Distribution

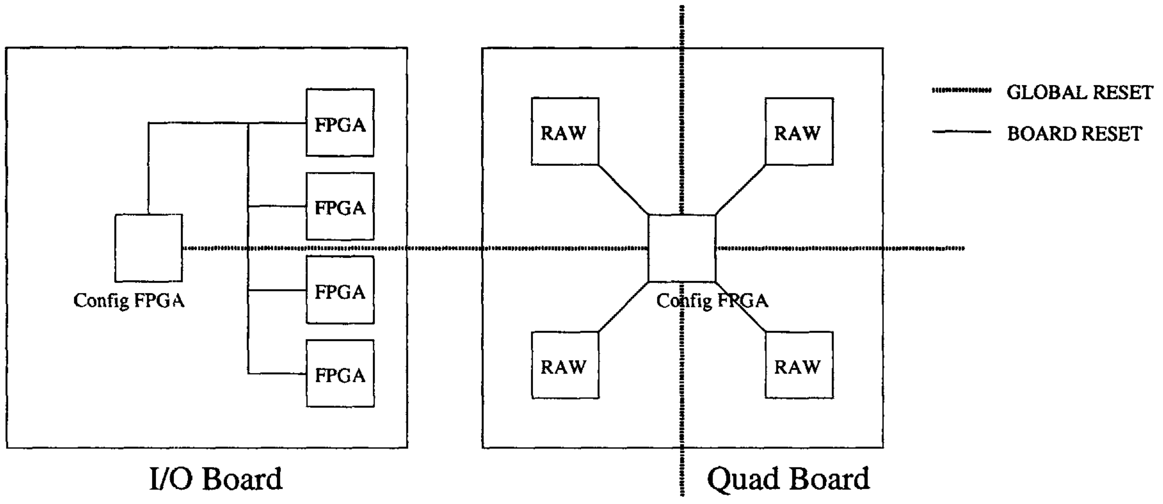
The proper design of the reset system is challenging because all of the boards in the system must receive the same signal, but any FPGA on the I/O board should be able to initiate a reset. The main challenge comes from the fact that the Fabric system should be flexible enough to accommodate anywhere from 1 to 16 Processor boards.

The reset signal informs FPGAs and Raw chips to reset their state, so that programs can execute from a known "clean" state. The reset signal must be propagated throughout the entire Raw Fabric system. In other words, it should not be possible for one board to be reset but another board to remain running.

One approach considered might be to have a global reset net that all FPGAs and Raw chips are connected to, and have them pull down. However, this approach is not readily scalable, because the external resistor value with which to pull up the signal changes as more boards are added. Since we are using the same design for each board, there must be a pull-up resistor on each board. However, the equivalent resistance of the pull-up decreases with each board added. It is therefore difficult to make one design that works for a one Quad board system that is also guaranteed to work for a sixteen Quad board system.

Instead, our reset system is a hierarchical one. First, there is a global reset signal that runs between each pair of neighboring boards' config FPGAs. Next, the config FPGA on each board distributes a local board reset signal to all the other components on the board. Figure 4-1 demonstrates the hierarchical layout of a one Quad board, one I/O board Raw Fabric system.

Figure 4-1: Reset Signal Distribution



Inter-board Reset

The top-level reset scheme is a directional one. Each quad board receives reset signals from all four neighboring boards; however, it uses only one of them as its active input. Our implementation uses DIP switch settings to choose which direction it receives its input reset signal. When a board is reset, it outputs the reset signals to all neighboring boards except its active input direction. This scheme allows a configurable path by which the reset signals are propagated across the Fabric.

Note that the reset propagation is a sequential process: the boards which receive the reset signal first are reset prior to the subsequent boards. This design allows there to be reset skew: different parts of the Fabric receive the signal to reset at slightly different times. This side effect was deemed admissible because there is no pressing need to have all processors reset at the same time, since the design of the software system is careful about beginning code execution at the same time.

Intra-board Reset

The reset signals on the Quad boards and I/O boards are handled differently, since I/O board FPGAs are also allowed to initiate resets whereas the quad boards only propagate the signal to its neighbors.

Quad Boards In the Processor boards, the reset signal that the config FPGA triggers the board reset. The Raw processors are not allowed to initiate a reset, so the reset signal is simply initiated by the config FPGA only. Each of the four Raw processors has its own reset net connected to the config FPGA. This is an active high signal that uses the HSTL_I signaling standard.

I/O Boards On any given I/O board, the reset signal is an active low signal. In other words, the reset signal normally rests at a high voltage by way of a pull-up resistor (all FPGAs tristate the pin). Any FPGA wishing to initiate a reset may pull the reset signal low. The active low choice allows multiple FPGAs to initiate a reset without bus contention. More importantly for us, this choice allows disconnected boards or unprogrammed FPGAs to default to a non-resetting state. If the config FPGA did not receive the reset from the neighboring Quad board, then it propagates the reset to the neighboring Quad board. The stipulation that the reset did not come from the neighboring Quad board makes sure that reset signals bouncing back and forth between the Quad board and I/O board are not possible.

Typical Reset Propagation Path

Figure 4-2: Reset Signal Distribution

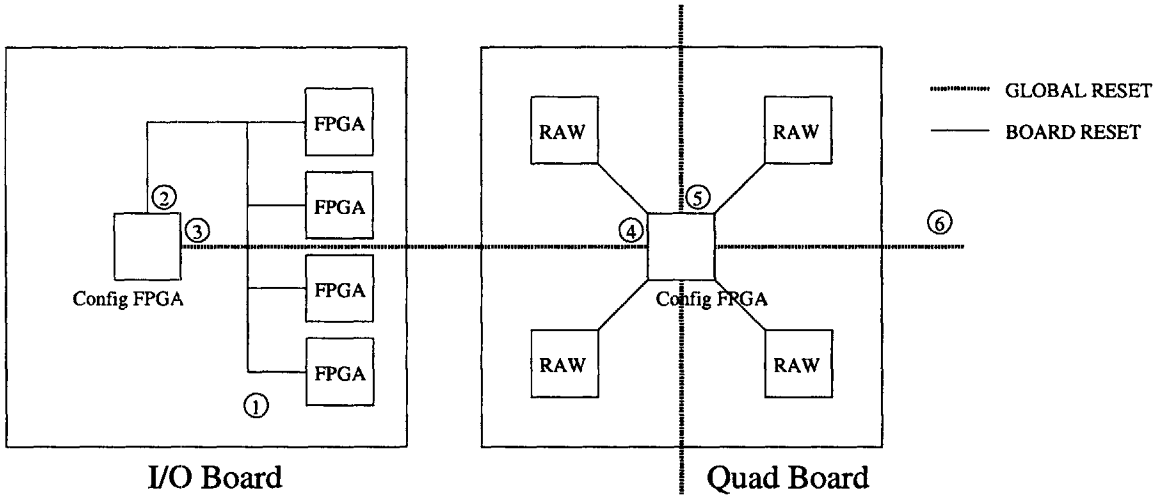


Figure 4-2 illustrates the typical reset signal path. This example will walk through the most common situation that occurs before booting code into the Raw Fabric system. The numbers in 4-2 match up with the following 5 steps:

1. First, the reset signal is requested by the host computer. This is achieved by sending a special reset packet to the USB Controller on the I/O board. The FPGA receives this packet and pulls the board reset signal low.

2. This signal causes all other FPGAs on the I/O board to reset, including the config FPGA.

3. When the config FPGA is reset, it also asserts a reset to the Quad board it is connected to.

4. If this Quad board was properly configured to receive the reset signal from the I/O board, then it will accept the reset signal. (If the Quad board was not configured to receive the reset signal from that direction, then it simply ignores the signal.)

5. It then resets the Raw processors on the board, and it also propagates the reset signal to all directions except the direction it received the signal originally (so that the reset signal isn't being bounced back and forth between the two boards).

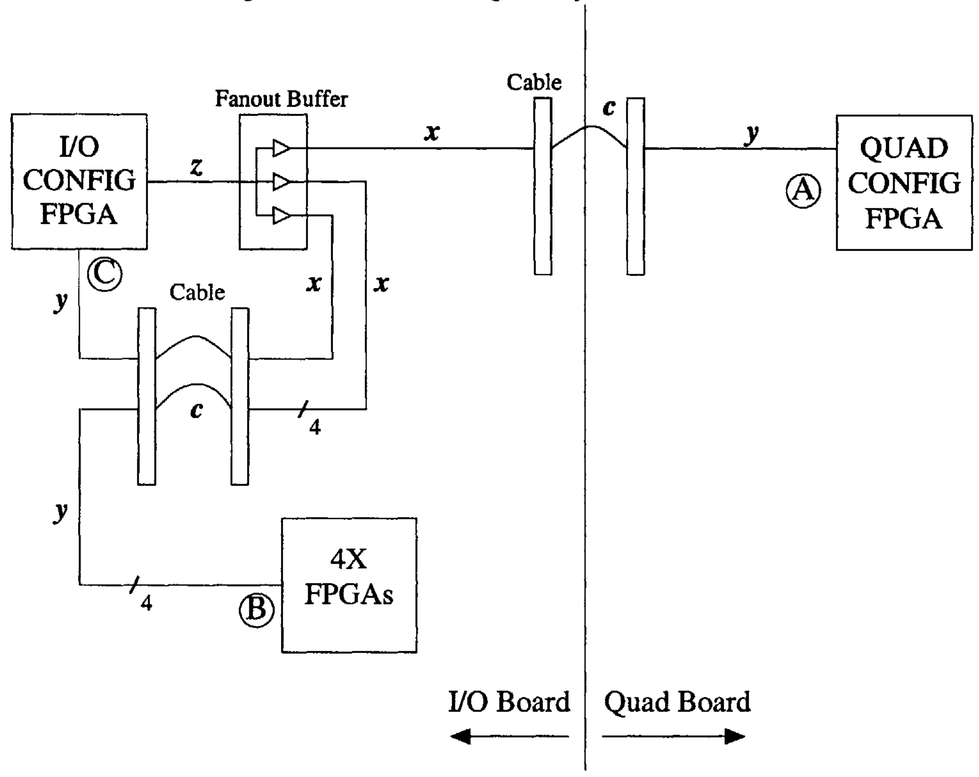
6. The neighboring I/O boards are reset, and if the neighboring Quad boards are configured to receive the reset signal from the correct direction, they will also be reset.

The reset signal propagates throughout the system until the reset signal is not received by any more new Quad boards. The reason the reset terminates and does not bounce forever is because the Quad boards are configured to trigger a reset only on one side.

4.1.3 Clock Distribution

The clock distribution system was one of the trickiest parts of the Raw Fabric to design because great care must be taken to ensure that all boards receive a synchronized clock signal. This synchrony is critical for data to be transmitted correctly.

Figure 4-3: Clock Signal Synchronization



Point-to-Point Synchronization

To solve the problem of synchrony between each pair of boards, we employed Digital Clock Manager (DCM) components available in the Xilinx FPGAs. Using a feedback loop, we de-skew clocks and make sure the phase is aligned at both locations.

Instead of creating a large feedback loop that creates a loop between two boards, we essentially fake the clock signal to the destination board. Each clock signal needs to be replicated using a fanout buffer to produce separate signals for each component, to minimize capacitance on the clock signal line. These separate signals go to each FPGA on the I/O board and each Raw chip on the Quad board. In order to produce a feedback signal for the DCM to lock to, we designate one of the copied clock signals to go back to the FPGA as the feedback loop. The total wire length of this loop matches the wire length to the config FPGA on the neighboring board (within .1 inches). This careful wire length matching ensures that the clock edge on the destination board occurs at exactly the same time as the clock edge that arrives on the feedback input pin. Using this delay information, the DCM within the FPGA is able to synchronize the clocks on both boards.

Figure 4-3 depicts the wire length matching scheme in more detail. Note that the wire length from the I/O config FPGA to all 3 points of interest (A, B, and C) are the same (z+x+c+y). This is achieved by matching the lengths of each segment. In particular, notice that we feed the signal across a dummy cable within the board, with the same length as the inter-board cable, that connects right back to the I/O board. The trace from the connector back to the clock feedback input pin is again the same length as the trace on the destination board from the connector to the FPGA input pin.

This approach is preferred over a single large feedback loop because it is scalable. Using this approach, we are able to synchronize an arbitrary number of boards together. This flexibility would be much harder to achieve by using a single loop approach.

Clock Propagation Path

The approach for distributing the clock is very similar to the approach for distributing the global reset signal. Any given quad board may receive its clock input from any of the four directions, as well as externally from a connector. The Quad board then sends a synchronized (using the DCM as described in the previous section) clock signal to all other directions. As with the reset settings, the direction of input is determined by DIP switches in the current implementation. Any given I/O board may receive its clock signal from either the Quad board it is connected to or externally from a connector. If it receives the clock signal externally, then it also outputs the clock to the Quad board it is connected to; otherwise it does not.

In this scheme, it is possible for any board to be the "root" of the tree. Since the signals are all synchronized, it does not matter which board acts as the clock source.

4.2 Mechanical Considerations

Due to the size of the Fabric System, various mechanical issues become problems that must be overcome. Many of these aspects were pondered during the original design of the system, but others were rectified only after board revision.

4.2.1 Connections

The number of data signals presents a unique challenge in the Raw Fabric system. Because each Raw tile contains a 32-bit port bus out and a 32-bit port bus in, each side of an 8x8 array (each Raw Processor board houses 64 tiles, please see Section 3.4.1) has (32+32)*8 = 512 data signals. Add in the 5 control signals for each port, and we have nearly 600 separate wires that must be connected between any two neighboring boards.

