The Implementation of a Reliable Router Chip

by

Kin Hong Kan

S.B., Computer Science and Engineering (1993)
S.B., Electrical Engineering (1993)
Massachusetts Institute of Technology

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

May 1994

Author .....
Department of
Electrical Engineering and Computer Science
May 7, 1994

Certified by .....
William J. Dally
Associate Professor
Thesis Supervisor

Accepted by .....
F.R. Morgenthaler
Chairman, Department Committee on Graduate Theses

The Implementation of a Reliable Router Chip

by

Kin Hong Kan

Submitted to the Department of
Electrical Engineering and Computer Science
on May 16, 1994, in partial fulfillment of the
requirements for the degree of
Master of Engineering in
Electrical Engineering and Computer Science

Abstract

This thesis describes the implementation of a router chip for a network of processors. The router features the simultaneous bidirection pad drivers and a diagnostic port with a JTAG interface. The design process results in some useful experience for the development of a complex VLSI system in an academic research environment. The router chip is designed for a 3.3v 0.8μm 3 metal layers CMOS process, and is targeted to run at 100MHz.

Thesis Supervisor: William J. Dally
Title: Associate Professor

Acknowledgment

I would like to thank my thesis supervisor, Professor William Dally, for his guidance and support throughout the project. I would also like to thank Larry Dennison, the architect of the Reliable Router, from whom I learned the essence of good engineering work. My thanks also go to other members of the Reliable Router Team: Thucydides Xanthopoulos and David Harris.

I would like to thank the funding agency for their financial support. [1]

Thanks go to Dicky, Stephen, and David for making MIT a fun place. Eric and Sarah Au, for their prayers and encouragement. Elaine, for her love and all the good time we have together.

I am deeply indebted to my parents. Though they are so far away physically, their phone calls are always a source of my comfort.

Praise to the Lord - the Creator and the Provider. Thank Him that I never lacked anything because of His grace.

"I will praise thee, O LORD, with my whole heart;
I will shew forth all thy marvelous works."

Psalms 9:1, KJV.

[1]The research described in this thesis is supported by the Defense Advanced Research Projects Agency under contracts N00014-88K-0738, F19628-92-C-0045, and N00014-91-J-1698.

Contents

List of Figures

Chapter 1

Introduction

This thesis describes the implementation of the Reliable Router as a VLSI system. The thesis project involves digital system and MOS circuit design, VLSI layout design, and the verification of the design using various CAD tools. The thesis summarizes the work being done as part of the team effort of the Reliable Router team. [1]

1.1 Overview

This chapter discusses previous research that leads to the development of the Reliable Router, and the major features of the VLSI system. Chapter 2 describes the different functional blocks of the Reliable Router. Chapter 3 details the simultaneous bidirectional pad drivers employed by the router. Chapter 4 describes the IEEE JTAG interface and the diagnostic port of the router. Chapter 5 discusses the design methodology for implementing the router as a VLSI system, and some experience derived from the design process. Chapter 6 provides a conclusion of the thesis and suggestions for future work.

[1] William J. Dally, Larry R. Dennison, Thucydides Xanthopoulos, Kin Hong Kan and David Harris.

1.2 Motivation

In a message-passing parallel computer, different processors communicate by sending point to point messages. These messages allow the processors to dispatch concurrent tasks and to synchronize with one another. The performance of the network substrate through which the messages are passed thus affect the performance of the parallel computer significantly.

The effects of network performance becomes more obvious as we explore the regime of fine-grain parallelism[2]. A large message latency imposed by a network means that we need to amortize the messaging overhead by aggregating many fine-grain tasks into one large-grain task, which implies less concurrency.

The Reliable Router project stems from the need for an off-the-shelf high performance router to facilitate parallel computing. The design of the Reliable Router chip, either as a packaged product, or as a piece of core logic to be incorporated into other VLSI design, aims to provide the communication substrate needed for the next generation parallel computer.

Another goal of the Reliable Router project is to serve as an implementation testbed for empirical routing research. In particular, the design incorporates the Unique Token Protocol to provide a reliable routing service, virtual channels and adaptive routing to improve the network performance, and also the simultaneous bidirectional pad drivers to increase the per-wire bandwidth. The experience drawn from implementing the VLSI system would help us evaluate the above-mentioned studies in a real system.

1.3 Previous Research

The idea of partitioning a physical communication link into logical channels, namely virtual channels, was shown to increase the physical bandwidth utilization of a messaging network [3].

An adaptive routing algorithm, which determines the path of a packet dynamically

according to network traffic, also provides an opportunity for increased performance[4]. Because incremental routing decisions are made locally when a packet is routed, an adaptive routing algorithm can easily allow fault tolerance by routing around a faulty node.

The "Turn Model" [8] is particularly useful for analyzing adaptive routing algorithms. The model determines the turns that can be made by a packet in a network, so as to avoid deadlock.

A deadlock-free adaptive routing algorithm was developed for the Reliable Router[12] based on the above-mentioned studies. The Reliable Router also supports link-level retransmission with the Unique Token Protocol [6], which guarantees the delivery of exactly one copy of every packet. The protocol does not require a source node to store old packets for retransmission; therefore no local storage is needed, making the network truly scalable.

Simultaneous bidirectional signalling [7] [9] increases inter-chip bandwidth by allowing point-to-point digital signalling over the same package pin at the same time. The technology reduces the packaging cost for a network router by reducing the number of pins required.

1.4 Design Features

The major design features of the Reliable Router are that:

- each VLSI chip represents a node in a 2-D mesh network, which supports efficient point-to-point messaging;
- it implements the Unique Token Protocol and tolerates any single faulty node on the network;
- it is a packet router which employs worm-hole routing;
- it implements a deadlock free adaptive routing algorithm;
- it supports 5 virtual channels;

- it implements the simultaneous bidirectional pad drivers for inter-chip communication;
- it provides an IEEE JTAG interface for setup and diagnostic purpose;
- it has a low latency of 8 clock cycles for each network hop;
- it is targeted to run at 100MHz, which amounts to a maximum throughput of 400 MBytes per network data link in each direction;
- it is designed for a 3.3v 0.8μm 3 metal layers CMOS process.

Chapter 2

Architecture

This chapter summarizes the architecture of the Reliable Router. It describes the main functional blocks of the VLSI system, namely the Input Controller, the Output Controller, the Arbiter and the Processor Interface.

2.1 Functional Blocks

Figure 2-1 shows the block diagram for the Reliable Router. The Reliable Router has 6 input and output ports, which are all connected to a 6 x 6 crossbar. Out of the six pairs of I/O ports, four are the network interface ports, one is the processor interface port, and the remaining one is the diagnostic interface port. An arbiter

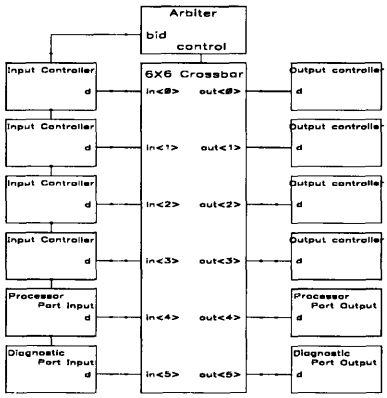

Figure 2-1: Reliable Router Block Diagram

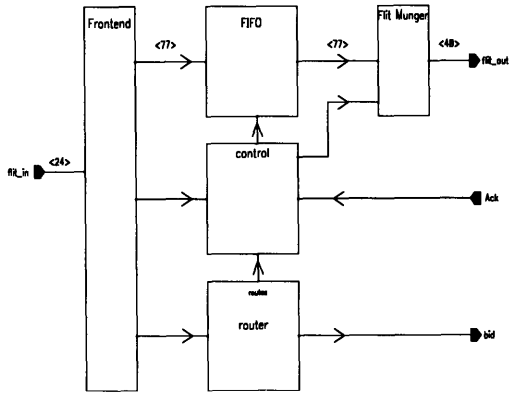

Figure 2-2: Input Controller Block Diagram

determines which of the input ports wins the bid to an output port, and controls the corresponding crossbar connection.

