VLSI Implementation of a Wormhole Run-time Reconfigurable Processor

Maneesh Soni

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Dr. Peter Athanas – chair
Dr. Mark Jones
Dr. Jeff Reed

June 2001
Blacksburg, Virginia, USA

Keywords: CCM, VLSI, RTR, DSP

VLSI Implementation of a Wormhole Run-time Reconfigurable Processor

Maneesh Soni

(ABSTRACT)

Until now, the performance improvement of computing machines was a mostly a result of shrinking transistor geometries and increasing clock speeds. With the advent of signal processing applications that have stringent performance requirements from processing hardware, the field of configurable computing has received a lot of attention. Efforts are being made to improve computation bandwidth by architectural innovations. Among these, the *wormhole runtime reconfigurable* architecture introduces the concept of *stream* processing. It enables dynamic reconfiguration of hardware with little overheads and is very much suited for data-path based computations with deep computational pipelines. Stallion, second in the generation of Wormhole runtime reconfigurable processors, demonstrates the efficacy of wormhole runtime reconfiguration. The work presented here deals with the VLSI implementation of Stallion and discusses the full-custom physical design flow adopted for Stallion. Also, the tools and techniques to customize this flow are detailed. The Stallion design methodology offers a possible solution that can be pursued for executing similar efforts in future.

Acknowledgements

While working towards the goal of prototyping Stallion, I have received tremendous amount of motivation, support and co-operation from a large number of people. This, needless to mention, is the most opportune moment to acknowledge my gratefulness.

First of all, I express deep sense of gratitude towards my advisor, Dr. Peter Athanas. Without him, I would neither have gotten the opportunity nor would have taped Stallion out. I also sincerely appreciate the willingness of Dr. Mark Jones and Dr. Jeff Reed to be on my advisory committee and provide motivation and support for the project.

It would have been very difficult on my time had Kiran Puttegowda not teamed with me at the right time and created layouts for several blocks in Stallion design. It was with his help and company in the long nights in CCM Lab that the tape out of Stallion processor became a reality.

I also appreciate the help rendered by Srikathyayani Srikanteswara and Jody Neel of the Mobile and Portable Radio Research Group (MPRG). They made it easier for me to overcome the obstacles that one faces in the initial phases of a new project. Kathyayani also furnished information about the performance of Stallion towards the later course of this work.

The help of Luke Scharf and John Harris is also thankfully acknowledged. Their excellent understanding of the temperament of computers helped me get over the software related issues. I also acknowledge Jos Sulistyo for helping me with the VISC computing facilities.

I also thank Dr. Sanjay Raman and Daniel Johnson for the photos of Stallion chip taken in the Wireless Microsystems Lab.

Without the financial support that I received as a research assistant at Virginia Tech, this

thesis would hardly be in existence. I am thankful for that and express sincere admiration for the fact that Virginia Tech provides ample opportunities to students from countries around the world.

I am grateful to my parents and family members for their unconditional love and faith in me. I also thank all my friends, at VT and beyond, who have kept me continually focused on the task at hand and provided me support whenever I needed.

Finally, I thank all my colleagues in the Configurable Computing Lab who form one of the smartest communities of technocrats at Virginia Tech. I gratefully acknowledge them for providing cordial and enthusiastic atmosphere in the CCM Lab.

Maneesh Soni
June 2001

CONTENTS

List of Illustrations

List of Tables

Chapter 1

Introduction

The field of Configurable Computing Machines (CCM) has been the focus of research for about a decade now. These machines are attractive due to the escalating demand for powerful computing platforms suited mainly for the high-end signal processing applications. Digital Signal Processor (DSP) technology has demonstrated considerable success in meeting these needs. However, it has been shown that silicon utilization of even DSPs is low. While rapid advancement in VLSI technology has contributed to increase in performance of computing hardware, innovations in the architecture of computing machines to improve performance have been modest. The Stallion processor [1], second in its generation of Wormhole run-time reconfigurable processors, offers a novel architecture to speed up computation performance. Designed by Ray Bittner, Stallion processor is a novel CCM that offers advantages of ASICs while retaining the flexibility of general-purpose processors and FPGAs [2]. Its hardware can be re-programmed during run-time and has processing elements specially suited for DSP applications. This thesis details the process of translating the design of Stallion processor from schematic to layout. Various issues involving full-custom physical design of VLSI in sub-micron geometries are also discussed. Effort has also been made to document this experience in a style that aims to assist future endeavors of similar nature.