Connectors

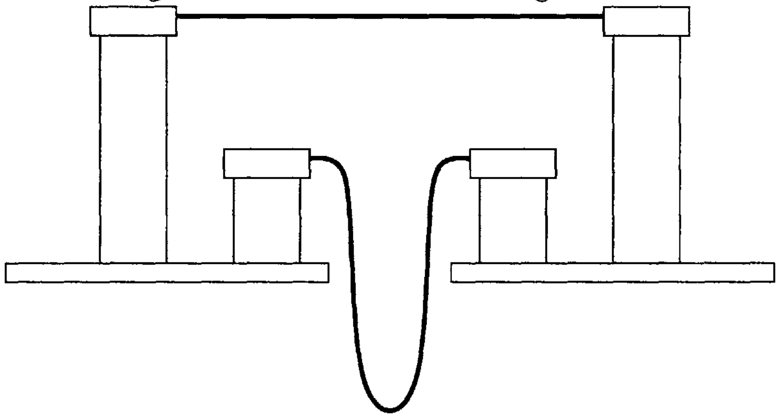
The selection of connectors is greatly limited by the signal density that is required to accommodate the number of signals required. We chose to use specialized MICTOR Blue Ribbon cables. These cables feature the high density we require. In addition, the cables feature impedance-matched coaxial signaling, providing high signal quality and speed.

Even with these extremely dense connectors, we still had to "stack" the connectors to attain the number of connections required. Figure 4-4 shows the side view of the connector stacking. There are two sets of connectors: a tall and a short. The short connectors are closer to the board edge. The idea is that the cable for the short connectors is installed first; and then the cable for the tall connectors is installed on top. This arrangement allows us to effectively double the number of wires connected.

Retention

Because of the high density and the previous experience with these cables in the Raw Handheld system, we designed the board to accommodate a retention mechanism to

Figure 4-4: Connector Stacking Scheme



apply constant pressure to the cable. This proved to be an invaluable component to having a functional system, because unconnected inputs to the Raw processor register as active. When a cable is not seated properly, data ready signals may be asserted, injecting spurious data into the processor before it boots.

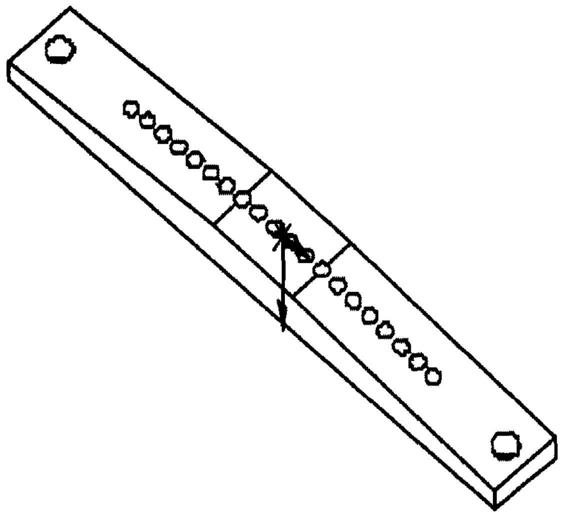
The retention mechanism we designed involved a small metal bracket to be screwed on over the cable connection. The retention bracket is screwed in on either side in mounting holes in the board. Since the specifications of the connector's manufacturer indicate that pressure should be applied in the middle of the connector, our retention bracket design was curved such that the middle would receive more pressure. Figure 4-5 shows the mechanical design of the retention brackets.

4.2.2 Heat Sink Board Flex

Despite our efforts to choose the lightest possible heat sink for the Raw Processor boards, the fact that there are four heat sinks on one board causes the board to sag in the center.

In effort to combat gravity, in the second revision of the board we introduced a new method for mounting the board. Instead of using standoff screws directly resting on the lab bench, we mounted the entire board on a support board. In our case, our board was made of wood, but other rigid materials were considered and would likely provide the same support. Standoffs of equal length were placed between the processor board and the support board, and all heatsink and standoff screws were

Figure 4-5: Retention Bracket Mechanical Design



passed straight through. In this manner, the board was held rigidly flat by the screws.

To balance the height and to provide support, this approach was applied to the I/O boards as well.

4.3 Tilability

In order to limit the number of boards to be designed to two (the I/O board and the Quad board), some ingenuity was needed to ensure that the two board designs could be tiled together to produce any Fabric configuration specified in Section 3.5.

Figure 4-6 demonstrates how tilability is achieved among the Quad boards. Note the arbitrarily labeled connectors between the top two boards. The outer connectors connect to one another and the inner connectors connect to one another.

However, things become more complicated when we introduce the I/O board. We would like to use the same I/O board design for all I/O boards surrounding the Quad boards. However, when we rotate the I/O board to the opposite side, the connectors become mismatched.

The following example illustrates the difficulty introduced by rotation. Figure 4-7 shows the I/O board connected to a Quad board in the connector naming scheme

Figure 4-6: Tilability of the Quad board

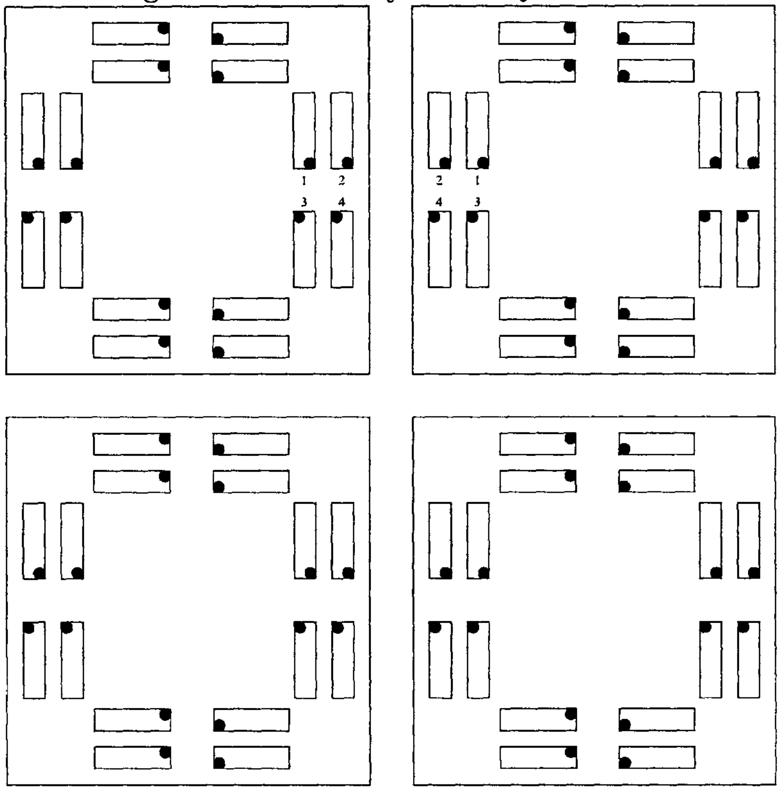
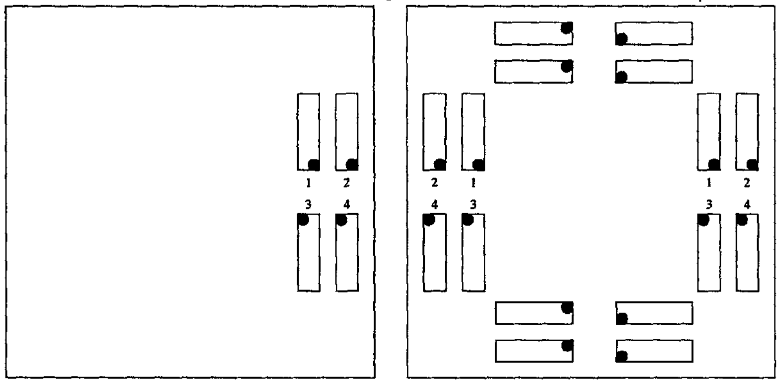
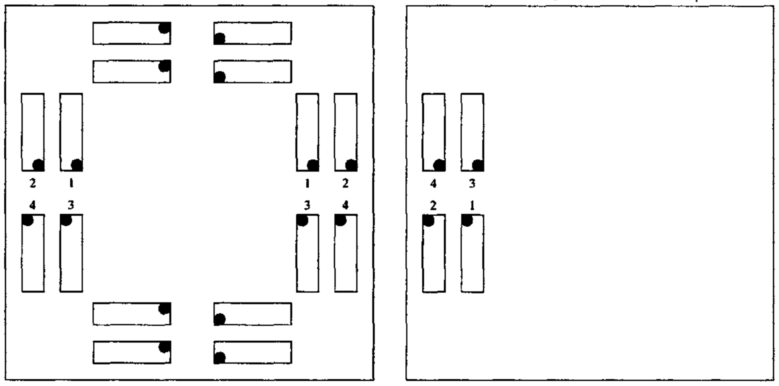


Figure 4-7: Connector Matchings between Quad and I/O board



in Figure 4-6. Observe that the connector numbers all line up correctly between the I/O board and Quad Board. However, when we rotate the I/O board and move it to the opposite side, we find that the connectors no longer match up. However, we do find that the black dots, which represent pin 1, do match up in the correct way.

Figure 4-8: Connector Mismatch between Quad and I/O board



Thankfully, the use of FPGAs on the I/O board partially solves the I/O board tilability problem. Since the data pins are connected to the Xilinx XC2V3000 FPGAs, we can simply re-configure the pin assignments to accommodate connecting the I/O board to any side of the Quad Board. Unfortunately, this choice leads to the need to having four different versions of all of the firmware.

However, not all signals on the connectors route directly to reconfigurable pins on the FPGAs on the I/O board. The global clock and reset signals do not route to the large FPGAs. Even worse, the clock is generated by a fan-out buffer (see Section 4.1.3), so the connection is hard-wired into the board circuitry. For these signals, a more careful approach allows tilability to still be achieved. Basically, if we mirror the top connectors (for example, in Figure 4-6, connectors 1 and 2) over a horizontal line onto the bottom connectors, analogous pin assignments should match up. This symmetry helps us find a set of pin assignments that allows for the same I/O board design to be used in the entire Fabric system. This symmetry is difficult to explain without a figure, but a full explanation is beyond the scope of this document. To fully appreciate the complexity of the problem, please consult the Raw Fabric Revision 2 schematics (the Revision 1 implementation was incorrect).

Chapter 5

Implementation

This chapter describes the implementation of the Raw Fabric system in detail. The purpose of this chapter is to provide the reader with an idea of what is involved in building a flexible system of this magnitude.

Specifically, the Raw Fabric system makes extensive use of reconfigurable logic so that boards may be coaxed into taking on a variety of roles. The development of the logic that resides in the FPGAs is one of the most important portions of the implementation of the Raw Fabric.

5.1 Firmware Development

The I/O Board uses reconfigurable logic to allow for rapid and flexible development. The firmware in the FPGAs must be re-programmed each time the Raw Fabric system is turned on.

5.1.1 Programming the I/O Board

The I/O board FPGAs are programmed through a JTAG chain connecting the five configurable FPGAs as well as a SystemACE FPGA. Each link on the JTAG chain is connected only to its predecessor and successor, and the final FPGA connects back to the beginning of the chain, forming a ring.

The firmware may be programmed either through loading a CompactFLASH card in the slot in the board with appropriate firmware, or by directly connecting a Xilinx cable to the JTAG header on the board and directly programming specific FPGAs.

5.1.2 Programming the Quad Board

The Quad board's configuration FPGA may be programmed either through JTAG or by programming the PROM connected to it. The PROM automatically programs the config FPGA on power-up; however, the firmware may be overwritten by subsequent JTAG programming. The nice feature of the PROM is that its contents are not erased when the board is powered off (unlike the config FPGA).

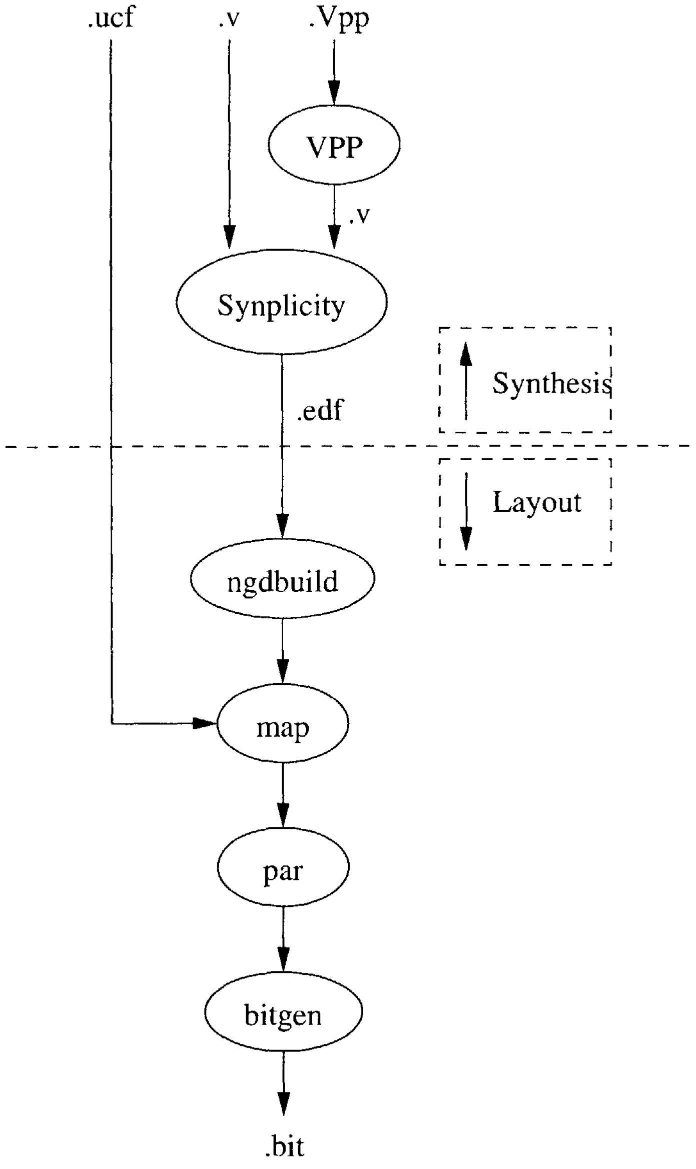
5.1.3 Firmware Writing

Firmware for the Raw Fabric system is developed using Verilog. We have set up a development platform called veribuild_fabric, similar to the veribuild platform for the Raw Handheld system. Veribuild's greatest strength is the ease of creating highly modular code, since it uses a GNU Make structure to include sources.