2.2 Input Controller

Figure 2-2 shows the block diagram for the Input Controller. The Input Controller module synchronizes an incoming flit, computes a route for the flit, and makes a bid to the arbiter. This module contains most of the control logic of the Reliable Router.

The Front End module is responsible for retiming an incoming flit to the local clock domain. The Reliable Router operates at 100MHz, and it is difficult to distribute a global clock signal to the entire network with a low skew. As a result, the Reliable Router implements a queueless plesiochronous data recovery scheme[5]. The scheme allows adjacent router nodes to operates on their own clock signals, each of which has approximately the same frequency.

The Front End module samples the 24 bit input data path on both edges of the 100MHz clock signal. The module then checks its parity, extracts the control information, and puts a flit onto a 77 bit data bus on every rising edge of a 50MHz clock signal.

The FIFO module stores an incoming data flit before it is routed to another node. The FIFO implements the storage for 5 virtual channels. Each channel queue can hold 16 77-bit data flits.

A data flit coming off the data queue is passed onto the Flit Munger module, which appends to it other control information. The module retimes the 50MHz, 77 bit FIFO data bus by doubling its bit rate, thus reducing the required crossbar bus width to 40 bits.

The Router module contains the logic for computing a route for an incoming flit. The piece of combinational circuit is replicated 5 times, so that each virtual channel has a dedicated router of its own. The router implements a dead-lock free adaptive routing algorithm[12].

The Control module records most of the state information of the VLSI system. In particular, it keeps track of the routing problems to be solved, the computed routes for each virtual channel, and the number of vacant slots in the flit queues of the adjacent routers. The Control module also picks, in a round-robin fashion, a virtual channel to bid for a crossbar connection. It then informs the FIFO module the identifier of the channel queue to read from, upon receiving an ACK signal from the Arbiter module.

2.3 Arbiter

The Arbiter module determines which of the 6 input controllers wins a bid to a particular output controller. The arbitration priority is again determined in a round-robin fashion using a set of shift-registers. This guarantees that any input controller can acquire at least one-sixth of the bandwidth of any output controller. The Arbiter module returns the arbitration result to the 6 input controllers using the ACK signal bus; it also sets up the crossbar connections according to the arbitration result.

In addition, the Arbiter module records the BUSY status of the 30 virtual channels[1]. This information is broadcasted to the Router modules of the Input Controllers, which require it to compute the routes.

[1] 5 for each of the 6 Output Controllers.

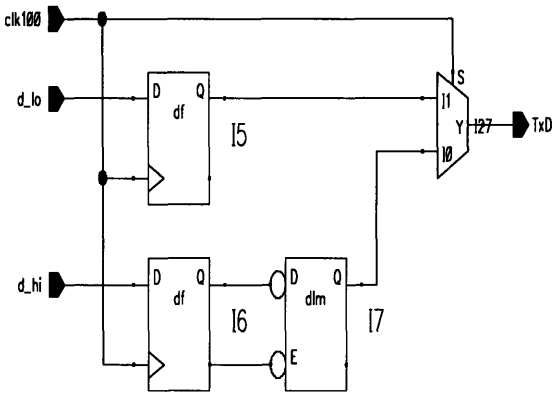

Figure 2-3: Output Controller Data Retiming Logic

2.4 Output Controller

The Output Controller module retimes the 40 bit data path from the crossbar by doubling its bit rate to 200MBits per second. The module also computes a parity and appends it to the flit together with other flow-control information. The bus width of the Output Controller is thus 24 bits[2], which is also the width of a link between two adjacent routers. Figure 2-3 shows the data retiming logic for a single bit of the data link. Its function is to maximize the interchip bandwidth for the router network.

2.5 Processor Port

The Processor Port provides the network interface to a Reliable Router node. It has a 40 bit input data bus and an output bus of the same width. The frequency at which the network processor communicates with the Reliable Router could be the quotient resulting from dividing the 100MHz router clock by an integer, i.e. 100MHz, 66MHz, 50MHz, 33MHz and 25MHz.

The Processor Port is similar to an Input/Output Controller pair in most aspects; the difference lies mainly in the flow-control logic. Flow-control is achieved using explicit CTS signals to and from the processor, instead of the state information stored

[2]Includes 20 bits of data plus other control information.

at the Input Controller.

2.6 Diagnostic Port

The Diagnostic Port provides a means to send or receive data flits through a serial port. Its design would be described in Chapter 4.

Chapter 3

Bidirectional Signalling

The 6 input and output ports shown in Figure 2-1 require a large amount of interchip bandwidth. To allow the use of a conventional chip carrier with fewer than 300 pins, we employ simultaneous bidirectional signalling [7]. This method allows bidirectional point to point digital signal communication on the same chip carrier pin and at the same time.

The design of the driver presented here is similar to a previous design[9]. The major difference is that this design is powered at 3.3V, instead of 5V. This chapter describes the performance of the driver as shown by simulation result, using an extracted netlist from the driver's layout.

3.1 System Overview

Figure 3-1 shows the block diagram of a pair of simultaneous bidirectional transceivers. The printed circuit board connection is modelled as a transmission line, with both ends of the line terminated with an internal resistor. A logic 1 or a logic 0 is transmitted as a positive current or a negative current respectively; the signal across the chip boundary is thus a superposition of the two current streams. To receive the digital signal from another transmitter, a local transmitter generates an internal copy of its own transmitted signal as a reference. The reference voltage is then subtracted from the superimposed signal to generate the received signal.

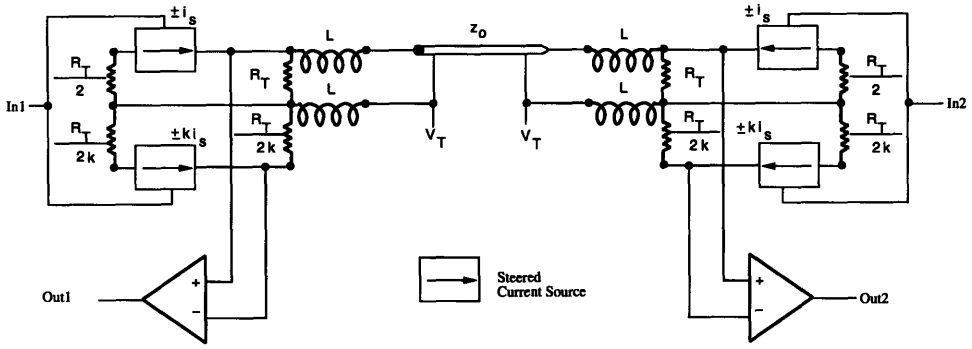

Figure 3-1: Simultaneous Bidirectional Signalling

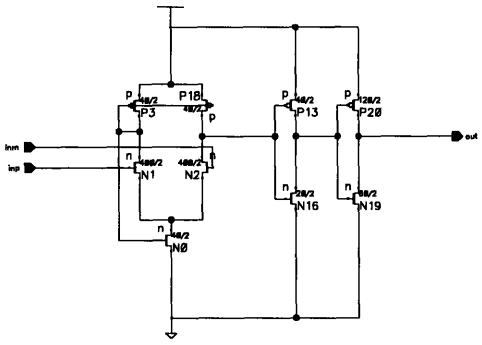

Figure 3-2: Receiver Schematic

3.2 Receiver Circuit

The design of the receiver circuit is a Chappell-style differential amplifier [1]. Figure 3-2 shows the circuit diagram.

Figure 3-3 and figure 3-4 show waveforms of HSPICE simulation on an extracted netlist from the receiver layout.

Figure 3-3 shows the difference in response for a pair of receivers. The output waveform for an isolated pulse is compared to the corresponding rail-to-rail transition. In the diagram, the skew for the waveforms is less than 0.2ns for both the up-going and the down-going pulse, and at least 65% of the input pulse width is preserved. A simple latch is thus sufficient for sampling these pulses correctly at the data rate shown, i.e. 800MBits Per Second.