1.1 Methodology

CCMs, in general, are comprised of processing elements in a structured topology and a programmable interconnection network. This architecture, being inherently multi-

dimensional, complicates the task of flattening the design on silicon medium. The work documented here describes how the multi-layered architecture of Stallion was laid out on silicon. As means to this end, Cadence VLSI CAD tools were employed for most of the physical design work. Among these, Virtuoso family of tools comprising of Schematic Composer, Layout Editor, Layout XL; Diva verification tools suite comprising of Design Rule Check (DRC) and Layout versus Schematic (LVS) and IC Craftsman for placement and routing were the most used ones. Synopsys Design Compiler was used to synthesize small portions of the design. Data translation and scaling tools were used early and towards the end of the design cycle and were intended for fabrication technology migration and format conversion of the design database.

Chip fabrication was done by the MOSIS IC prototyping service using scalable CMOS technology based on TSMC25 process. This is a five metal and one poly-silicon layer process with minimum drawn feature size of 0.3 μm and effective minimum feature size of 0.25 μm. The design rules are based on lambda parameter. In the case of Stallion, λ was set at 0.15 μm.

1.2 Contributions

This work demonstrates the application of VLSI design techniques to successfully build Stallion, a chip exceeding half a million transistors. It also demonstrates how a multi-dimensional architecture can be mapped onto silicon. A full-custom physical design methodology that evolved with the progress of the work has also been documented. It is shown that with a subset of the wide spectrum of tools available for VLSI design, it is possible to develop a successful strategy to make the design process efficient and productive. At the same time, the shortcomings of such an approach have also been exposed. A full-custom design approach mandates thorough understanding of the physical design as well as the use of CAD tools. As the design size grows, the full-custom design flow manifests its complexity when it comes to fixing errors in the layout. It can be very time-consuming and frustrating to locate wrong connections in a large multi-layered layout.

Finally, this work also provides a strategy that can be pursued in future to implement VLSI designs that are of comparable size or even larger. The design practices highlighted here would hopefully make the future efforts more productive and better informed of the upcoming issues.

1.3 Organization

Chapter 2 documents various efforts that have been made in academia and in industry aimed at creating computing machines similar to FPGAs but focused at mitigating the shortcomings of FPGAs and retaining ASIC advantages. The architectures that have been documented have been geared towards creating integrated circuits that have programmable processing elements and interconnection network of some type. Chapter 3 describes the architecture of Stallion processor in brief. This will give the reader a good background for the material presented in the following chapters of this document. In chapter 4, the full-custom physical design methodology adopted for Stallion has been illustrated. Various tools associated with each step of the design cycle are also briefly explained. This discussion will provide foundation for the detailed discussion of Stallion's physical design process in subsequent chapters. Chapter 5 illustrates the floor plan and layout of individual components. Verification scheme that was adopted for Stallion is also discussed. It also elaborates on the flow used for creating the layouts. Chapter 6 summarizes the effort and gives suggestions for future efforts in this direction. Some results obtained from simulation of Stallion processor are also presented. Appendix A illustrates the Stallion design hierarchy. In Appendix B, the process parameters of MOSIS fabrication run are presented. The list of pins and package used for Stallion is given in Appendix C. In Appendix D, the details of calculations made for power consumption in Stallion are given. More details about VLSI CAD tools used are offered in Appendix E. Some photographs of the Stallion die are shown in Appendix F.

Chapter 2

Background

Performance expectations from computing machines are exploding due to the recent growth in consumer products aimed at providing personal communication services that demand high bandwidth, high quality, low power consumption and low cost. These requirements put great strain on the conventional computing architectures. A paradigm shift in the design of computing machines to obtain high performance coupled with small design time and flexibility is being witnessed in the form of Configurable Computing Machines. Most conventional processors have low efficiency because of "forced serialization of intrinsically parallel operations; wasted space (small data elements do not use processor's wide data path); and excessive instruction bandwidth for regular data-flow dominated computations on large data sets" [3]. Traditional FPGA based computing offers several advantages in the short design time and speeds approaching ASICs and comparable with DSPs. However, most commercial FPGAs suffer from limitations such as coarse logic element granularity, long configuration and compilation time and limited reconfiguration bandwidth [1, 3]. To overcome the shortcomings of conventional processors and FPGAs, efforts have been ongoing to create innovative CCMs architectures as an alternative to microprocessors, Digital Signal Processors and FPGAs by using hardware that can be reconfigured on the fly. In this chapter, a survey of such efforts made in industry and academia is presented.