The Veribuild system automates the creation of firmware. Figure 5-1 shows the steps between Verilog code to bitfiles, which are programmable into the FPGAs. The firmware generation is divided into two main stages: synthesis and layout. During synthesis, a netlist containing wires and nodes is generated from a set of source Verilog files. In layout, this netlist is converted into a binary file with which FPGAs can be programmed.

Before compiling the Verilog code, there is an optional Verilog Pre-Processor step. The Verilog Pre-Processor allows the user to use special tags that extend Verilog's functionality. For example, the Verilog Pre-Processor has a "for" directive that can quickly replicate instantiations and declarations, so that numerous copy-and-paste operations are not required. Please note that Verilog 2001 has since implemented many of these functions, so the need for the Pre-Processor is perhaps not as great as it once was.

Figure 5-1: Veribuild Tool Chain Flow



Synplicity is a synthesis tool used to compile the Verilog code. The code is translated from Verilog RTL into a netlist depicting wires and nodes through a 2-step process consisting of compiling and mapping. At the end of this process, we have a netlist stored in a .edf file.

After synthesis, the netlist and .ucf file together are used to create a bitfile. First, the output .edf file from synthesis is then translated into a .ngo file. Other Xilinx components may also be imported as .ngo components at this point. The .ngo components are then mapped to the specific part during the map step. In addition, the .ucf constraints file is translated into a .pcf file, which are used to let the tools know what the pin mappings and timing constraints are.

Next, the Place and Route (par) step does the actual wiring of the components. During the Place step, the logic components are laid out on the FPGA. The placer optimizes the placement to ensure that timing constraints dictated by a .ucf file are met. Once the Placer has designated the position of all of the logic, the router actually connects the signals running between the logical units. In other words, the placer defines the location of the pieces, and the router wires together the pieces.

After the par stage is finished, its output .ll file is converted into a .bit file in the bitgen step. The .bit file may then be programmed onto the FPGA as outlined in Section 5.1.1.

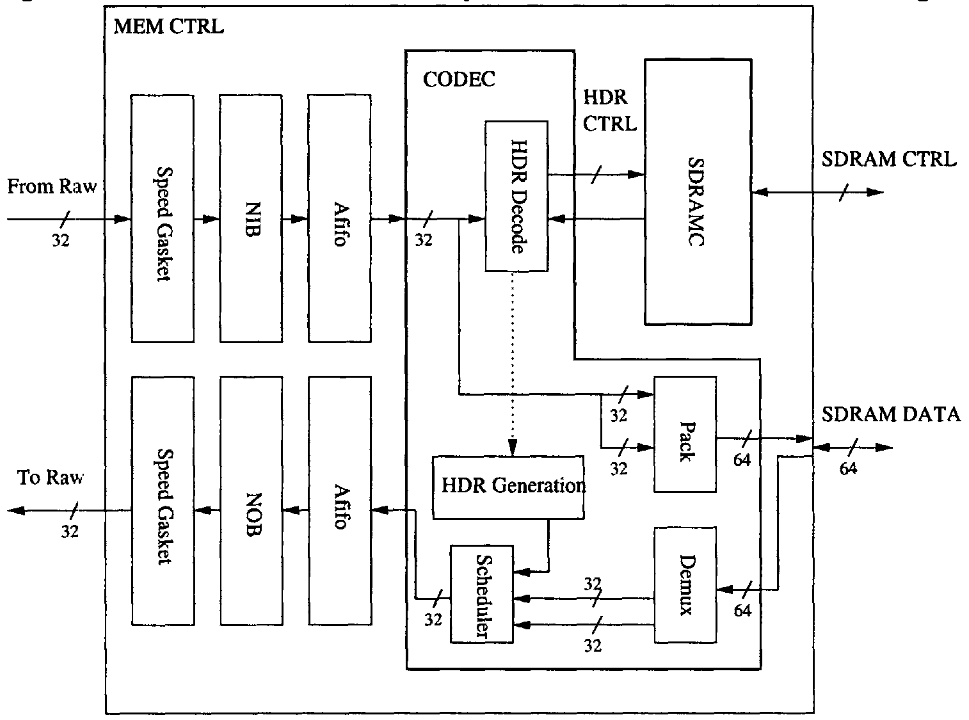
The toolchain flow seems complicated, but a firmware author simply needs to write the Verilog .v files, include the appropriate files and set the desired flags in the Makefile, and type make in the command line, and Veribuild takes care of the entire process. This simplicity allows for rapid firmware development.

5.2 Memory System

The memory system is critical for running any useful programs on the Raw Fabric system. Though it was possible to write simple assembly programs that did not use the memory system, any useful programs will require significant use of memory. For example, any C program needs the memory system to set up its stack and heap.

5.2.1 Firmware

Figure 5-2: Raw Handheld Memory Controller Firmware Block Diagram



The memory controller firmware is based upon the Raw Handheld memory controller firmware. Figure 5-2 is a block diagram of the different functional units within the memory controller. First, the speed gasket interfaces with the Raw processor. The data is then placed in a First-In-First-Out (FIFO) queue dubbed the Network Input Block (NIB) to be processed by the memory controller. The SDRMC module is a finite state machine (FSM) that retains state information about the memory controller. Finally, the codec block simply translates data into a form that the memory modules are able to understand.

The most prominent changes to the firmware between the Handheld system and the Fabric system include support for different size memory modules and the adjustment of the memory initialization and refresh times. Unfortunately, these changes require a complete firmware recompile every time they are adjusted.

5.2.2 Layout

The layout configuration of the memory system in the Raw Fabric system is roughly sketched in the Raw Specification, page 46 [11]). The organization of the memory system is beyond the scope of this thesis; please refer to the Raw Specification for further details.

5.2.3 Debugging in Revision 1

Although the specification of the layout dictates either 32 or 16 ports, in actual implementation the board flex problems (See Section 6.1) prevented all the ports from functioning. As a result, a great deal of effort was expended in debugging the memory system in tracking down the root of the errors.

During the debugging process, a suite of memory tests was written in assembly to test various patterns on various ports. The patterns include all ones, all zeroes, striping a one across all bits, and writing the address out to memory.

The bugs were found to be both on the Raw-to-memory controller and the memory controller-to-memory module links. Disturbingly, many of the errors were able to be temporarily rectified by physically pushing down on the relevant FPGA on the I/O board. This led to the design changes as outlined in Section 6.1.2.

5.2.4 Temporary fixes for running C programs

Though we could not use a full 16 ports, we were nonetheless able to set up the memory system such that we could give each tile stack and heap space. This was achieved by modifying the software system temporarily. Instead of using the default configuration depicted in the Raw Specification [11], we overrode the configuration with our own configuration. In our configuration, the ports with known problems were avoided, and those tiles for which we were writing running code on received a larger proportion of the total memory in the system. This configuration was a critical step to our ability to run fully functional C code on our Raw Fabric system.

5.3 USB Interface

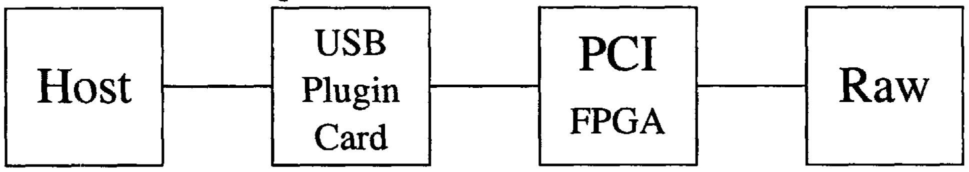
The USB interface allows the Raw processors to communicate to an external host computer. This is vital for any interaction between the Raw processors and the outside world.

The USB controller was one of the first firmware modules to be implemented for the system, because it allows us to stream boot code into the Raw processor. Without the USB connection, the only way to stream boot code would be to manually compile it in the I/O board FPGA firmware. Unfortunately, this is not a viable long-term option, since the amount of data that could be stored in the FPGAs is relatively small.

5.3.1 USB Booting

The USB system can be used to boot the Raw processors. Figure 5-3 shows the path that boot code traverses. First, the code is compiled on the host machine. The code is then sent to the USB driver on the host machine. The USB driver communicates to the SX2 USB controller on the plug-in board via the USB bus. The SX2 USB Controller is a chip that converts USB packets into parallel data form, and it sits on a plug-in board off of the Fabric.

Figure 5-3: Fabric USB Boot Path



The USB plug-in board is connected through a Teradyne connector onto the Raw Fabric I/O board. There are two USB plug-in board connector headers on the I/O board, allowing us to accommodate up to two USB plug-in boards per I/O board. Each USB plug-in board is connected to one of the two PCI FPGAs.

The PCI FPGA is takes the 16-bit data bus from the SX2 chip and packages it into 32-bit data packets to be transported to the Raw processor. The data packets include not only 32 bits of data, but also a header word that is used for routing the packet. The packet is introduced into a routing network within the firmware, and at the other end it reaches the speed gasket, which communicates directly to the Raw processor. At the end of the speed gasket, the header is dropped and the data is transmitted directly to the Raw processor. In other words, the routing network's header word never reaches Raw.

The Raw communication protocol is such that for each data word the Raw processor receives, it sends an acknowledge signal (dubbed "Yummy"), indicating it has received data (see the Raw Specification, page 17 [11]). This protocol is introduced to throttle data transfer, since the sender knows how many empty queue slots the recipient has at any point. The yummy signal must be propagated through every stage of the communication process. However, in order to save USB bus bandwidth, these yummy signals are bundled together and sent at once. The Raw processor receives data and sends a yummy for each data word, but the USB controller firmware tallies these yummy signals and sends a single acknowledge signal back to the host computer as a single USB packet.

5.3.2 SX2 Configuration

The SX2 chip is used to convert parallel data into packets that adhere to the USB protocol. The SX2 chip requires configuration, which is achieved by the "USB" module in the USB firmware. This firmware sends a sequence of commands that programs the SX2 during power-up.

5.3.3 Host Interface

The USB system makes the host interface possible. The host interface allows the Raw processors to call common I/O library functions such as println and file I/O functions. In order to implement this functionality, it was necessary to implement bi-directional communication over the USB link. In addition, it was necessary to modify the software infrastructure so that the boot code of the Raw Fabric would know which port the host interface is connected to.

5.3.4 Future USB Implementation

The implementation of multiple USB connections to one Raw Fabric in the future would be useful to increase the maximum bandwidth possible through the USB connection. This change would allow for increased performance in bandwidth-limited tasks, such as video streaming.

In addition, different header/data protocols have been explored to improve the maximum effective bandwidth.

Chapter 6

Lessons Learned

This chapter describes the most important lessons learned during the course of hardware development for the Raw Fabric. These lessons range from how to build the physical system to how to design the circuitry, but have one thing in common: they would have been very useful to us if we had learned them before building the system. The aim of this chapter is to give the reader an idea of what pitfalls to avoid if designing a system similar to the Raw Fabric.

6.1 Mechanical Considerations

The mechanical aspect is often overlooked during the design of a hardware system. We ran into difficulties with the first revision of the board that warranted changes in the second revision. Though the need for a second revision was not necessitated by mechanical considerations, many of the changes helped make the second revision much more robust than the first.

6.1.1 Unreliable Cables

The use of cables to connect two boards adds a significant variable. Our case was especially severe because we needed to resort to special-purpose, high-density cables. In addition, we were not using the cables in the manufacturer-recommended fashion.

The most prevalent problem we faced was with flaky connections between the I/O board and the Quad Board. As described in the Section 4.2.1, there are over 500 signals running between the two boards. If any one of these connections is not connected, then we receive erroneous results. More importantly, if the boot code is streaming into the Raw Fabric over one of these connections, then the Fabric will be not boot.

As a result, we needed to use retention brackets, which introduced their own set of problems. If we were to redesign the system, we would consider reducing the reliance on wide buses running between boards. A possible implementation might be to serialize the data and increase the clock frequency of the I/O. Clearly, the reliability of this design would need to be validated before adopting the design.

6.1.2 Retention Bracket Stress

Our solution of clamping down the cables had the unforeseen side effect of causing the board to bend. We believe this board flex may have been a factor in the unconnected FPGA pin phenomenon we observed.

Unconnected FPGA Pins

We suspected that a few of the FPGA pins were not connected to the pads, since some connections between the FPGA and the connectors and some connections between the FPGA and the memory DIMM sockets were broken. After careful inspection of nets that were causing trouble, we found there was a pattern of corner pins of FPGAs to be more susceptible to problems. Board fabrication experts advised that pin connection problems on our FPGAs are much more likely to be in the interior of the FPGA than on the outer corners. Therefore, we conclude that the problem may have been a combination of manufacturing error as well as flexing of the boards.

Changes Made to Improve Connections in Revision 2

In Revision 2, we undertook additional measures to reduce the board stress. We added stiffener components to combat the board flex, and we are also mounting the entire board on a stiff backboard. In addition, we added a host of debug tests on the memory controller side to be performed before we accept the boards as correctly manufactured.

6.2 Schematic Errors

While the mechanical considerations were not sufficient to warrant a second revision of the boards, the schematic errors we discovered could only be rectified by undergoing a board revision.