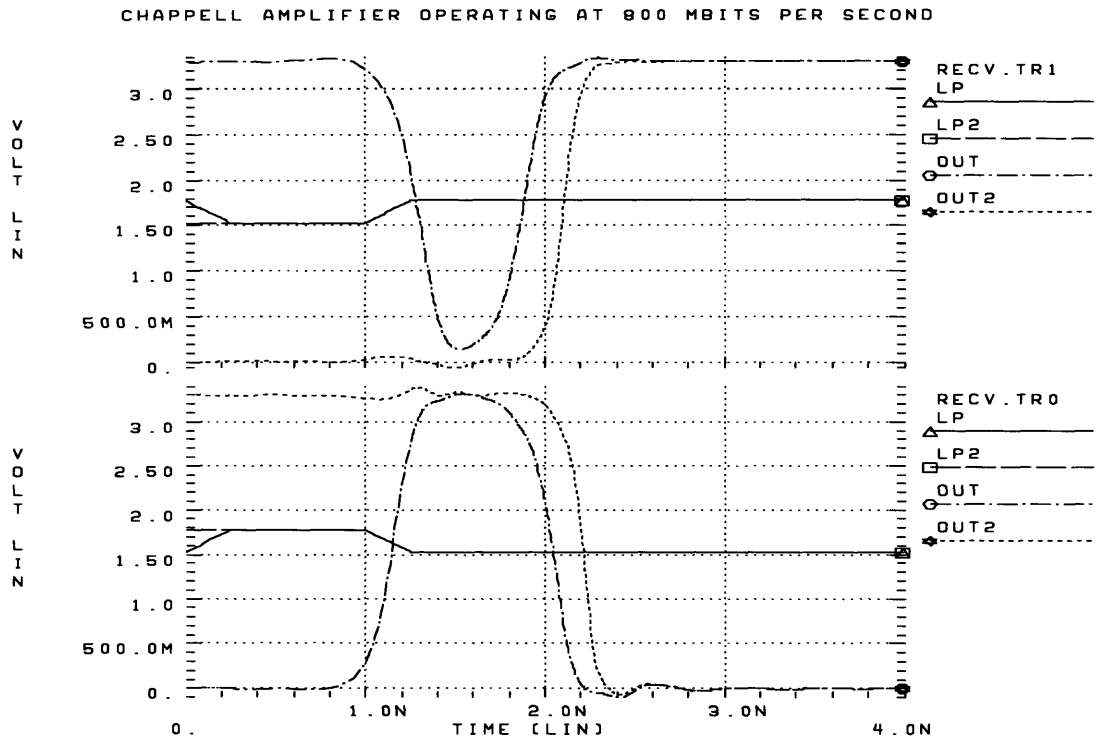

Figure 3-3: Receiver Operating at 800MBits Per Second

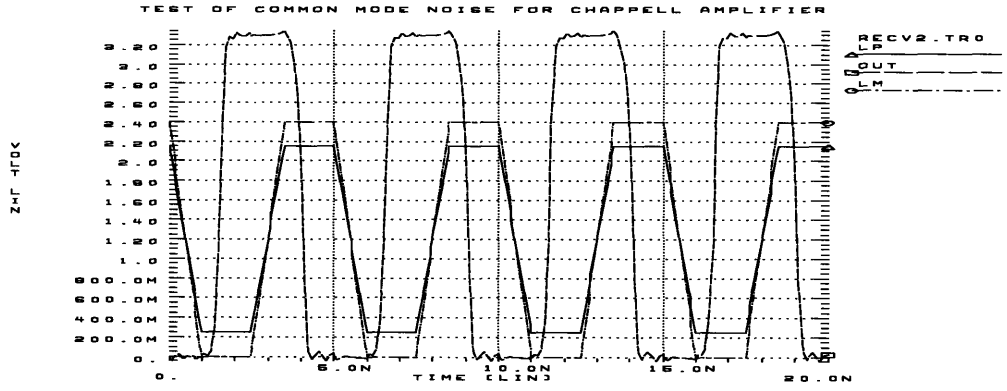

Figure 3-4: Test of Common Mode Noise Rejection for Chappell Amplifier

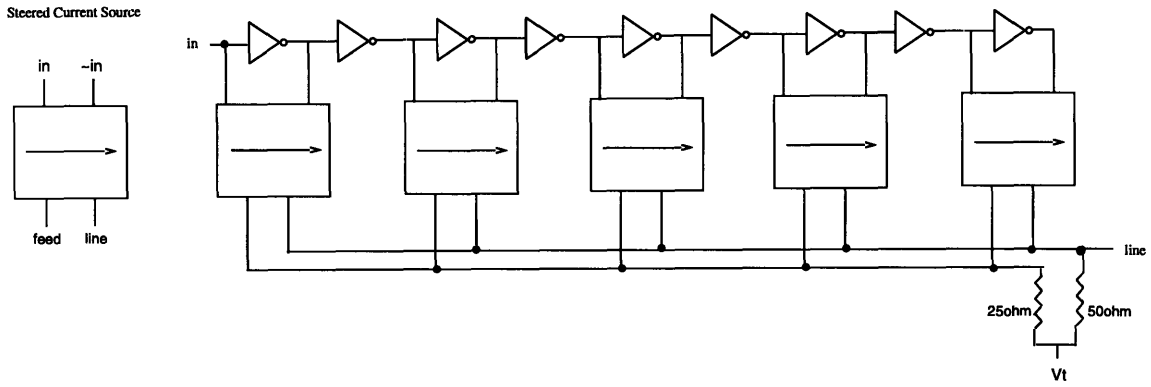

Figure 3-5: Staggered Transmitter Stages

Figure 3-4 shows the common mode noise rejection performance of the receiver. The figure indicates that for an input signal of 125mV above or below the reference voltage V_T, a valid range of V_T is 0.13V to 2.28V. This also means that the driver can tolerate an AC common mode signal with a frequency of 200MHz, up to a magnitude of 0.6V.

3.3 Transmitter Circuit

The transmitter sources or drains 5mA of current into the package inter-connect, according to the digital input value. As shown in figure 3-5, the turn-on of the transmitter is staggered into 5 stages, so as to reduce the $L\frac{di}{dt}$ noise induced by the package pins.

Figure 3-6 shows the schematic for a single stage of the transmitter, together with the biasing circuit. The transmitter has a steered current source design, which keeps the current drawn by the drivers to be roughly constant, so as to reduce the noise on the power planes.

The transmitter current sources are biased deep into saturation to reduce the noise that could be induced from the power supply. Figure 3-7 shows the power supply noise rejection performance for the transmitter. The noise induced from the power supply into the signal line is as low as -20dB even at 5GHz.