2.1 PipeRench

At Carnegie Mellon University, PipeRench – a re-configurable fabric consisting of inter-connected processing units and storage elements – was invented [3]. It is targeted towards

data path type computations that are mainly used in DSP applications. Based on the concept of *cached virtual hardware*, it implements pipelined computations involving *v* steps on its re-configurable processing fabric in *p* (where *p*<*v*) physical stages by run-time reconfiguration. Thus, the computations involving more stages than are physically available on PipeRench can also be mapped by using some of the pipeline stages more than once and run different configuration on them every time. Configuration is performed from an on-chip re-configuration buffer that is controlled by a small external controller. A pipeline stage can be configured while other stages are running thereby maintaining efficiency and overlapping configuration with execution.

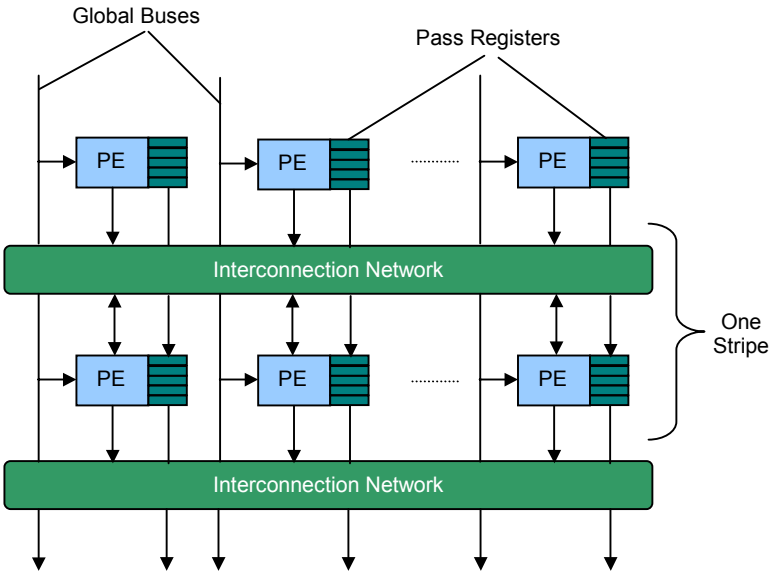

Figure 2-1 PipeRench Architecture

As illustrated in Figure 2-1, PipeRench consists of physical pipeline stages called *stripes*. Each stripe consists of a set of processing elements (PE) and an interconnection network. The processing element has an ALU and a set of pass registers. One or more ALUs are used to implement combinational logic. The ALUs can be cascaded to increase the width of operation. Interconnection network provides access to registered outputs of previous stripe or of the ALUs in the same stripe. Inputs and outputs for an application are transmitted over the global buses. Pass registers in each stripe offer a convenient way of inter-stripe connectivity. Application programs can write the output of ALU to any of the registers in the register file. Otherwise, the register takes the value of corresponding register of the register file in previous stripe. Thus, the register file provides pipelined connectivity between PE in

one stripe and corresponding PE in the subsequent stripe. A barrel shifter in each PE makes bit alignment in word-based computations possible.

Performance of PipeRench compares very well with other processors. It gives a performance speedup of approximately 10 to 190 times compared to a 300 MHz UltraSparc II processor on algorithms like DCT, ATR etc. The PipeRench architecture, programming and performance have been treated in greater detail in [3, 4].

2.2 Re-configurable communications processor (RCP)

A recent ongoing effort in commercial world is the reconfigurable communications processor (RCP) from Chameleon Systems, Inc. The RCP architecture has a reconfigurable fabric along with a 32-bit PCI Controller, 32-bit ARC processor core and a 64-bit memory controller [5]. There is also a 128-bit wide bus for high-speed data transfer among various blocks of the RCP.