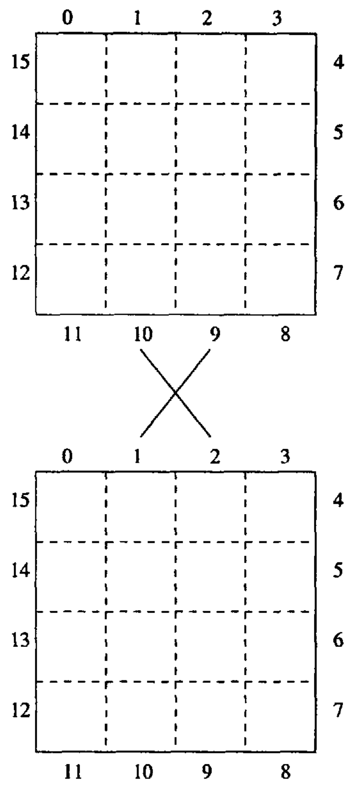
Criss-Crossed Yummies

As shown in Figure 2-1, the middle top and bottom ports are multiplexed because of a shortage in pins. However, a schematic error prevented correct functionality.

The yummy signals are acknowledgment signals in the Raw SIB protocol. As shown in Figure 6-1, the top two middle ports are 1 and 2, whereas the bottom two middle ports are 9 and 10. Unfortunately, in a design oversight, the port 1 yummy is hooked up to port 9, and the port 2 yummy is hooked up to port 10, as shown in Figure 6-1. Instead, port 1 should be connected to port 10 and port 2 should be connected to port 9.

These multiplexed ports share data pins, and the ready signals are routed correctly, so the data is transmitted to the correct location. However, the acknowledge signals go to the wrong tile. This leads to a bug that is difficult to find and difficult to correct.

Figure 6-1: Criss-Crossed Yummies



Unconnected Pins

Pin 39 and pin 40 on connector P19 on the South side of the Quad Board were left unconnected to the connectors in the first version. This precludes the possibility to transmit correct data on the south side of the Quad Board.

Incorrectly assigned control pins

The clock, reset and config FPGA-to-config FPGA bus pins were not assigned correctly. This results in the inability to tile two quad boards with each other. The error also makes it unsafe to connect the I/O board to the west and south sides of the Quad Board without modification. We modified our third I/O board slightly so that output pins would not be driving the same signal, but this was a temporary fix to test whether I/O boards could be connected to the West and South sides (and they can).

Minor Reset Issues

First, the Board Reset on the I/O board was not connected to the config FPGA. Therefore, there was no way for all 5 FPGAs to share the same reset line, defeating the purpose of having a board reset net.

Also, the circuitry on the Quad board did not soften the power-up ringing on the JTAG programming circuitry, so the PROM on the Quad board does not properly program the Quad config FPGA on power-up. Instead, it is necessary for the user to manually depress the JTAG program button on the Quad board every time the Fabric system is powered on.

6.3 Design Lessons

6.3.1 Parallel I/O

We learned the practical difficulty of implementing a large system using solely parallel communication. The number of wires running between boards in this system is simply astounding, and we found the debugging of the connections to be quite tedious. From a debugging standpoint, we would have appreciated a system with fewer connections.

6.3.2 Active high signaling

The problem we found with active high signaling was that in our case unconnected pins register as active. This choice led to the need to ground any signals we did not explicitly mean to assert.

Specifically, the there was a need to terminate the ready and acknowledge pins to the processor to ground if I/O boards do not fully populate the boundary of the Quad boards. If, instead we had used an active low system, then there would not have been a need to install the extra hardware on the periphery of the Quad board.

6.4 Minor Lessons

This section describes a few minor common-sense lessons learned during the development of the Raw Fabric system.

6.4.1 2 Weeks Can Easily Become 4 Weeks

We found that tasks often take twice as long as expected. Fabrication and assembly often run into unexpected delays, and unexpected bugs can destroy any schedule.

6.4.2 Backup Often

rm -r has happened a few too many times in this project. Use of the CVS repository, placing code on a periodically backed up file system, and manual backup are all highly recommended strategies.

Chapter 7

Characterization

This chapter characterizes the Raw Fabric system by providing a few electrical measurements taken on the hardware. The most important aspects of a functional Fabric system include a clock that is synchronized and a reset signal that is distributed throughout the Fabric, so this chapter walks through measurements to make sure that the system is performing adequately.

7.1 Clock Synchronization

As described in Section 4.1.3, it is important for us to maintain a synchronized clock across the entire Fabric so that data may be transmitted and received correctly.

7.1.1 Test Setup

To test the clock delay between an I/O board and a Quad Board, I connected an Agilent Infiniium oscilloscope to each board using a BNC to mini-BNC cable. The mini-BNC connector plugs directly into the debug BNC connector on each board.

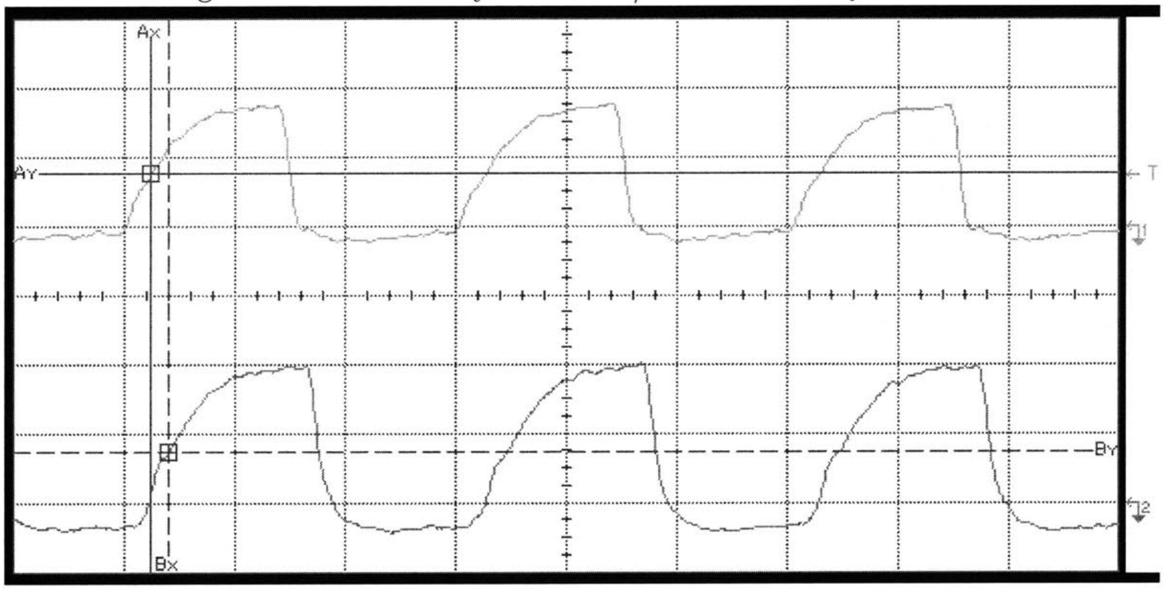
The firmware loaded in the configuration FPGA on each board selects a signal out of a variety of possible debug signals using the user-set DIP switches. Each board was set to output the clock signal after the clock signal input source selection.

This methodology is convenient, but is not necessarily accurate. It is convenient because the BNC cables eliminate the need to probe pins on the FPGA, pins that are very small and would require one to manually keep the probe in place. However, it is not necessarily accurate because there may be variation in the way the configuration FPGA boards are wired. There are no guarantees that the trace lengths to the debug BNC connector are matched by length like the real clock paths are. In addition, there are no guarantees that Xilinx's FPGA place and route programs made the internal routing similar.

7.1.2 Results

Figure 7-1 shows a snapshot of the clock waveforms. The top waveform is the Quad board clock (supplies the clock signal in this setup) and the bottom waveform is the I/O board clock. The scale is 10 ns/division horizontally and 2 V/division vertically. Thus, it is pretty clear that the clock we are distributing has a period of 30 ns (for a frequency of 33 MHz).

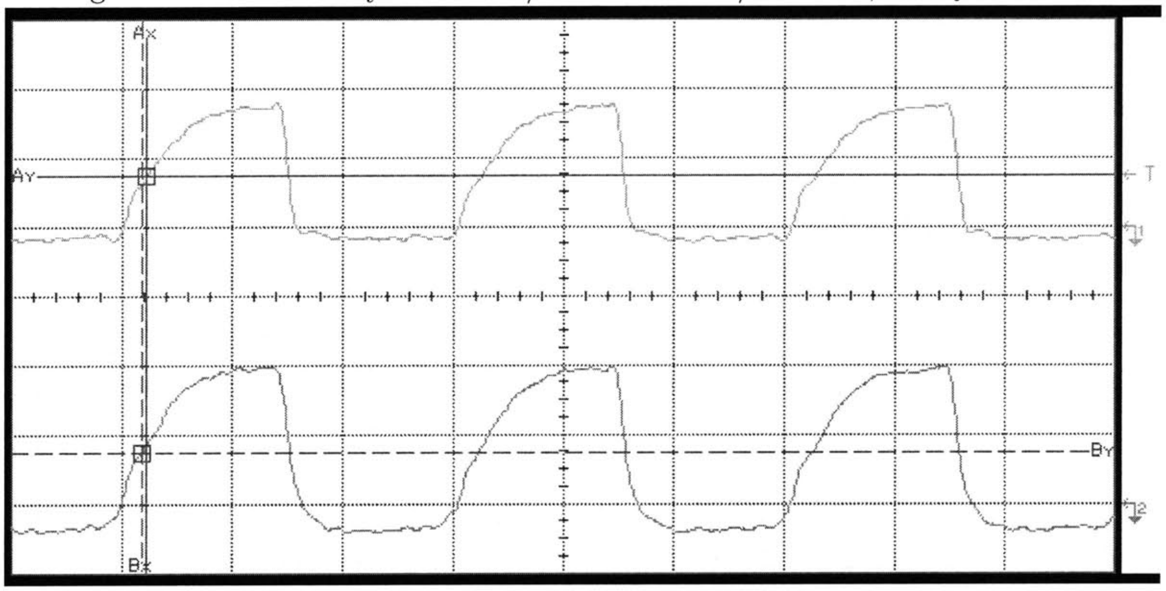
Figure 7-1: Clock delay between I/O board and Quad board



The delay from the top waveform to the bottom waveform was measured to be 1.636 ns. Compared to 30 ns, this is a larger delay than we would like, but it is not large enough to produce errors in the data transmitted. However, we may be in trouble if we build a large Fabric and each hop adds 1.6 nanoseconds of clock skew.

Therefore, I ran a second test of the clock delay from a clock starting from one I/O board to another, going through the Quad Board. Thus, the clock signal traverses an extra hop. Surprisingly, I found that the delay was virtually nonexistent, as shown in Figure 7-2. When measured, the delay was found to be 454 picoseconds.

Figure 7-2: Clock delay between I/O board and I/O board, via Quad Board



Considering the wiring and firmware layout of both I/O boards are identical, it seems that the second test is more accurate than the first. Therefore, the testing suggests that clock synchronization across a large Fabric will be feasible.

7.2 Reset Propagation

In section 4.1.2, it was stated that the reset signals do not need to be exactly synchronized, because the software system has its own strategy for starting programs on tiles at the same time. Although not critically important to the functionality of the system, the following measurements characterize the delay in the reset signal from board to board.

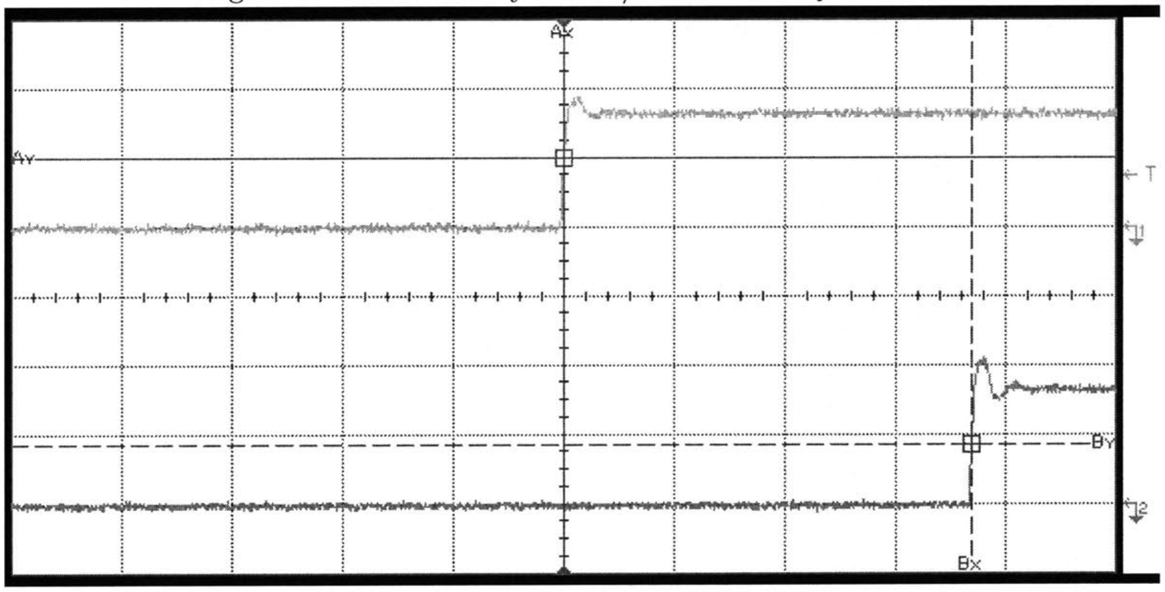
7.2.1 Delay from I/O board to Quad board

In this test, I measure the delay time between the I/O board's configuration FPGA reset to the Quad board's configuration FPGA. I chose to use these two points because they are most easily interfaced with the oscilloscope. This methodology omits a portion of the reset delay, because typically the reset signal originates from the PCI FPGA containing the USB firmware (which propagates the reset to the configuration FPGA).