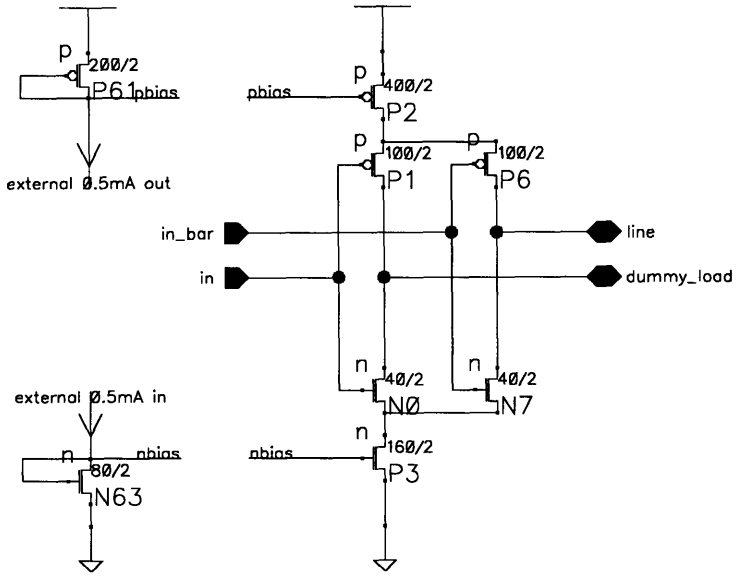

Figure 3-6: Steered Current Source and Biasing Circuit Schematic

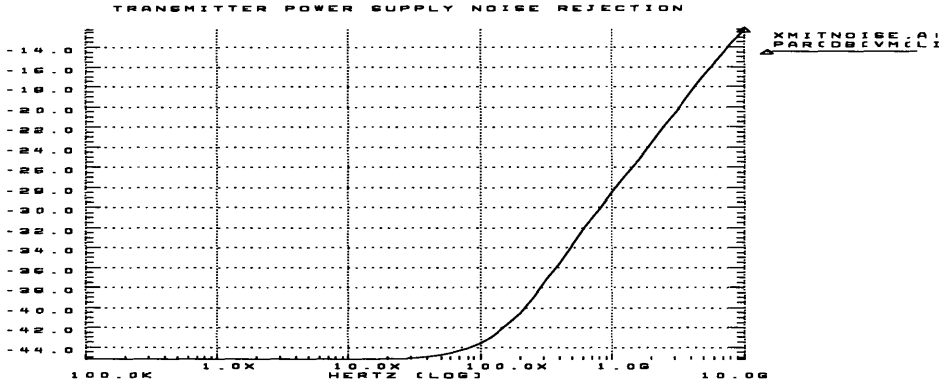

Figure 3-7: Power Supply Noise Rejection for the Transmitter

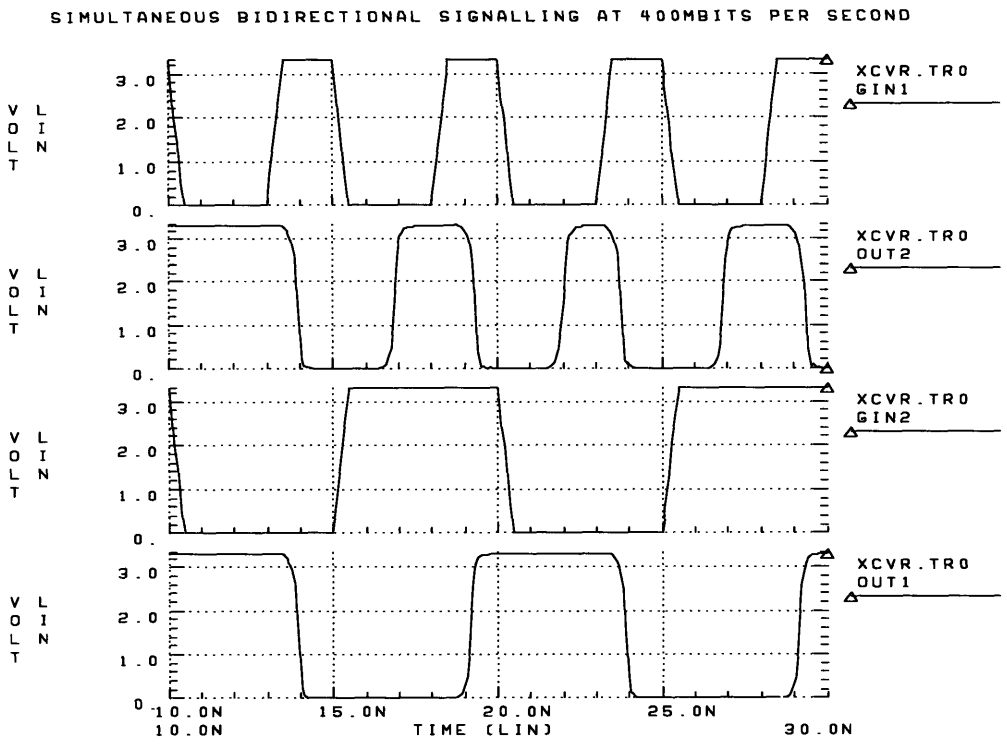

Figure 3-8: Inputs and Outputs of the Driver at 400MBits Per Second

3.4 System Performance

Figure 3-8 shows the inputs and outputs of a pair of simultaneous bidirectional drivers running at 400Bits per second. Gin1 shows the input signal at chip 1, and Gout2 shows the corresponding signal recovered at chip 2. Gout2 and Gin1 shows the signal travelling in the reverse direction. 3-9 shows the corresponding signals as seen at the inputs to the differential receivers.

When compared to a previous design[9], the design proposed here is at least as good with respect to the data rate (i.e. 400MBits per second). The new design is also superior in terms of the receiver response (i.e. 800MBits per second vs 660MBits per second), and transmitter power supply noise rejection (i.e. -28dB vs -20dB at 1GHz).

Figure 3-10 shows the layout for a bidirectional driver. The driver measures 1265x430 lambda, which is about 2.5 times the area of a normal I/O driver used for the Reliable Router.

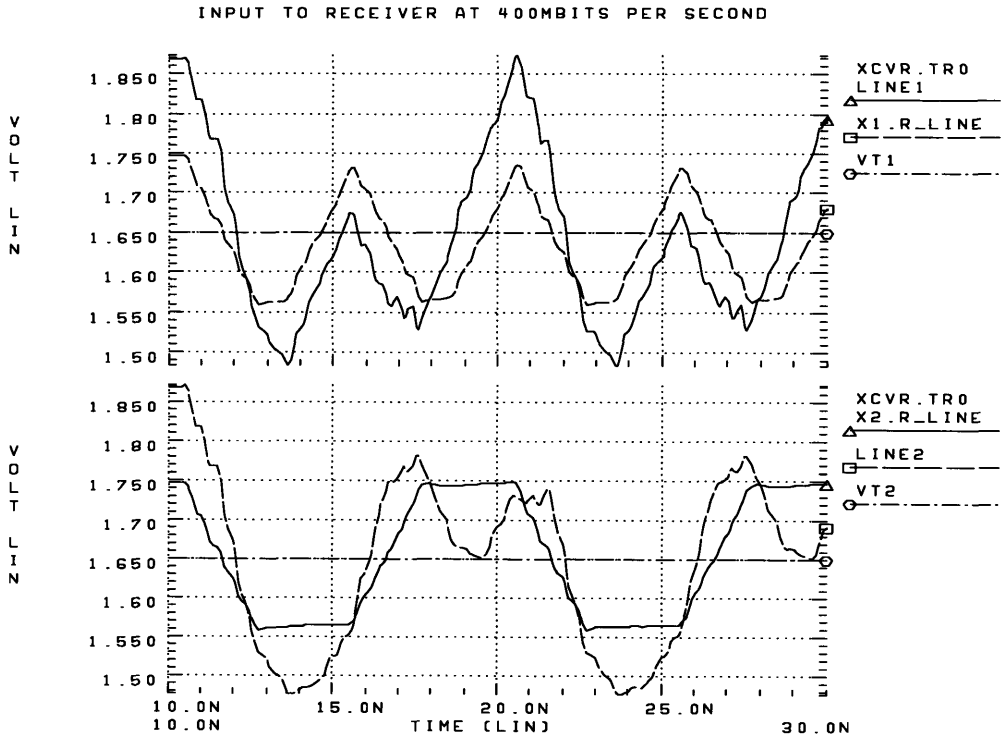

Figure 3-9: Inputs to the Receiver at 400MBits Per Second

Although the bidirectional drivers are bigger in area, the packaging cost for the Reliable Router is greatly reduced by using them. All of the 24 bidirectional signals that connects an adjacent pair of Reliable Router employs the simultaneous bidirectional signalling method, which amounts to a saving of 96 signal pins over the conventional rail-to-rail signalling method.