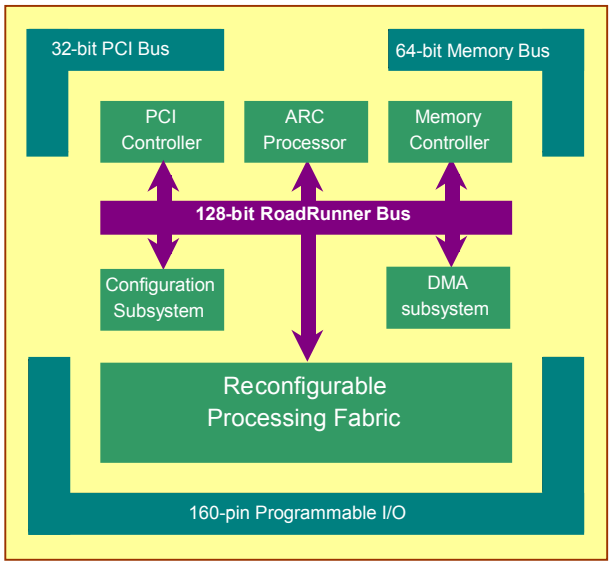

Figure 2-2 Architecture of CS2000 RCP

The RCP chip contains a 32-bit ARC processor core that provides 120 MIPS at 125 MHz and is used to perform higher-level control tasks like processing-fabric reconfiguration. The processor core can access configuration information of every tile in the reconfigurable fabric, memory contents and registers. A PCI Bus controller enables RCP to be used as a part of a

large PCI based system. Through the Memory bus, RCP can access external memory devices. Apart from that, the programmable I/O pins provide large bandwidth for data streaming applications. The I/O pins are capable of interfacing to SRAMs, A/D, D/A and FPGAs.

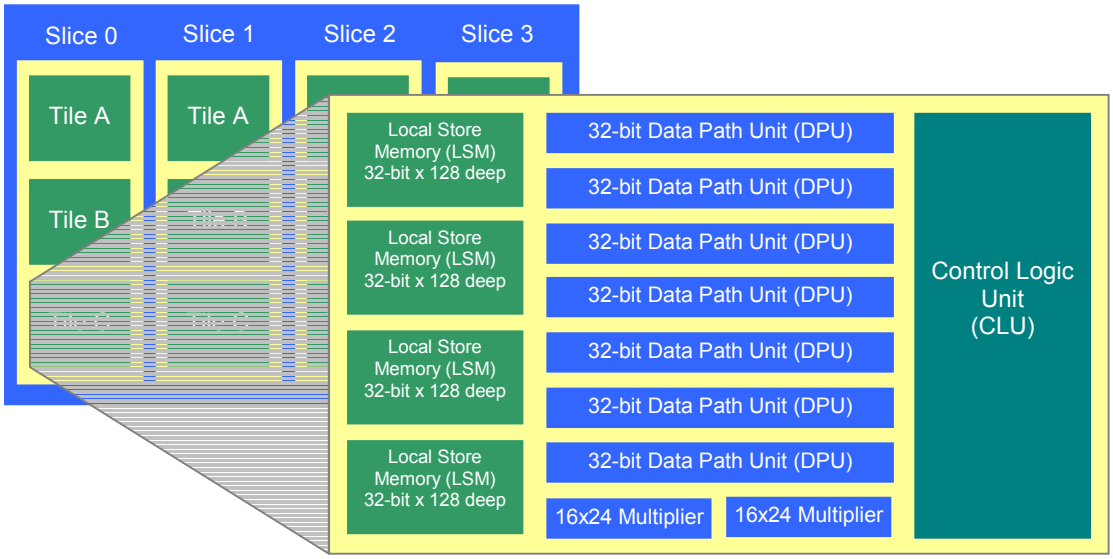

Figure 2-3 Components of a tile in RCP2000 processor

The Reconfigurable Processing Fabric (RPF or Fabric) consists of twelve smaller identical Reconfigurable units called Tiles. Each Tile has seven 32-bit data processing units (DPU), four blocks of 32-bit x 128-deep memory, two 16x24 multipliers and a control logic unit. The DPU supports all C and Verilog operations. Memory modules can be programmed to build wider or deeper memory blocks. The control unit is a state machine that controls each DPU in a tile. It stores instructions for each of the seven DPUs with each instruction equivalent to storing complete configuration information for the DPU.

Each of the tiles in RPF has two configuration storage planes. The background plane can be loaded independently without interfering with the configuration in active plane and the ongoing processing. Switching between the active and shadow plane takes 3 µs. It is possible to switch between various sections of an algorithm at a very fast pace. Thus, the CS2000 circuit has considerable amount of computing resources to cater to the demands of high-end signal processing applications.