Figure 7-3 shows the reset signals being triggered on the I/O board and on the Quad board. The top waveform corresponds to the I/O board's reset signal and the bottom waveform corresponds to the Quad board's reset signal. As expected, there is a delay between the I/O board reset and the Quad board reset.

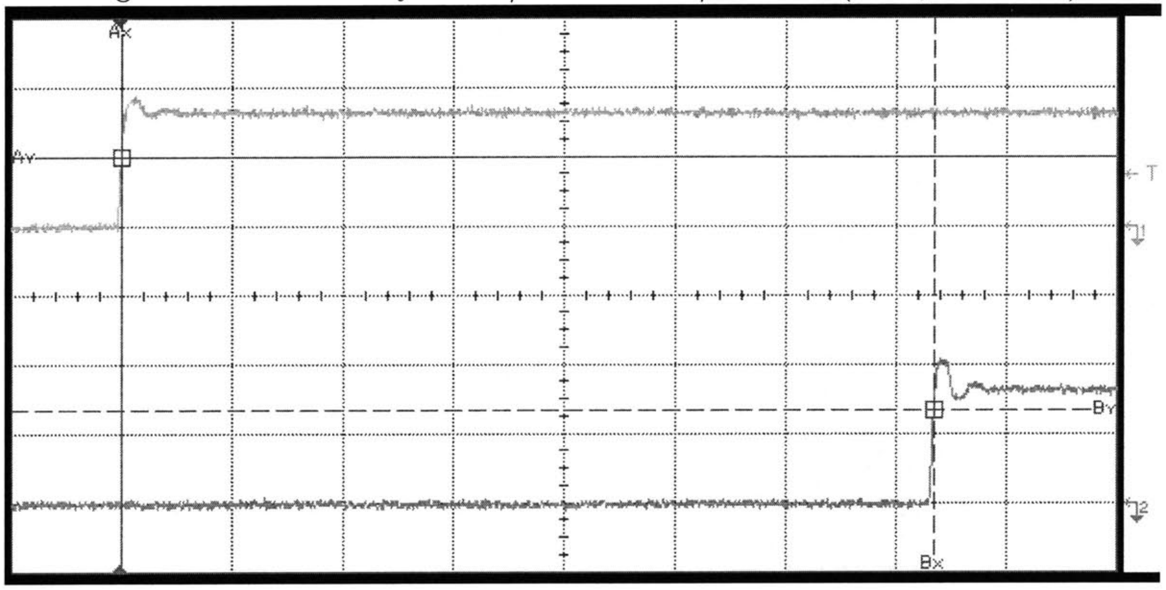
The amount of delay was measured to be 369 ns. This is a rather sizable amount of delay (roughly 60 Raw cycles at 66 MHz). However, again, this delay is inconsequential since the program execution is synchronized in software.

Figure 7-3: Reset Delay from I/O board to Quad board



To extend the reset propagation one step further, I measured the delay between an I/O board and another I/O board connected to the same Quad board. Thus, the reset signal originates on one I/O board, travels to one Quad board, and is relayed to a second I/O board. Figure 7-4 shows the waveforms of the reset signals. The top corresponds to the reset-initiating board and the bottom corresponds to the final recipient I/O board.

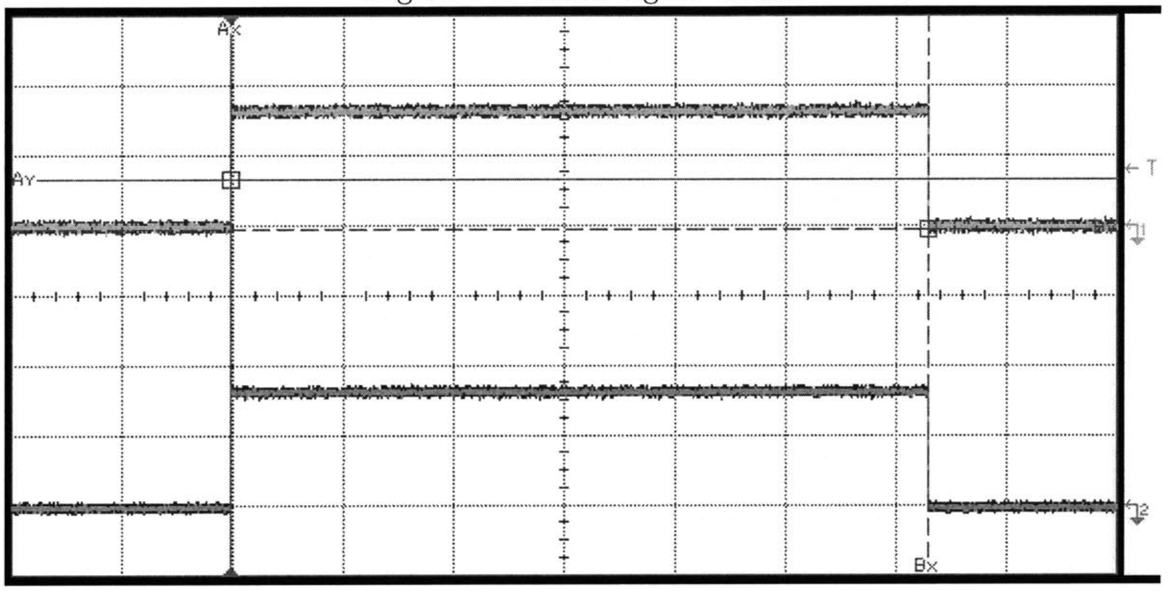
Figure 7-4: Reset Delay from I/O board to I/O board (via Quad board)



The delay was measured to be 734 ns. Thus, each hop of the reset path adds approximately 360 ns to the reset delay. In a large Fabric, the originating I/O board and the board on the opposite corner will be reset at much different times.

On the other hand, to put the delay into perspective, I also measured the full reset signal's width. In firmware, this reset width is controlled by a (rather large) global constant that determines the number of cycles the reset should be held. As shown in Figure 7-5, the reset delay is iconsequential when compared to the amount of time the reset is asserted. The measurement of the width of the signal was 1.26 seconds. We see that at this time scale, the reset signals of both boards appear to be simultaneous.

Figure 7-5: Reset Signal Width



Chapter 8

Conclusion

This section provides a summary of the completed work described in this thesis and a discussion of future work that may be explored.

8.1 Summary

This thesis tells the life story of the Raw Fabric Hardware system, from its inception to its current state.

The Raw Fabric story begins before the Raw Fabric was even designed. Chapter 2 starts the story off by depicting the Raw architecture and the central scalability features that would be extended by the Raw Fabric system. The chapter also enumerates various relevant features of the Raw Fabric's predecessor, the Raw Handheld system, that would be important for the development of the Raw Fabric.

Chapters 3 and 4 describe the design phase of the Raw Fabric hardware system. Chapter 3 focuses on the high level system design and leaves the more difficult design decisions to be explained in Chapter 4. Chapter 4 describes the challenges faced in distributing signals reliably and the solutions utilized to create a robust system.

Chapter 5 provides details on how the firmware system in the Raw Fabric was implemented. The majority of the hardware implementation effort centered about firmware implementation because it was the most time consuming task.

Chapters 6 and 7 move on to the later stages of the Raw Fabric's life. Chapter 6 is a reflection of things that could have been done better had we but known. It provides useful advice to anyone building a similar system. Finally, Chapter 7 provides a verification that the system we designed performs the way we designed it to perform. It shows that the clocks maintain synchronized and the reset is distributed correctly, providing reinforcement that the system will work as the system scales up.

8.2 Future Work

Despite the amount of work that has been done already, the Raw Fabric is far from complete. The Raw Fabric is rife with future research opportunities yet to be explored.

8.2.1 Larger Fabrics

At the time of writing this thesis document, the largest Fabric system constructed is a 64-tile system consisting of four 16-tile Raw processors.

The next natural step is to connect multiple Processor boards together to create an even larger Raw Fabric system. A few details were overlooked in the first revision of the board, but with the next batch of boards, the tiling of multiple Processor boards will be possible. The implementation process should be much smoother in this second iteration given the experience gained from the first.

8.2.2 Performance Profiling

Now that the test platform has been constructed, performance tests can be run on it. Performance tests can be used to strengthen the case that the parallel processing nature of Raw architecture is a viable alternative to improving performance by increasing clock rate.

8.2.3 Parallel Architecture Algorithms Research

In addition, the hardware provides a testbed on which parallel algorithms may be run. The Raw Fabric system could be used as a tool to help researchers evaluate the performance of their algorithms on real hardware.

8.2.4 Large Applications

Finally, the implementation of the Raw Fabric hardware system means that there is hardware on which useful applications may be run. These applications could be used not only to show off the power of Raw's tiled architecture, but also perform useful computation. For example, there is a continuing effort to run the MPEG-2 algorithm on the Raw Fabric. Once this is complete, the application can be used both to showcase a functioning Raw Fabric system, but also to encode real video streams.

Appendix A

Raw Hardware User Guide

A.1 Overview

This document provides implementation details of the Raw Fabric Hardware. It serves as a **reference** for anyone debugging and testing the hardware system.

A.2 FPGAs

A.2.1 Quad Board FPGAs

The Quad Board contains only one FPGA, the configuration FPGA, that must be programmed. The config FPGA handles the following:

1. Clock Distribution
2. Reset Distribution
3. Test Network Messages

Clock Distribution

In order to distribute the same clock to all processors in the system, the Quad Board must provide routing to redistribute the signal, as well as PLL logic to keep the signal synchronized.

The Quad Board may be configured to input the clock signal from one of five choices: North, East, South, West, or External Connector J7. The selection is made by setting the DIP switch SW1; please refer to Section A.12.1 for the specific settings.

The Quad Board config FPGA outputs the clock signal to a fanout buffer that copies the clock signal, and from there the clock is distributed to each of its four neighboring boards.

Reset Distribution

There are two levels of reset in the Raw Fabric system: Global and Board. Both are active low signals. An external pull-up resistor ensures that if no FPGA is asserting the reset signal, the reset net is held high.

A global reset signal runs between each pair of config FPGAs in the Fabric system. The reset is not directional, so either FPGA may initiate a reset.

When a config FPGA receives a reset signal, it may choose to propagate the reset signal to the rest of its neighbors and to the board. The I/O boards do not propagate the reset signal if they received it. The DIP switches determine which direction the board is listening for resets. Section A.12 portrays the DIP switch settings for Rev 2 Hardware.

Test Network Messages

The test network is a debugging framework that allows test messages to be transmitted from a Raw tile during program execution.

The configuration FPGA handles the task of receiving test network messages, decoding them, and handling them. The test network was added to the Raw chip design using a one-bit serial protocol to minimize the number of pins required.

In the current (as of 4/25/06) version of the firmware, the configuration FPGA does not decode the test network messages. There exists test firmware that forwards the test network signal to the USB FPGA on the I/O board; however, this implementation only forwards the test messages from one Raw chip. A scalable version of a test network message routing network has been discussed, but the implementation remains a project for the future.

In the meantime, there are jumper header pins that contain the test network output. The test network output appears on pin 6 of J2, J20, J5, and J23. With the Quad Board in its default orientation (North = up), the top right pin of J2 and the bottom left pin of J20 carry the test network data for the top left and bottom left Raw chips respectively.

A.2.2 I/O Board FPGAs

The I/O Board contains 5 FPGAs: 1 XC2V250 and 4 XC2V3000 FPGAs. The board also contains a SystemACE chip that allows a user to load FPGA code from a CompactFlash card into each FPGA on power-up. The order in which the chips appear in the JTAG chain is: systemACE, config, PCI0, EXP0, EXP1, and PCI1. Geographically, if the I/O board is oriented in its default orientation with its connectors to the Quad Board on the right, the FPGAs are ordered PCI1, EXP1, EXP0, and PCI0 from top to bottom. Figure A-1 shows the ordering of the XC2V3000 FPGAs.

config FPGA

The configuration FPGA is the small XC2V250 FPGA responsible for clock and reset distribution, as well as inter-FPGA control communication. Its code is very similar to that of the config FPGA on the Quad Board.

PCI0, PCI1 FPGAs

The PCI FPGAs are XC2V3000 FPGAs responsible for I/O with the host computer. The PCI FPGAs have connections to the PCI, USB, and memory busses, as well as a 20-bit wide bus to each of its two neighboring XC2V3000 FPGAs. At this point, "PCI" is a bit of a misnomer because the FPGAs no longer implement PCI bus control. The PCI FPGAs' current primary purpose is to provide USB support.

EXP0, EXP1 FPGAs

The EXPansion FPGAs are connected to an expansion connector header. Like the PCI FPGAs, the EXP FPGAs also have a 20-bit wide bus to each neighboring XC2V3000 FPGA. The EXP FPGAs' current primary purpose is to communicate with SDRAM.

Communication with Raw

Figure A-1 shows the connections between the FPGAs on the I/O board with the Raw processors on the Quad Board. The numbers next to the FPGAs represent the Port numbers as labeled in the schematics. Each box on the right column represents a Raw tile on the East boundary of the Quad Board (the East side of the Quad Board lies on the left side of the board because of the way the Raw chip was designed. The label in the box is the number of the tile in the Fabric system (as opposed to the number on the chip). Finally, the numbers next to the Raw tiles are the numbers of the port in the Raw Fabric system. Please note that these port numbers do not match up with the Quad Board schematics, but rather with the port numbers of the simulated 64-tile Raw processor, as described in the Quad Board User Guide.

For example, Port 1 on the PCI0 FPGA communicates through Port 14 of the Quad Board to tile number 55.