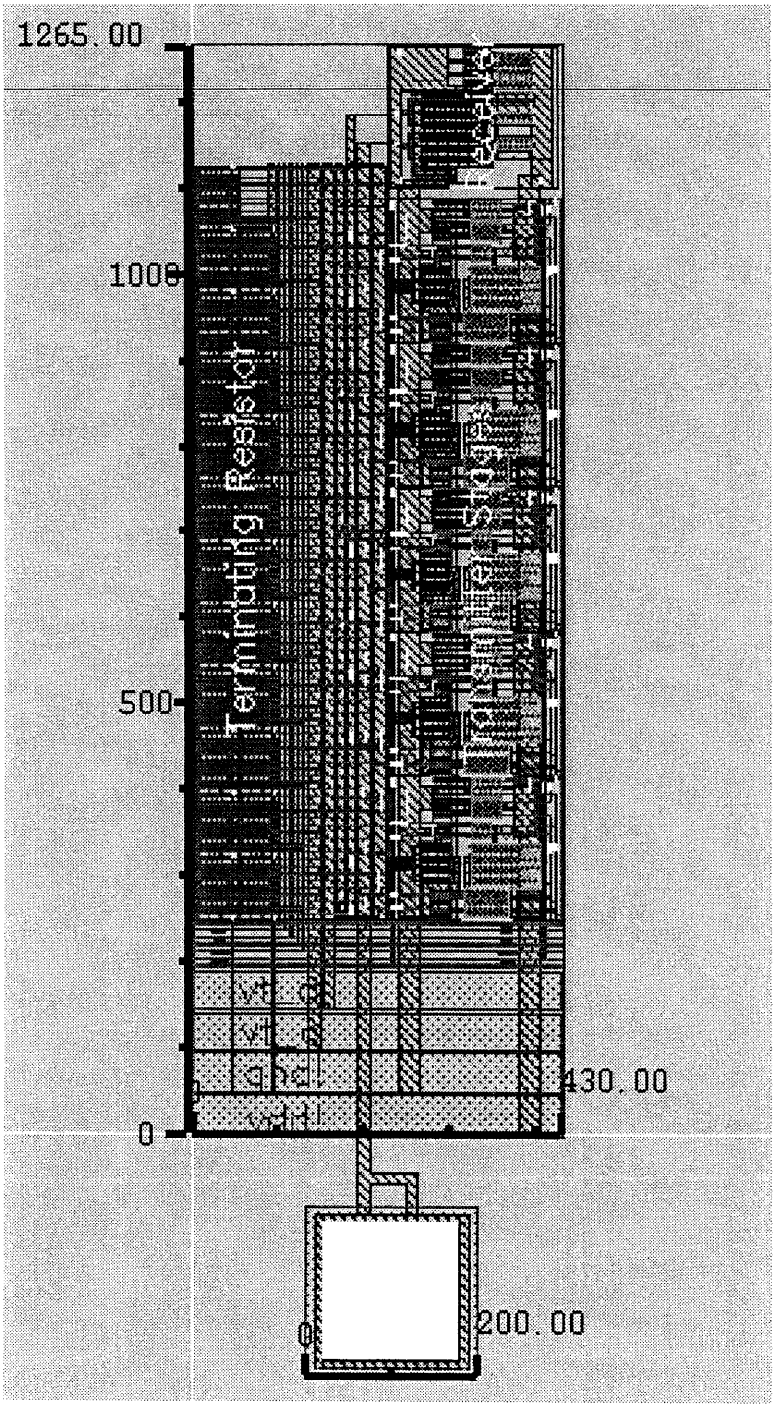

Figure 3-10: Pad Driver Layout

Chapter 4

Diagnostic Port

This chapter describes the architecture of the IEEE test standard employed by the Reliable Router. It also explains the design of the Diagnostic Input and Output ports.

4.1 Boundary Scan Architecture - JTAG

The IEEE Standard Test Access Port and Boundary-Scan Architecture 1149.1[11], more commonly known as JTAG, was defined in 1990 to provide a standard for testing assembled printed circuit boards. The standard allows the testing of components and circuit board interconnections through a serial port using a minimum number of control signals.

Figure 4-1 shows the block diagram for the JTAG architecture. The boundary

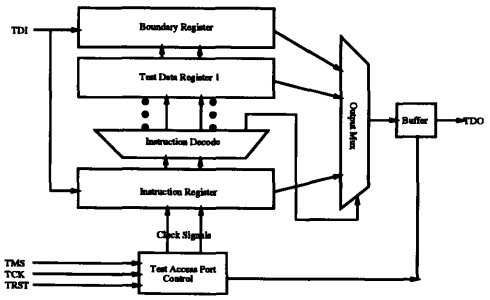

Figure 4-1: JTAG Architecture

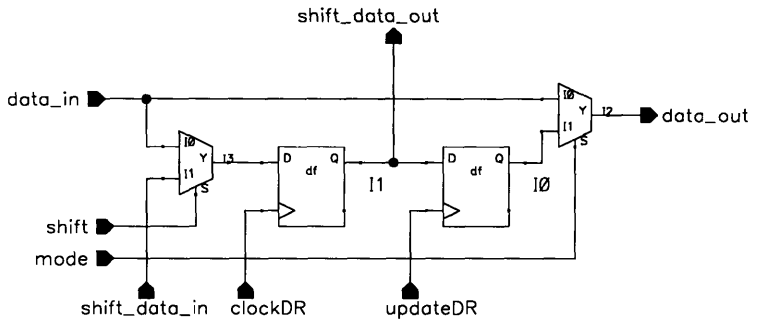

Figure 4-2: A JTAG Scan Cell

register, which provides access to all the external pins, and a set of test data registers, which provide access to selected internal registers, all share a common serial data input. The instruction register selects the data register being serially loaded/unloaded through the instruction decoder. The test access port control module generates the different clock signals needed for shifting the registers.

Figure 4-2 shows the schematic for a JTAG scan cell. A boundary register or a test data register is formed, depending on the register width, by a number of scan cells connected in series.

Data output of the scan cell can either be a value shifted in through the JTAG serial port, or simply be the input data. The scan cell is shifted when clockDR goes from low to high; the register is loaded when updateDR goes from low to high.

Figure 4-3 shows the state transition diagram for the Test Access Port (TAP) state machine. Transitions are controlled by the TMS signal and synchronized by the TCLK signal. The state transition sequence for loading the instruction register is very similar to that for a data register. Different clock signals for shifting the registers are generated by decoding the state bits of the TAP state machine.

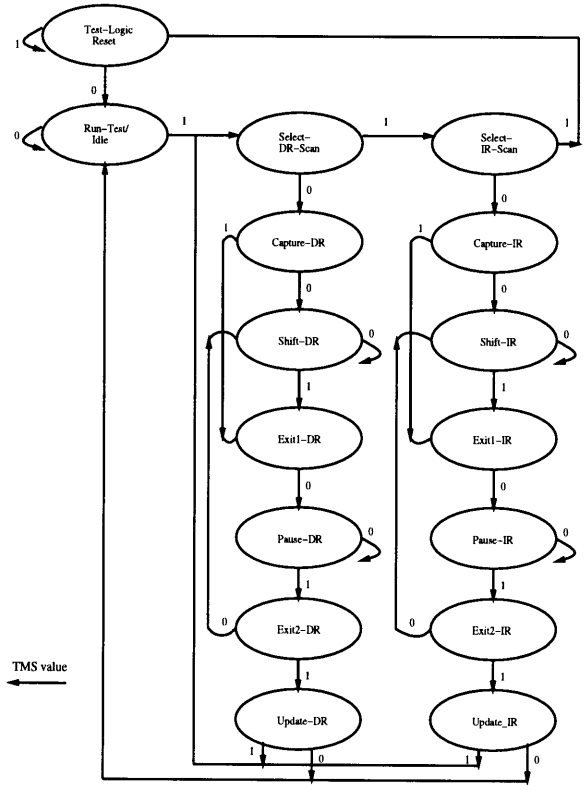

Figure 4-3: Test Access Port State Machine

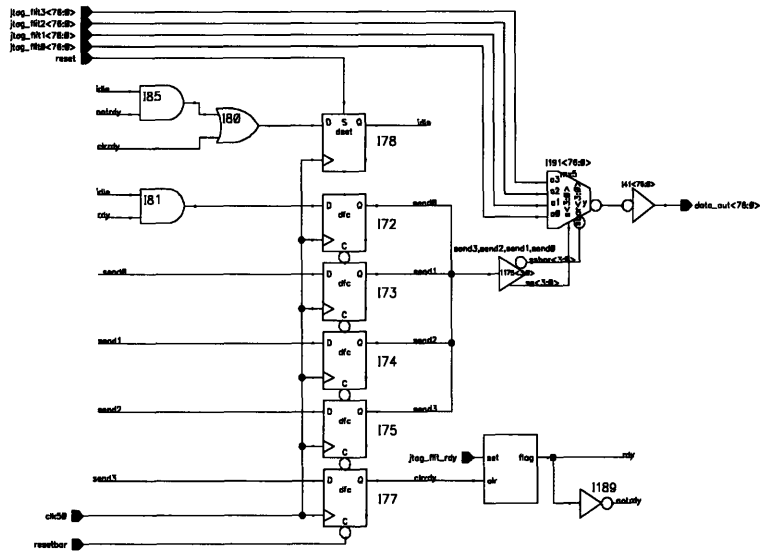

Figure 4-4: Diagnostic Input Port Schematic

4.2 Diagnostic Input Port

Figure 4-4 shows the schematic for the Diagnostic Input Port. The port takes the four flits of data shifted in from the JTAG interface and sends them to an Input Controller sequentially.

The state machine starts sending the flits when jtagFlitRdy, which is accessible externally through the JTAG interface, sets a flag signal. The state machine resets the flag signal after it has finished sending the flits.

The finite state machine of the Diagnostic Input Port employs a single register for each of its states. Such an encoding scheme simplifies the logic design and is particularly useful for a sequential state machine like the one shown.

4.3 Diagnostic Output Port

Figure 4-4 shows the schematic for the Diagnostic Output Port. The port can store four flits of data received from the network and shift them out serially through the JTAG interface.

The design of the output port is very similarly to that of the input port. It encodes

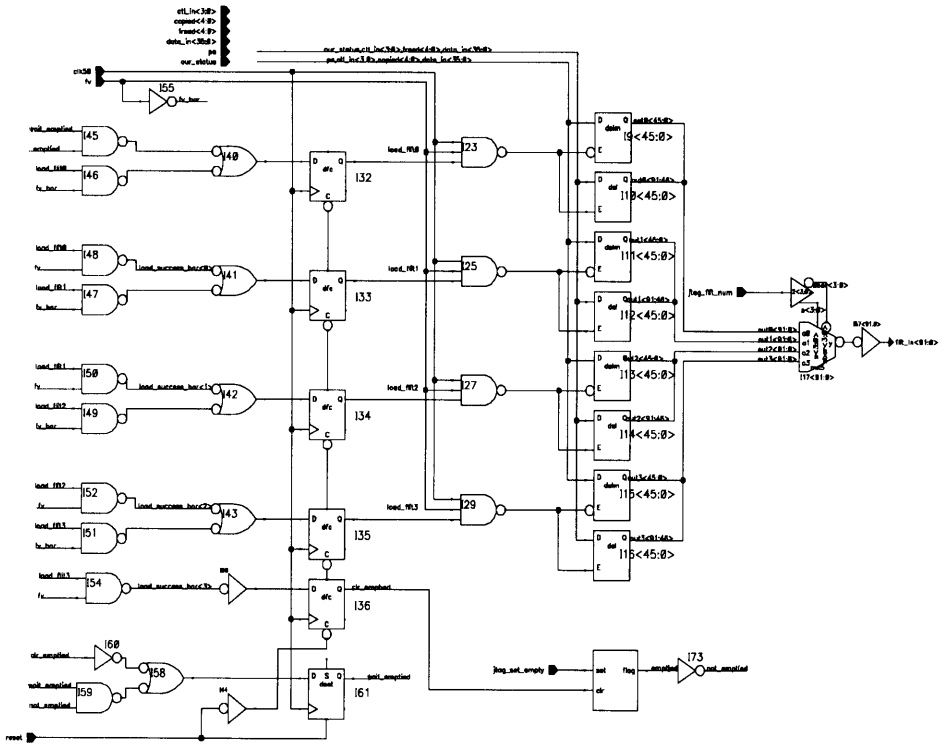

Figure 4-5: Diagnostic Output Port Schematic

each of its state bits in a different register, and it also makes use of a flag signal. The flag signal can be accessed externally through the JTAG interface, which notifies the user that four flits of data are received at the Diagnostic Port.

Chapter 5

The Design Experience

This chapter summarizes the design experience acquired from implementing the Reliable Router as a VLSI system. It describes the design process and some of the problems that the Router Team encountered. The chapter is targeted towards those who would design a chip in a similar research environment as the Router Team.

5.1 The Design Process

The design process of the Reliable Router involves iterations over different design choices. In general, our process is divided into 5 stages; though the boundary of the stages may be blurred, and a non-optimal design decision could possibly nullify the work of the previous stage(s). The different design stages are:

1. Algorithm: The algorithms that satisfy a design goal are first developed, studied and proved to be correct. For the Reliable Router, the algorithms are the Unique Token Protocol, and a dead-lock free adaptive fault-tolerant routing algorithm.

   The tools that we use at this level are some algorithmic techniques [1] for proving the correctness, and simulators written in C/C++ for studying the performance.

   There was a major redesign of the fault-tolerant routing algorithm when the schematics of the Reliable Router was close to be finished. The reason was a

[1] The Turn Model by Glass and Ni[8].

change in the design goal, which allows the router to tolerate a faulty node in the network, instead of just a faulty link. The algorithmic change was easily accommodated because of the good abstraction of the same class of problems by the Turn Model.

2. Architecture: The different modules of the VLSI system are defined at this stage. The modules are specified such that the low-level implementation details are ignored; each module is treated as a black box, with its input signals, behavior and output signals specified.

   The tools that we use at this stage are some behavioral Verilog models. The models are particularly helpful for understanding the interactions between the different modules.

   Several designs were considered before we settled with the current one. The major factor affecting the design choice is the amount of state information that flows between different modules. Because of the nature of the Router, most of the state information is needed by the module which makes the routing decision. We, therefore, minimized the information flow by putting most of the control and state logic in the module responsible for picking the route, i.e. the Input Controller.

3. Schematic: The different modules are specified using logic gates at this stage. A rough manual timing analysis, using unit gate delay, is performed to ensure that each module have a chance of running at the desired clock speed.

   The tools employed at this stage includes a schematic editor, which allows us to enter the schematic manually, and a schematic-to-Verilog netlister, which allows us to verify the logical functions of the different modules by simulation. Verilog test suites are written for testing each module, as well as for testing a 3x3 router network.

4. Layout: A layout editor is used for editing some irregular cells such as a D flip-flop, while most of the layout are drawn using a LISP-like language called

SKILL. The language allows us to perform low level operations such as drawing rectangles; a library of higher level routines are also developed to perform standard cells place-and-route operations(Refer to Appendix A). Layout-Versus-Schematic (LVS) is then performed to verify the layout with the schematic. LVS can be done on a plainly extracted netlist from the layout, or on a hierarchically extracted netlist from the LVS view of the modules.

The layout design of the different modules depends heavily on the budgeted total area and the floorplan. To minimize the rerouting of the data paths between the different modules, many schematics for the Reliable Router were redrawn to rearrange the bit definition of the data path, i.e. to pitch-match the data path. Changes also happen to the schematics at other situations: when common signals of different cells are factored out to reduce area consumption, when buffers are added to accommodate the capacitive loads, or when cells are regrouped to reduce the amount of wiring. All the modified schematics are re-verified using Verilog simulation, either on the original test suites or on modified ones.

Other SKILL tools are developed[2] to help in the layout process. The "module generator generator" converts a schematic netlist into SKILL code, which forms the framework for manual place-and-route. The "logic gate generator" converts CMOS logic gates schematics into layout.

5. Timing Verification: The amount of timing verification done by the Router group is minimal because of the limited amount of man-hours. SPICE simulations on extracted netlists from layout are performed only on critical paths as determined from the schematics.

   More sophisticated timing analysis tools, such as Veritime, could have be used if the project time frame allowed. Such tools can read in annotated timing information for individual logic cells, delay information for each net, and a

[2]By Larry Dennison.

schematic netlist. A full timing analysis can then be performed on the whole VLSI system.