A.3 Programming the Quad Board

JTAG programming of the configuration FPGA is achieved by connecting the Xilinx Parallel Cable IV or USB Cable to J7 on the Quad Board. Please note that the TDO is connected to TDO and TDI is connected to TDI. Also note that Parallel Cable III may be incompatible due to voltage level differences.

There are two chips on the JTAG chain: the actual XC2V250 configuration FPGA and a XC18V04 PROM that may be used to program the configuration FPGA on power-up.

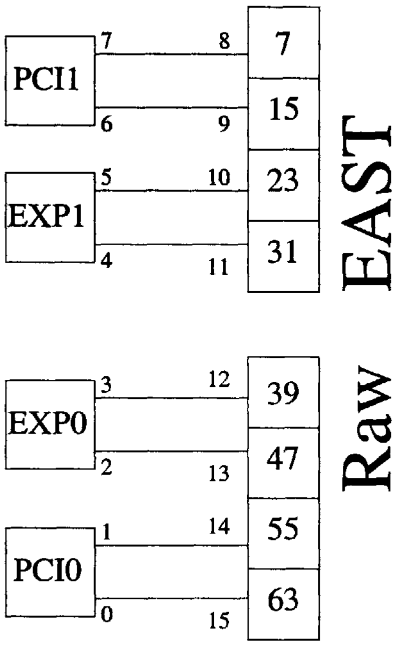


Figure A-1: I/O and Quad Board Connections

The XC2V250 configuration has been programmed with the most current firmware. There is a design bug in revision 1 of the hardware that prevents the PROM from auto-configuring the FPGA on power-up. The XC2V250 may be programmed by the PROM by depressing SW3. This bug should be fixed in revision 2 of the Quad Board.

Please note that the No Connects on the PROM may not be tied together. This information could have saved us a lot of debugging time.

A.4 Programming the I/O Board

The I/O board FPGAs are programmed through a JTAG chain connecting the five configurable FPGAs as well as a SystemACE FPGA. Each link on the JTAG chain is connected only to its predecessor and successor, and the final FPGA connects back to the beginning of the chain, forming a ring. Please see the schematics for more details on the signal path.

We currently use three different methods to program the JTAG chain: Parallel Cable, USB Cable, or CompactFlash.

A.4.1 Parallel Cable IV

JTAG programming of the configuration FPGA is achieved by connecting the Xilinx Parallel Cable IV to J1 on the I/O Board. Please note that the TDO is connected to TDO and TDI is connected to TDI. Also note that Parallel Cable III may be incompatible due to voltage level differences.

The programming is performed using the Xilinx IMPACT software included when installing the Xilinx ISE. First, put the software in Configuration Mode. Then, right click the window and select Initialize Chain... Assign .bit files to each FPGA to be programmed, then right click and hit Program... to program the FPGA.

Multiple FPGAs may also be programmed in one cycle. First, highlight multiple multiple FPGAs using ctrl+click or shift+click. Next, press the toolbar button for programming FPGAs (it is the fifth from the right).

A.4.2 USB Cable

The USB Cable is used very similarly to the Parallel Cable, only there are a few software drivers required before it may be used. These drivers may be found by obtaining the latest version of the Xilinx ISE software.

A.4.3 CompactFlash SystemACE

The CompactFlash SystemACE programming method involves writing data onto a CompactFlash card and inserting it into the slot on the I/O Board. Technically, the Xilinx literature claims it is Hot-Swappable (don't need to power off the system before inserting), but it's probably safer to turn off the I/O board before inserting the card.

Programming is achieved by using File Mode on the Xilinx IMPACT software. There is no "Initialize Chain..." command like there is in Configuration Mode, so the user needs to know the exact order of the JTAG chain to put the correct code in the correct FPGA. The order of the chain is: Config, PC0, EXP0, EXP1, PCI1.

The SystemACE system can handle up to 8 different configurations stored on the same card. The configuration to boot is selected on the I/O board by flipping DIP switches on SW2. On SW2, the first 3 switches select the configuration location to use (0-7), and the last switch enables whether or not the SystemACE system is enabled at all.

Once in a while, the SystemACE will fail to boot for no obvious reason. In these cases, instead of flashing the STAT LED and programming, the ERR LED lights steadily (when no card is present, the ERR LED blinks on and off). In these cases, the CompactFlash card needs to be reformatted. This CANNOT be done on a Windows XP computer, or the card will remain unusable. Formatting with a Windows 2000 computer has been verified to work; other methods such as using the mkdosfs program may work as well, but have not been explored.

A.5 Debug Module

Since the I/O board has very few debug pins, we are using a memory debugging module to access memory bus pins. These pins are hooked up to Agilent logic analyzer headers for easy hookup to the analyzer. In addition, the analyzer contains configuration files in the Raw_Group folder to work with the pin assignment made in the existing ucf files.

A copy of the pin assignments may be found in /raw-boards/io_fabric/doc/ucf_files/mem_debug.ucf. Simply use these pin assignments rather than the memory bus assignments.

Also, please note that as documented in the mem_debug.ucf file, there are many pins that do not match up to the assignments in the file. At this point, we are uncertain whether the mislabelling is due to the memory debug module or the schematic designation.

A.6 USB

The original USB controller firmware was written by Levente Jakab. He has since left MIT.

The USB firmware interfaces with the USB Plugin Card. The card is currently in revision 2.1. The blue colored cards are revision 2+; the changes between 2.0 and 2.1 are cosmetic.

The USB Plugin Card contains the Cypress EZ-USB SX2 chip that handles communication in the USB 2.0 protocol. The firmware is designed to interface with this chip. Using either version 2.0 or 2.1 is fine.

We have soldered on one test header to one of our USB Plugin Cards. It allows us to connect to an Agilent logic analyzer. Please note that one of the pins is unconnected, so it always appears as a "0" on the logic analyzer. It is NOT shorted to ground, so it does not pose a direct hazard.

The first indication that the Plugin Card is working correctly is to look at the LEDs on the card itself. Under normal operation, there should be two LEDs lighted: the board power and the USB bus power (top 2 LEDs). The Tx and Rx LEDs should NOT be lit (bottom 2 LEDs). If they are lit, the USB pins on the FPGA are most likely unconnected (for example, as in the termination FPGAs). Additionally, please note that the USB card draws its power from the board; the USB bus provides power only to light up the LED on the plugin board.

Figure A-2 shows the assignment of the USB module LEDs in PCI0 and PCI1.

On power-up, the LEDs in PCI0 make a nice symmetric pattern, which serves as a quick check that the SX2 was configured correctly (see Figure A-3.

After the USB module has been reset by way of Reset Packet from a host machine (e.g., after trying – successfully or not – to boot code into the Fabric by USB), the LEDs on PCI0 will display the memorable pattern seen in Figure A-4.

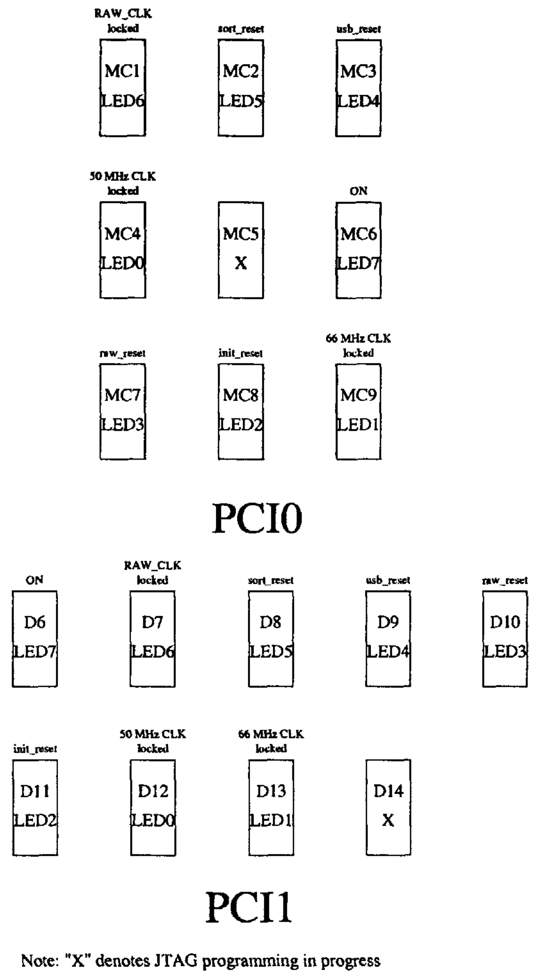


Figure A-2: USB LEDs on PCI0 and PCI1

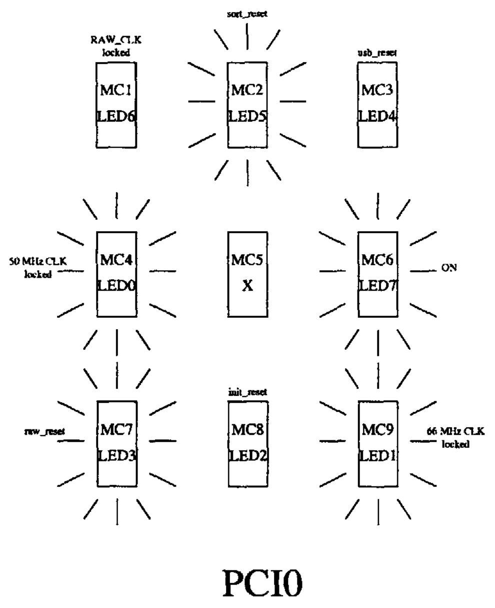


Figure A-3: USB Initialized LED Pattern

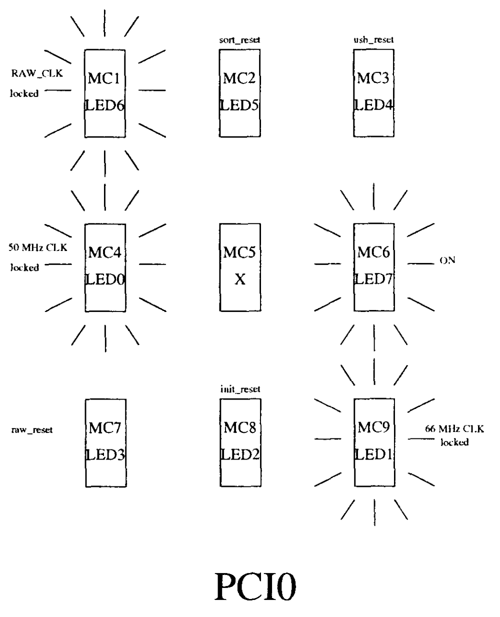


Figure A-4: USB Booting LED Pattern

A.7 Memory Controller

The memory controller was written by Nathan Schnidman.

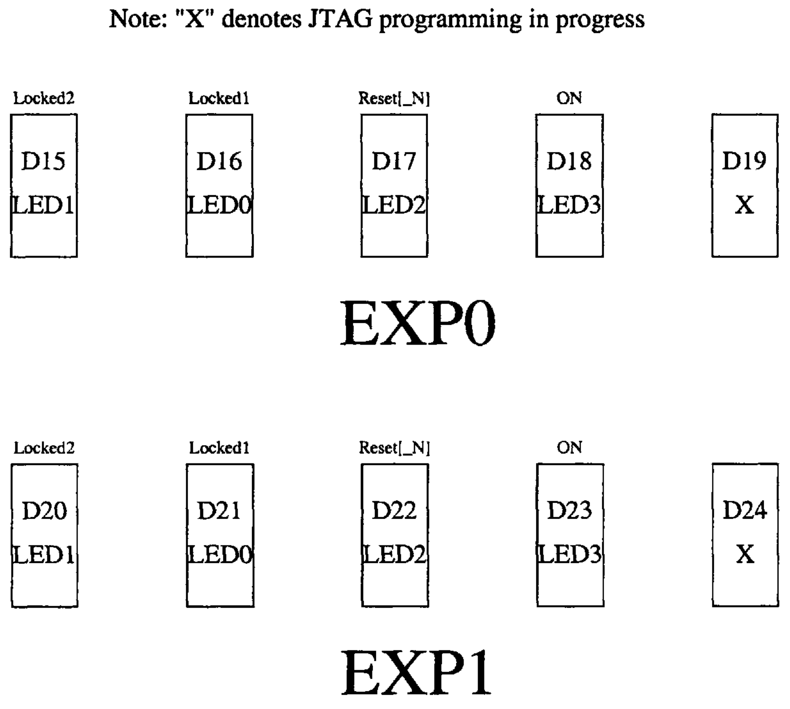


Figure A-5: USB LEDs on EXP0 and EXP1

Using the memory controller is straightforward. Simply program the memory controller firmware into the relevant FPGA. When the memory controller is functioning correctly, four LEDs should be lit. Note that this occurs only after reset. Figure A-5 show what the four LEDs represent.

The one issue that we wrestled with was making the refresh and initialization times correct for the Fabric system, since we are running the Fabric at a measly 33 MHz. This means that the memory should be refreshed more often, in terms of clock cycles. The file to edit is the sdrmc.v file in the sdrmc folder. There are existing comments detailing what the values must be set to. Note that the firmware needs to be changed every time the global clock is changed.

One major quirk of the firmware is the way that the define.v source is handled. Other firmware files refer to this definitions file for global constants. However, since we compile the code in a separate directory (in the syn directory), the tools have a difficult time finding the file. The unideal way we have gotten around this problem is to import the define.v file through a relative path from the synthesis folder. The upshot is that every time the name of the sources folder is adjusted, all of the files depending on define.v need to adjust the path of the import.

A.8 Rev 1 Buglist

This section tracks the various hardware bugs discovered in revision 1 of the Raw Fabric hardware. These busgs should have all been resolved in the second revision.