5.2 Hurdles and Lessons

The Reliable Router Team encountered different problems during the design process. Some were solved; others are simply facts of life. Throughout the design process, the team has learned a lot which may benefit another VLSI design team:

1. Learning the Tools[3]: The CAD tool set that we use has a steep learning curve; the Router Team perhaps spent as much time learning the tools as drawing the schematics. A huge amount of time was spent on understanding the correct setup for the tools, and learning the different features.

   The initial tool setup period could have been reduced if more assistance from the vendor were obtained. A MOSIS technology file and some courses about the tools would have been helpful. The best way for a novice user to learn the tool is to ask for a demo from another user.

2. Picking the Right Tools: The Router Team has learned that understanding what a tool can do, and where it would do a good job is crucial.

   The team once considered doing logic synthesis of schematics from Verilog. We gave up on the idea and switched to manual schematic editing after a few trials. It is difficult to specify the constraints so that the logic synthesis tools would do what it is expected; the schematics generated are also not intuitive enough to be readable. We learned that the logic synthesis tools could not be easily configured to synthesize a large state machine in a reasonable fashion, while schematics for a small state machine could be edited manually as quickly. The reason, of course, has a lot to do with the fact that we did not invest a lot of effort in learning the synthesis tools.

[3] A tool set from Cadence Design Systems.

In another situation where we put into a considerable amount of effort, we did not benefit from the tool either. We spent about 2 man-months in learning the automatic place-and-route tool from a graphical user interface; then we spent another man-month in writing the SKILL code to drive the place-and-route tool. The tool did help us in shipping a small test chip in a short period of time. However, the tool failed to optimize as required on the Reliable Router, which is area-limited. Manually generated layout, especially for regular data path structures, is much denser than that generated by the tool. We learned that the automatic place-and-route tool is good at generating a "quick-and-dirty" block of standard cells logic, but is inferior to manual place-and-route in assembling regular structures.

3. Integrating the Tools: A considerable amount of time was spent on integrating different tools. In particular, the generation of a HSPICE netlist from either a schematic or an extracted layout requires special SKILL code for resizing the transistors(Refer to Appendix B). The generation of a Verilog netlist using gate primitives, which is much faster for simulation, also requires set-up at the schematic level.

   The team has learned that tool integration involves a lot of ad-hoc hacking. The more abstraction a tool has, the more difficult it is to be integrated with other tools as desired.

4. Testing: As mentioned in section 5.1, the design process involves a lot of changes in the schematics. A typo in naming a bit of a bus, or a click of the mouse on the wrong pin would result in a wrong design.

   The team has learned that the development of a complete Verilog test suite for each module is desirable. This ensures that a module preserves its function across the many schematic changes. The Router Team, however, fails to maintain such a test suite because of the amount of work involved.

5. Communication: A weekly group meeting, or a design review, is highly recommended. Although it is true that a lot of the design decisions happen during informal white-board session, a group meeting allows team members to have a better understanding of modules they are not working on. The knowledge is important when the different modules are integrated into a single design.

   A group meeting also provides a formal mechanism for progress checkpoints, which is important for delivering the design on time.

5.3 Rules of Thumb

Different people develop different design styles of their own[10]. Some general rules of thumbs for VLSI system design, as recommended by the author, are:

- Know Your Tools: Do not rely on your tools. Be prepared for an alternate plan in case your tools fail you: since they often do. Ask an experienced user about the limitations of a tool, and when it best serves its purpose. Be ready for a steep learning curve and plan for a lot of man-hours if you are the adventurer for a new tool. If it is not obvious that a tool could do a better job than you do in the scheduled amount of time, trust your intuition and do it manually.

- Use Late Binding: Allow as much flexibility as possible for changes later on. Use relative co-ordinates when you place a cell on a piece of layout; refer to the pins of the cell with its net name, instead of its position. The same rule applies to schematic editing. Use abstracted modules when you draw schematics; avoid a flat hierarchy that has only MOS transistors on the top level sheet.

- Be Budget Conscious: Architect your design on the amount of chip area you have. Start doing a floor-plan early; this helps you reduce the number of wires between modules. In most cases, extra design features mean extra area. Plan on what you can afford.

Chapter 6

Conclusion

This thesis describes the implementation of the Reliable Router. In particular, it details the simultaneous bidirectional signalling method and the design of the JTAG diagnostic port. It also discusses some of the design experience acquired by the Reliable Router Team through the implementation process of the VLSI system.

The next step for the Reliable Router project is to build a substrate for testing the functionality of the chip. A printed circuit board that accommodates a 3x3 network, with both processor and serial port access to each node would be minimal for testing most features of the Reliable Router.

Detailed performance measurement should then be done on a full scale router network. In particular, the Reliable Router allows the comparison of dimension order routing and an adaptive routing algorithm. Experiments on a network of workstations could also be performed on the Reliable Router if a high-speed SBus interface is built.

The SKILL code developed by the Reliable Router Team could be documented and packaged so that other VLSI projects could benefit from our tool base.

Appendix A

SKILL for a Typical Module Generator

This appendix includes a piece of SKILL code that does manual standard cells place-and-route. The framework of the code was generated automatically from the schematic netlist, by a SKILL routine called mgg(), written by Larry Dennison.