A.8.1 South Side bus disconnect

Description: There is currently a flaw in the routing for the south side of the Quad Board that prevents its functionality. Pins 39 and 40 on P19 are not connected to data pins from Raw.
Fix: This error will be corrected in the next hardware revision.

A.8.2 Reset pull-ups

Description: The pull-up resistors on the reset line in parallel result in a resulting equivalent resistance that is too low, preventing it from functioning correctly. The current fix is to use the data lines connecting the config FPGAs between the I/O and Quad baords instead.
Fix: In the future, a hardware change will be required to fix the erroneous behavior (probably simply removing the pull-up resistor, R??).

A.8.3 Control line rotation

Description: The current connector pin assignments prevent the I/O board from being able to be connected on the West and South sides. In these configurations, the Raw clocks are connected to the FPGA clocks and vice-versa. These are incompatible

since the Raw clocks are HSTL_I whereas the FPGA clocks are LVTTL.
Fix: The connector pins have been reassigned for the next revision.

A.8.4 Criss-crossed yummies

Description: To fit all the pins necessary on the Raw chip, the north and south have the middle 2 ports muxed together (ports 1&2 in the north, and ports 9&10 in the south). Currently, the yummies are misrouted: port 1 is connected to port 9 and port 2 is connected to port 10. The correct routing is port 1 to port 10 and port 2 to port 9.
Fix: The signals will be re-routed in the next revision.

A.8.5 Bad N-S connection between tiles 31 & 39

Description: Please see A-10 to find tiles 31 and 39. The link between these two Raw tiles is occasionally flaky. The connection from port 11 is ostensibly related to this bug. We have seen bad behavior from the following nets:

`P_A8_C3_4` and `P_A8_C3_18`

on port 11. These nets are usually stuck low.
Fix: The correct pressure must be placed on the Raw chip using the heatsink screws. Further exploration of this bug may be warranted.

A.8.6 Cannot boot from ports 8 and 12

Description: The data coming from the I/O board when connected to the east side of the board is corrupted. The nets in question are:

```
WEST_PI7_13, WEST_PI7_14, WEST_PI7_19, WEST_PI25,
WEST_PI3_13, WEST_PI3_24, WEST_PI3_25.
```

These nets are usually stuck high.
Fix: The problem has been pinpointed on the I/O board side. Pressure must be applied on the corner of the FPGAs to allow for correct data transfer.

A.8.7 Incorrect yummies from I/O board

Description: We have found that the yummies on the majority of the ports going to Raw are stuck high.

Fix: Uncomment the yummy lines in the .ucf file.

A.8.8 Memory controllers on port 13 and 14 produce erroneous data

Description: When EXP0 and PCI0 are loaded with memory controllers, the data stored/received from port 13 and 14 is corrupted. Port 13, pin 15 (net: EXP0M0DQ47) is usually stuck high when it should be low, but in a few rare instances it is stuck low when it should be high. On port 14, a large number of pins are usually stuck high.

Fix: Apply pressure to the correct corner of the FPGA to temporarily alleviate the problem. Figure A-6 shows the correct corners to apply pressure to.

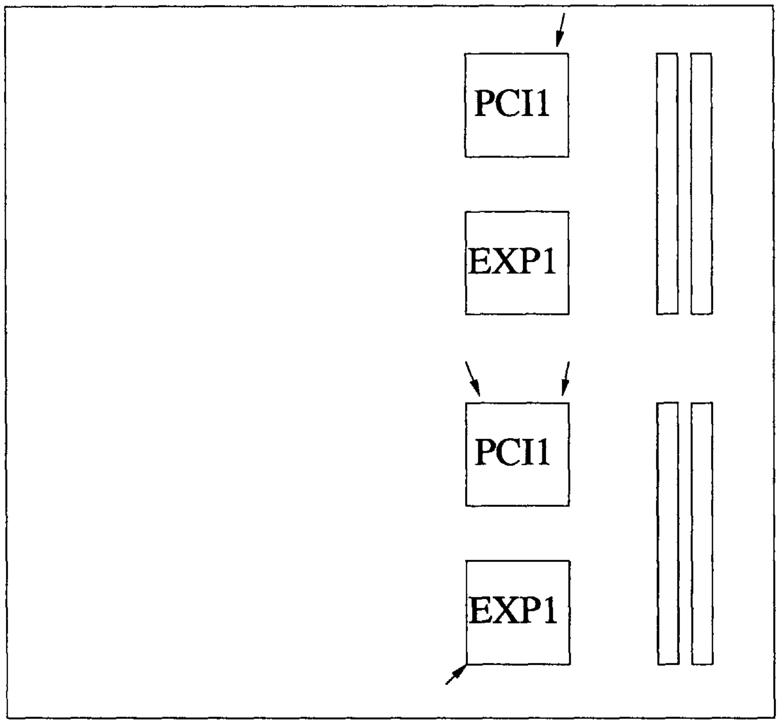


Figure A-6: I/O board FPGA corner problematic areas

A.8.9 Cannot use memory controller on port 11

Description: The memory controller test stalls when running tests on port 11. In running the External Port tests, the port also fails on a striping one test and the mdn yummies test.
Fix: The current theory is that the problem is on the Quad Board side. The problem is related to the Quad Board tile 31-39 connection problem (Please see the Quad Board Hardware User Guide).

A.8.10 Cannot boot from ports 8 and 12

Description: A few of the nets are stuck high, suggesting a lapse in connection. The nets in question are:

```
WEST_PI7_13, WEST_PI7_14, WEST_PI7_19, WEST_PI25,
WEST_PI3_13, WEST_PI3_24, WEST_PI3_25.
```

Fix: Apply pressure to the bottom right corner of the relevant FPGA (PCI1 or EXP0). We are going to investigate to see if the same problem exists in the other I/O Board. **Edit:** This problem does not exist in the other I/O board.

A.8.11 Memory Controller reporting garbage

Description: In our read-write tests, the memory controller occasionally returns values with pins stuck high
Fix: This was a problem that exists only on the first I/O board. The second I/O board was fairly consistent in the memory tests.

A.9 Project File System Organization

The I/O board project files exist in CVS in the /rawboards/io_fabric/ folder. The Quad board project files exist in CVS in the /rawboards/quad_fabric/ folder.

A.9.1 Revision 2 Hardware Firmware

The latest revision 2 firmware is found in the syn_rev2 and src_rev2 folderes. All firmware in these folders in compiled using veribuild_fabric.

Veribuild_fabric is the successor of the veribuild system created by David Wentzlaff for compiling firmware.

Advantages of veribuild:

1. Modular design
2. Batch jobs - automation
3. use multiple cagfarms for parallel compilation
4. Integration with CVS

Changes in veribuild_fabric over veribuild:

1. Pure Linux (no more wine!)
2. Newer Xilinx PAR tools
3. Better synthesizer: Synplify's Synplicity replaces Synopsys's FC2

Some of the changes in veribuild_fabric are incompatible with firmware that used to compile with veribuild. Please check the documentation folder in veribuild_fabric for ideas on how to resolve these differences.

How To Set Up Veribuild_Fabric

1. CVS
2. Code
3. Makefile
4. Compile

CVS cvs -d /projects/raw/cvsroot co veribuild_fabric

Code

1. Verilog code in src_rev2
2. makefiles and .ucf constraint files in syn_rev2

The reason for separation is that in the Fabric system, any given component may have many versions of essentially the same firmware. The differences are essentially just the pin assignments. The underlying code stays the same and goes in the src folder but the different pin outs are sorted out in syn. The separation of folders allows more a more organized system.

Makefiles The easiest way to create a Makefile is to simply copy an example and modify its contents.

A few of the important sections are:

1. Point to veribuild_fabric folder
2. Add device flags
3. Add sources (verilog, vpp - verilog pre-processor)
4. Specify command-line flags
5. Specify Makefile commands (rules)

Compile Compile by typing "make" in the folder containing the makefile.

Setting up Veribuild [DEPRECATED]

(Please note: the following is not relevent in the current version of the tools, but is included here just in case someone wants to try to resurrect veribuild in the future.)

For some reason, Veribuild does not work right out of the box after checking it out from the CVS. Here are the steps we went through to get it working:

1. Check out Veribuild from CVS.

2. Copy .wine from someone else, mess with CHMOD to get proper permissions (for some reason, the .wine files must not be readable by anyone other than the owner).

3. Change license file location in veribuild/makefiles/Makefile.license. We changed it to: export LM_LICENSE_FILE = 27000@mtlcad.mit.edu

In each project using Veribuild, change the VERIBUILD_TOPDIR variable in the Makefile.

A.10 Boot Process

The preferred method of booting is through USB. The Block RAM method is included here for reference.

1. Block RAM on FPGAs

2. USB

A.10.1 Block RAM Booting

The first method of booting implemented was simply to stream in data from a ROM in the FPGA. These projects are named "simple_rom" in the firmware archives. For reference, the shortest possible boot sequence is 5 words long:

```
0x00000080
0x701D0337
0x00ECFFFF
0x00000000
0x00000080
```

This boot sequence is sent over the static network to boot the tile connected to the port the code is sent through.

The default simple_rom code we use is a little bit longer because we wanted to make sure the data gets through the FIFOs (this was probably not necessary, but history is history).

A.10.2 USB Booting

The I/O board receives booting code from a host computer and sends it over to the quad board. The correct firmware must be loaded into the PCI0 or PCI1 FPGA.

Currently, we boot straight into the code of the program. Thus, a program compiled in starsearch will have the boot and program code in one file. The starsearch make infrastructure will generate a *.rbf-?? file, where ?? is the port number to which it is meant to stream the boot code.

We use a non-standard boot pattern in order to boot from the bottom right corner of the quad board. In order to boot to the Fabric, make sure to add these lines to your Makefile:

```
TILE_PATTERN=8x8
RGCCFLAGS += -O2
MULTI_SNAKEBOOT=0
SNAKEBOOT_CONFIG_FILE = xxx/starbuild/common/intravenous_configs/
boot/east_bottom_standard.bc
```

xxx represents the folder in which the starbuild has been checked out.

The resulting binary should be a .rbf-15 file.

In order to boot from USB, first log in to the rawhost connected to the Fabric. Next, navigate the folder containing the usb code (rawboards/io_fabric/usb_boot/programs/) in our case. If necessary, compile the usb_boot_and_read C program. Then, execute usb_boot_and_read rbf-file.rbf-15, where rbf-file is the name of the rbf file to boot.

Upon booting, the code will first let you know how many packets (words) are in tthe boot file, then it will scroll through the boot code as it sends it. At the end, our current standard boot file will says "running test..." at the end of booting. At this point, the USB is waiting for any messages (Static, MDN, or GDN) to arrive back out of either port 14 or 15.

Please note that when they arrive, the port numbers are incorrect, since they match with the handheld system. Thus, if it says port 12, it actually means port 15, and when it says port 13, it actually means port 14. This can be trivially changed.

A.11 File Locations

The CVS Repository system is used for version control. Please check out rawboards/quad_fabric/ for Quad Board firmware and documentation and rawboards/io_fabric/ for I/O board firmware and documentation.

The following folders are in active use. All firmware is compiled using veribuild_fabric at this point.

`quad_fabric/syn_rev2/`

Contains the firmware for the XC2V250 configuration FPGA on the Quad Board.

`quad_fabric/doc/`

Contains documentation for the Quad Board, including schematics, layout, and some useful data specification sheets.

`io_fabric/doc/`

Contains documentation for the I/O Board, including schematics, layout, and this guide.

`io_fabric/src_rev2/`

Contains the Verilog code for firmware projects.

`io_fabric/syn_rev2/`

Contains the .ucf files (timing constraints + pin assignments) and Makefiles for firmware projects.

The following folders are deprecated. Most of the firmware was compiled using veribuild (not veribuild_fabric) and therefore is not compilable anymore (unless we find some Synopsys DC2 licenses).

`quad_fabric/quad_config/`

Contains zip file of Xilinx ISE projects for the Quad config firmware.

`io_fabric/config/`

Contains zip file of Xilinx ISE projects for the I/O config firmware.

`io_fabric/bit_files`

This folder used to serve as a bitfile repository to dump bitfiles so that they were in a common directory. There was a command at the relevant Makefile rule that would basically copy the end product bitfile into this directory. The firmware in this folder is Fabric rev1 firmware.

`io_fabric/components, io_fabric/fpgas`

The oldest firmware folders for the Fabric. The components folder contains the source code, while the fpgas contains the .ucf files and Makefiles.

`io_fabric/src, io_fabric/syn`

The firmware folders for the Fabric used before src_rev2 and syn_rev2. These folders were replaced by src_rev2 and syn_rev2 rather than keeping the same folders because the rev2 firmware will be compiled exclusively using veribuild_fabric rather than veribuild (again, because of the expiration of licenses). I wanted to keep these firmware files intact for people too lazy to run through cvs back-versions.

`io_fabric/usb_boot`

Some simple usb interfacing code that streams rbf files into Raw.

A.12 DIP switch settings

A.12.1 Quad Board DIP switch settings

Figure A-7 and Tables A.1, A.2 describe the dipswitch settings on the Quad Board. Note that the Quad board is never the reset originator, but may be the clock originator. The Quad board receives its clock and reset signals from the same direction, dictated by the SEL DIP settings in Table A.1. The Quad board will treat this direction as an input, and forward the signal to the 3 other directions. Note that if the clock originator setting is set, then the Quad board will forward the clock to all 4 directions.

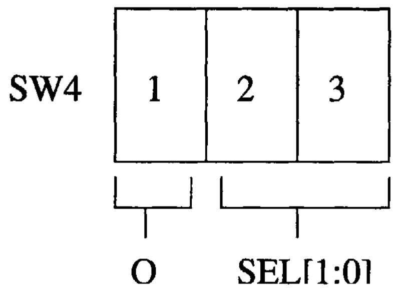


Figure A-7: Quad Board DIP Switch Positions

O	Clock Originator
0	No
1	Yes

Table A.1: Quad Board DIP Switch Clock Originator input select

SEL[1:0]	Clock and Reset Input Setting
00	North
01	East
10	South
11	West

Table A.2: Quad Board DIP Switch input select

A.12.2 I/O Board DIP switch settings

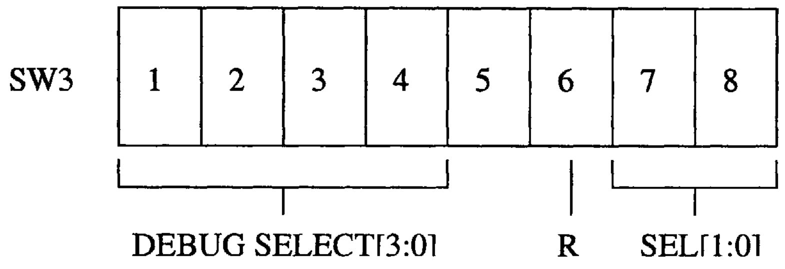


Figure A-8: I/O Board DIP Switch Positions

R Value	Reset Setting
0	Reset from Off Board
1	Reset from Local Switch

Table A.3: I/O Board DIP Switch Reset signal input select

SEL[1:0]	Clock Input Setting
00	33 Mhz Osc Clock
01	External Clock from J3
10	Offboard clock synchronized to DCM
11	Offboard clock synchronized to DCM

Table A.4: I/O Board DIP Switch clock input select

Dipswitch	Output Signal
0000	Reset after reset selection
0001	undefined
0010	Input reset from Off board
0011	SEL[0]
0100	SEL[1]
0101	R
0110	Local_Reset: Reset Button on board
0111	Clock: after the clock selection logic (BUFGMUX)
1000	Off board clock into the DCM
1001	33 MHz OSC Clock to the DCM
1010	External Clock from J3
1011	undefined
1100	undefined
1101	undefined
1110	undefined
1111	undefined

Table A.5: I/O Board DIP switch DEBUG signal settings

A.13 Port and Tile Numbering

Figure A-9 shows the numbering of the tiles on a single Raw chip, if we were to look down onto the chip on the board. Please pay careful attention to the fact that physically, East and West are flipped from their expected positions when we're looking down. They are drawn this way on the diagram to aid in debugging the hardware.

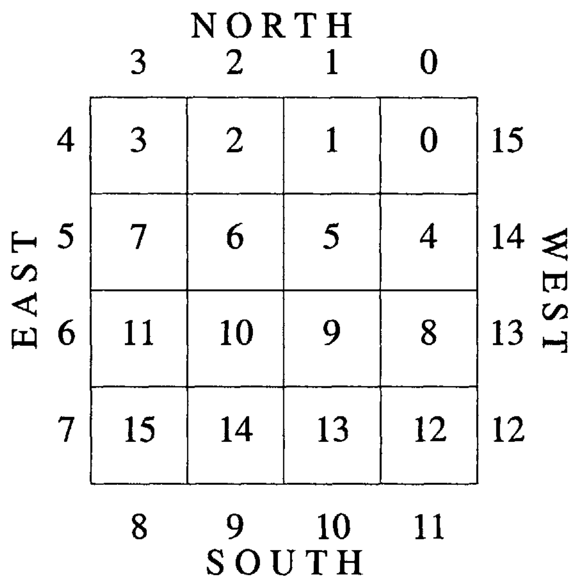


Figure A-9: Single Chip Port and Tile Numbering

Figure A-10 shows the numbering of the tiles on a single Quad Board. Please note that the port numbers are different from those of a single Raw chip. It is therefore important that when referring to a port number it is specified whether the number is in reference to the entire Quad Board, or in the frame of reference of a single chip. By default, port references should refer to the number in the entire Quad Board case.

Figures A-11 and A-12 show non-standard configurations for booting through the south-east port (port 11). These were used for debugging purpose before we could get the entire 64-tile board booted.

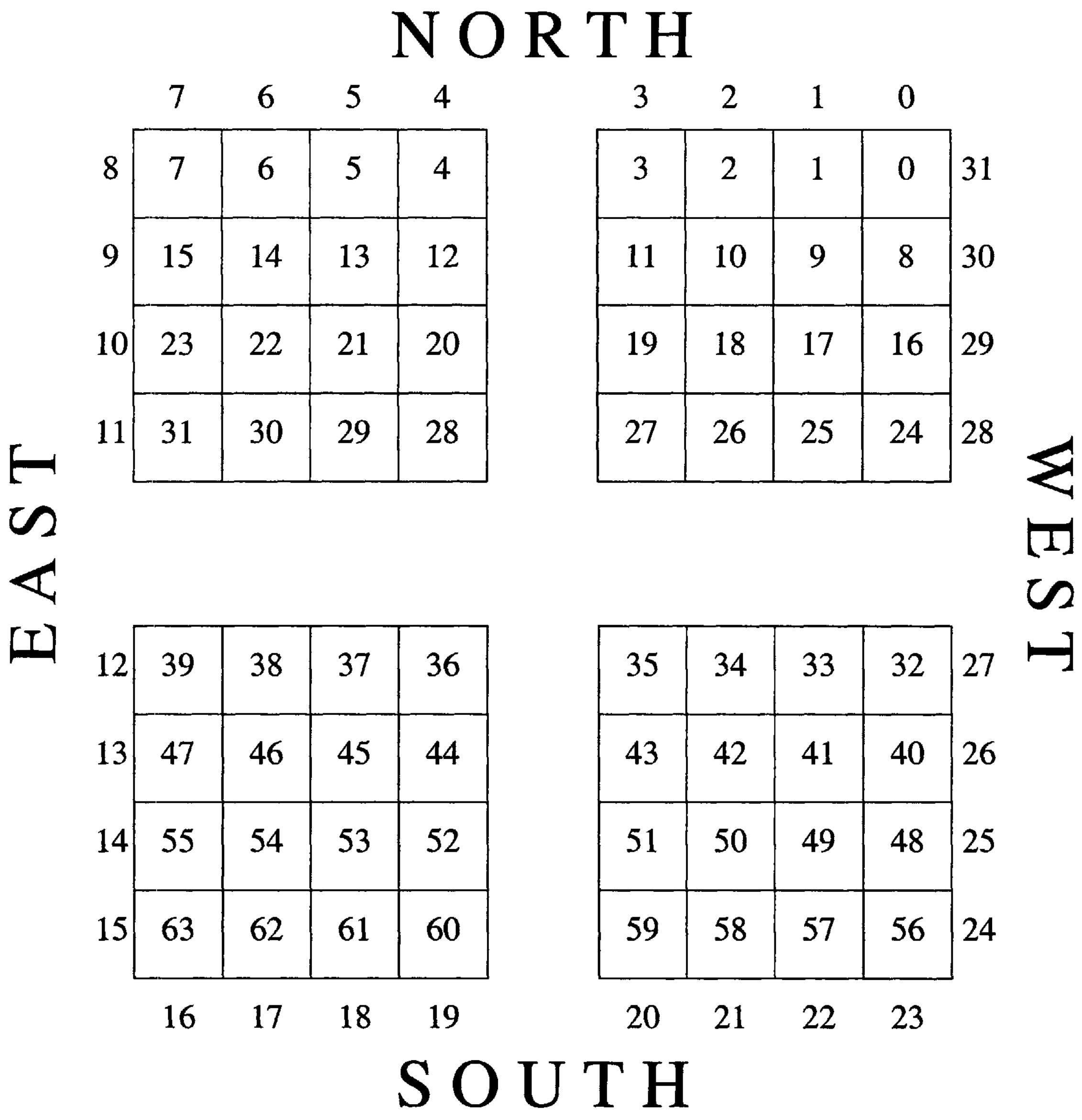


Figure A-10: Quad Board Port and Tile Numbering

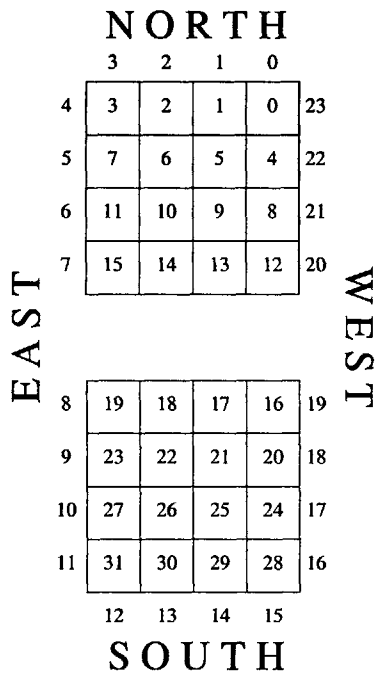


Figure A-11: Alternate configuration: 4x8 Fabric

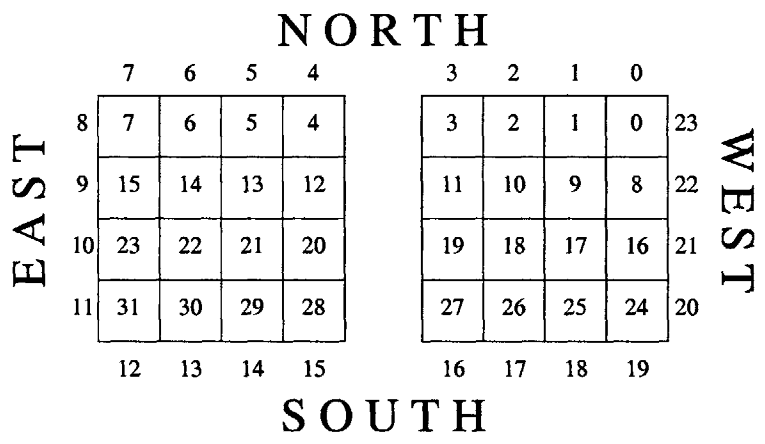


Figure A-12: Alternate configuration: 8x4 Fabric

A.14 History

5/3/06	- I/O Board sent back to BEMA for blown DC-DC converter
4/26/06	- Quad Board sent back to BEMA for misplacement of rigidizer
4/16/06	- Assembly of 1 I/O board and 1 Quad board (rev 2) complete
1/24/06	- Schematic and Layout for Revision 2 finalized
10/6/05	- Host interface I/O calls working on Fabric
9/20/05	- Finished testing of I/O Board #3 - connectors all work
9/19/05	- Received I/O Board #3 from ISI
8/21/05	- Compiled and Ran C Programs
8/2/05	- External Port Tests passed for North side
8/1/05	- Hacked version of port 10, port 11 East memory controller
7/25/05	- Speed Gasket 6 for muxed ports completed
7/11/05	- Discovered I/O board #2 pin contact problem on memory side
7/11/05	- Finished extensive testing of memory controller
6/23/05	- Received second I/O board
6/10/05	- Resolved port 8 & 12 booting problem
6/10/05	- Discovered FPGA pin contact problem in PCI1 & EXP0 FPGAs
6/9/05	- Extensive Memory controller testing begun
6/8/05	- Identified and characterized criss-crossed yummy issue
6/7/05	- Discovered Raw heat sink issue
6/7/05	- New retention brackets received
6/6/05	- External Port Tests created for debugging board connections
5/30/05	- Extensive low-level hardware debugging begun
5/28/05	- Completed 6-port muxed speed gasket for North & South
5/18/05	- Memory Controller port complete
5/10/05	- Booted all 64 tiles on Raw Fabric
5/7/05	- Remote booting infrastructure completed
5/5/05	- USB boot entire chip through PCI0
5/2/05	- USB booting through port 14 acheived

4/25/05	- USB Device recognized by host PC
4/25/05	- I/O Board User Guide documentation started
4/10/05	- Booting 1 tile
3/14/05	- Quad Board User Guide documentation started
3/13/05	- Clock and Reset distribution between boards working
3/8/05	- Clock Distribution working
3/7/05	- Fixed Quad Board to be able to program FPGA using JTAG
3/3/05	- Test bench set up
3/2/05	- Received 1 I/O Board and 1 Quad Board from ISI

Bibliography

[1] A. Agarwal, S. Amarasinghe, R. Barua, M. Frank, W. Lee, V. Sarkar, D. Srikrishna, and M. Taylor. The raw compiler project. In *Proceedings of the Second SUIF Compiler Workshop*, August 1997.

[2] R. Barua, W. Lee, S. Amarasinghe, and A. Agarwal. Maps: A compiler-managed memory system for raw machines,.

[3] C. E. Kozyrakis and D. Patterson. A new direction for computer architecture research. *IEEE Computer*, September 1997.

[4] R. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper, and K. Asanovic. The vector-thread architecture. In *Proceedings of the 31st International Symposium on Computer Architecture*, 2004.

[5] W. Lee, R. Barua, M. Frank, D. Srikrishna, J. Babb, V. Sarka, and S. Amaraasinghe. Space-time scheduling of instruction-level parallelism on a raw machine. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1998.

[6] W. Lee, D. Puppin, S. Swenson, and S. Amarasinghe. Convergent scheduling. In *Proceedings of the 35th International Symposium on Microarchitecture*, November 2002.

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

[8] R. Nagarajan, K. Sankaralingam, D. Burger, and S. W. Keckler. A design space evaluation of grid processor architectures. In *Proceedings of the International Symposium on Microarchitecture*, 2001.

[9] R.M. Rabbah, I. Bratt, K. Asanovic, and A. Agarwal. Versatile tiled-processor architectures: The raw approach. In *Proceedings of the Eight Annual High Performance Embedded Computing Workshop*, June 2004.

[10] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffmann, P. Johnson, J. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Schnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. The raw microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE Micro*, 2002.

[11] Michael Taylor.

[12] E. Waigold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal. Baring it all to software: Raw machines. *IEEE Computer*, September 1997.