Some commonly used SKILL routines by the Router Team: instAlignPins() places two standard cells next to one another; instFindPinOnSide() returns the list of named pins on a side of a cell instance; wireChannel() connects a list of poly and metal2 rectangles by searching for the first available wire track in a given channel.

```
(defun mg ()
  (let ( cv I100 I101 I102 I103 I104 I105 I106 I107 I108 I109
         bb_lx bb_ux bb_ly bb_uy
         channel0 channel1 channel2 channel3
         uy ly)

    (setq cv (dbOpenCellView "frontend" "latch_and_mux" "layout" nil "w"))
    (instantiate)
    (placement)
    (setq bb_lx -2.0)
    (setq bb_ux 66.0)
    (setq bb_ly -2.0)
    (setq bb_uy 341.0)
```

```
    (setq uy (rect_ly (first (instFindPinOnSide cv I100 "gnd!" "east"))))
    (setq channel0 (rectMake bb_lx bb_ly bb_ux uy))

    (setq ly (rect_uy (first (instFindPinOnSide cv I103 "vdd!" "east"))))
    (setq uy (rect_ly (first (instFindPinOnSide cv I106 "gnd!" "east"))))
    (setq channel1 (rectMake bb_lx ly bb_ux uy))

    (setq uy (rect_ly (first (instFindPinOnSide cv I107 "gnd!" "east"))))
    (setq ly (rect_uy (first (instFindPinOnSide cv I105 "vdd!" "east"))))
    (setq channel2 (rectMake bb_lx ly bb_ux uy))

    (setq ly (rect_uy (first (instFindPinOnSide cv I107 "vdd!" "east"))))
    (setq channel3 (rectMake bb_lx ly bb_ux bb_uy))

    (wire_clk50bar) (wire_clk50) (wire_pp2) (wire_pn0)
    (wire_pn2) (wire_pp0) (wire_rsel) (wire_qsel)
    (wire_i) (wire_flit) (wire_net26) (wire_net24)
    (wire_net25) (wire_net450) (wire_net460) (wire_net11)
    (wire_net473) (wire_net22) (wire_q) (wire_power)

    (dbCreateRect cv (list "prBoundary" "drawing")
                     (rectMake bb_lx+2 bb_ly+2 bb_ux-2 bb_uy))
    (dbSave cv) ))

(defun instantiate ()
  (let (cmos2 dl_2c inv in3x)
    (setq cmos2 (dbOpenCellView "ghost" "cmos2" "layout" nil "r"))
    (setq dl_2c (dbOpenCellView "ghost" "dl_2c" "layout" nil "r"))
    (setq inv (dbOpenCellView "ghost" "inv" "layout" nil "r"))
    (setq inv3x (dbOpenCellView "ghost" "inv3x" "layout" nil "r"))

    I100 = (dbCreateInst cv dl_2c "I100" (list 0.0 0.0) "R0")
    I101 = (dbCreateInst cv inv "I101" (list 0.0 0.0) "R0")
    I102 = (dbCreateInst cv inv "I102" (list 0.0 0.0) "MY")
    I103 = (dbCreateInst cv dl_2c "I103" (list 0.0 0.0) "MY")
    I104 = (dbCreateInst cv inv "I104" (list 0.0 0.0) "R0")
    I105 = (dbCreateInst cv cmos2 "I105" (list 0.0 0.0) "R0")
    I106 = (dbCreateInst cv inv "I106" (list 0.0 0.0) "R0")
    I107 = (dbCreateInst cv dl_2c "I107" (list 0.0 0.0) "R0")
    I108 = (dbCreateInst cv dl_2c "I108" (list 0.0 0.0) "MY")
    I109 = (dbCreateInst cv inv3x "I109" (list 0.0 0.0) "R0")
    ))
```

```
(defun placement ()
  I100~>xy = (list 0.0 16.0)
  (instAlignPins I101 "gnd!" "west" I100 "gnd!" "east")
  (instAlignPins I102 "gnd!" "west" I101 "gnd!" "east")
  (instAlignPins I103 "gnd!" "west" I102 "gnd!" "east")
  (instDX I103 -8)

  I105~>xy = (list 0.0 134.0)
  (instAlignPins I104 "gnd!" "west" I105 "gnd!" "east")
  (instDX I104 5)
  (instAlignPins I106 "gnd!" "west" I104 "gnd!" "east")

  I109~>xy = (list 0.0 252.0)
  (instAlignPins I108 "gnd!" "west" I109 "gnd!" "east")
  (instAlignPins I107 "gnd!" "west" I108 "gnd!" "east"))

(defun wire_clk50 ()
  (let ( r1 r2)
    (setq r1 (first (instFindPinOnSide cv I108 "g" "south")))
    (setq rc (wireChannel cv nil (list r1) channel2
                          ?gravity "top"
                          ?westExtend t
                          ?eastExtend t))

    (pinMake cv "clk50" "metal1" (rectSet_ux rc bb_lx+4.0) ?direction "input")
    (pinMake cv "clk50" "metal1" (rectSet_lx rc bb_ux-4.0) ?direction "input")

    (setq r2 (first (instFindPinOnSide cv I107 "gn" "north")))
    (setq rc (wireChannel cv nil (list r2) channel3
                          ?gravity "bottom"
                          ?westExtend t
                          ?eastExtend t))
    (pinMake cv "clk50" "metal1" (rectSet_ux rc bb_lx+4.0) ?direction "input")
    (pinMake cv "clk50" "metal1" (rectSet_lx rc bb_ux-4.0) ?direction "input")
    ))

(defun wire_pn0 ()
  (let ( r1)
    (setq r1 (first (instFindPinOnSide cv I103 "gn" "north")))
    (setq rc (wireChannel cv nil (list r1) channel1
                          ?gravity "bottom"
                          ?westExtend t
                          ?eastExtend t))
    (pinMake cv "pn0" "metal1" (rectSet_ux rc bb_lx+4.0) ?direction "input")
    (pinMake cv "pn0" "metal1" (rectSet_lx rc bb_ux-4.0) ?direction "input")))
```

```
(defun wire_net26 ()
  (let ( r1 r2 r3)
    (setq r1 (first (instFindPinOnSide cv I101 "A" "north")))
    (setq r2 (first (instFindPinOnSide cv I102 "A" "north")))
    (setq r3 (first (instFindPinOnSide cv I100 "qn" "north")))
    (wireChannel cv (list r3) (list r1 r2) channel1
                  ?gravity "bottom")))

(defun wire_net460 ()
  (let ( r1 r2)
    (setq r1 (first (instFindPinOnSide cv I107 "d" "south")))
    (setq r2 (first (instFindPinOnSide cv I106 "Y" "north")))
    (wireChannel cv (list r2) (list r1) channel2
                ?gravity "south"
                ?polyContDX -2)))

... <other nets deleted>
```

Appendix B

SKILL for the HSPICE Netlister

This appendix includes a piece of SKILL code which directs the netlister to generate a line for a HSPICE MOS instance. A considerable amount of SKILL code similar to the following was written to customize the Cadence set of tools.

```
procedure( hnlHspicePrintScaledMOSfetElement(scale)
  prog( ( s m i w l src gate drn off scaled_l scaled_w substrate area para)
    m = hnlHspiceUniqueBlockName( hnlCurrentMaster )
    i = hnlMapInstName( hnlCurrentInstName )
    l = hnlHspiceInstPropVal( "l" )
    w = hnlHspiceInstPropVal( "w" )
    src = hnlHspiceNetOnTerm( "S" 0 )
    gate = hnlHspiceNetOnTerm( "G" 0 )
    drn = hnlHspiceNetOnTerm( "D" 0 )
    off = hnlHspiceInstPropVal( "off" )

    if( ( w == nil ) then
       warn( "nil width in master '%s', instance '%s'\n" m i )
       w = "4.0"
    )
    if( ( l == nil ) then
       warn( "nil length in master '%s', instance '%s'\n" m i )
       l = "2.0"
    )

    scaled_l = evalstring(l) * scale
    scaled_w = evalstring(w) * scale
    area =  scaled_w * (5 * scale)      ;; assumes w * 5 lambda s/d region
    para =  scaled_w + (2* 5 * scale)   ;; only counts three sides
```

```
if( ( off == nil ) then
   off = ""
)

substrate =  "UNKNOWN"
if( (strcmp(m "pmos") == 0) then
   substrate =  "vdd" )
if( (strcmp(m "nmos") == 0) then
   substrate =  "gnd" )
if( (strcmp(substrate "UNKNOWN") == 0) then
   printf("unknown device - hnlHspicePrintScaledMOSfetElement\n") )

sprintf(s "m%s %s %s %s %s %s w=%g l=%g as=%g ad=%g ps=%g pd=%g m%%s\n"
        drn gate src substrate m scaled_w scaled_l area area para para off)
hnlPrintString(s)
))
```

Bibliography

[1] Barbara A. Chappell, Terry I. Chappell, Stanley E. Schuster, Hermann M. Segmuller, James W. Allan, Robert L. Franch, and Phillip J. Restle. Fast CMOS ECL receivers with 100-mV worst-case sensitivity. In *IEEE Journal of Solid-State Circuits*, February 1988.

[2] William J. Dally. *Network and Processor Architectures for Message-Driven Computers*, chapter 3. Morgan Kaufmann Publishers, Inc., 1990.

[3] William J. Dally. Virtual-channel flow control. In *IEEE Transactions on Parallel and Distributed Systems*, March 1992.

[4] William J. Dally and Hiromichi Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. In *IEEE transactions on Parallel and Distributed Systems*, April 1993.

[5] William J. Dally, Larry R. Dennison, David Harris, Kinhong Kan, and Thucydides Xanthopoulos. The reliable router: A reliable and high-performance communication substrate for parallel computers. In *Proceedings for Parallel Computer Routing and Communication Workship*, May 1994.

[6] Larry Dennison. Reliable interconnection networks for parallel computers. Technical Report 1294, MIT AI Laboratory, October 1991.

[7] Larry R. Dennison, Whay S. Lee, and William J Dally. High-performance bidirectional singalling in VLSI systems. October 1992.

[8] Christopher J. Glass and Lionel M.Ni. The turn model for adaptive routing. In *Proceedings of the 19th International Symposium on Computer Architecture*, May 1992.

[9] Kin Hong Kan. *The Design of a High Performance Simultaneous Bidirectional MOS Driver*. Bachelor's thesis, Massachusetts Institute of Technology, May 1993.

[10] Butler Lampson. Hints for computer system design. In *Proceedings of the Ninth ACM Symposium on Operating Systems Principles*, October 1983.

[11] Test Technology Technical Committee of the IEEE Computer Society. *IEEE Standard Test Access Port and Boundary-Scan Architecture*. The Institute of Electrical and Electronics Engineers, Inc., February 1990.

[12] Thucydides Xanthopoulos. Adaptive turns: A deadlock-free and fault-tolerant adaptive routing algorithm for wormhole networks. November 1992.