2.3 Context Switching FPGA

Sanders, a Lockheed Martin company, developed an FPGA, among the first in the class of context switching reconfigurable computing (CSRC) devices, that is capable of storing four different configurations and able to switch among them as needed. This context switching FPGA makes it possible to switch between different programmed tasks without the need of additional FPGAs [7]. The device can hold four different configurations and can switch from one configuration to another in one clock cycle. It is possible to retain the registers between contexts and remember the value of registers at last context switch. The CSRC chip was designed with a 4-bit wide data path. However, it has carry logic circuitry to scale the width and implement data-path of arbitrary size.

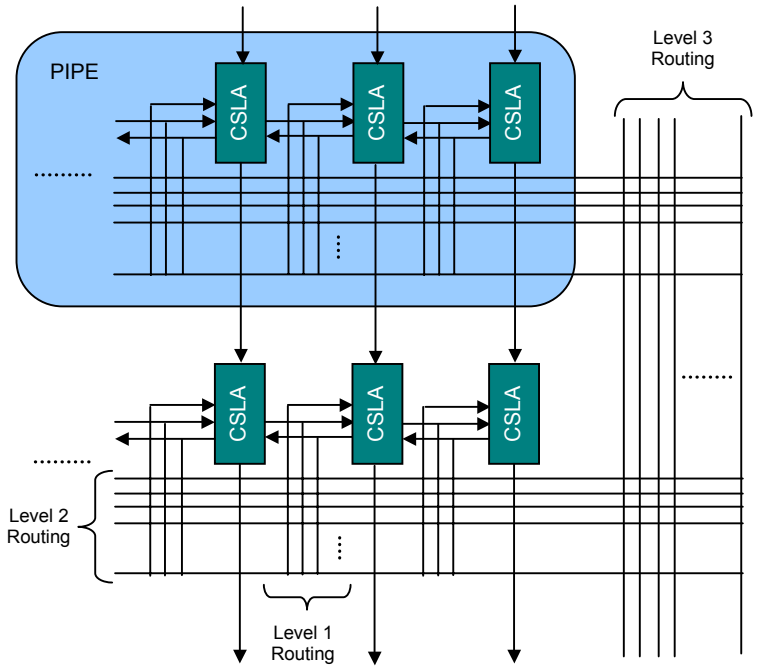

Figure 2-4 Architecture of CSRC chip

The CSRC device is arranged in 16-bit wide processing units called context switching logic arrays (CSLA) forming 16-bit wide data path called the *pipe*. The output of each CSLA is available to the adjacent CSLAs. This makes both left-to-right and right-to-left data processing possible within a pipe. The input and output data from each pipe is available on Level 2 routing buses. The buses of Level 2 routing span across the width of the chip. With

this routing, any signal driven on Level 2 buses is accessible to any CSLA in the pipe. A stack of CSLA data pipes one on top of the other build the CSRC chip. Carry logic circuitry between adjacent pipes make it possible to create processing system with arbitrary data path width. Another layer of interconnects, called the Level 3 routing, runs from top to bottom of the chip. The buses in Level 3 routing provide connectivity to Level 2 buses. A signal driven on Level 3 buses can be tapped by any of the Level 2 routing. There is no segmentation in either Level 2 or Level 3 routing.

The CSLA block has sixteen context switching logic cells (CSLC). The CSLC forms the computation unit of CSRC chip and it also implements context switching. Each CSLC has a 4-input LUT, a context-switching flip-flop and a tri-state output. The LUT has sixteen configuration bits that implement a programming function. These bits, called the context switching bits (CSBits), are specific to the context of CSRC chip and the current context determines which of the four will be fed to the LUT. Thus, context switching in CSRC chip is implemented in the processing unit itself. Context information is stored across the chip in a distributed manner.

CSRC chip provides the functionality to retain data of each context during two consecutive context switches. Thus, a context can start processing from where it left off last time it was active. In CSRC chip, two or more contexts can also share data. Thus, one context can use the results of last context as its input. This makes it possible to exploit the benefits of hardware context switching in an efficient manner.

By bringing the concept of software task switching to hardware, the CSRC chip aims to implement entire algorithms that were inconceivable to implement without the concept of context switching.

2.4 Jazz Processor

Re-configurable cores for computing are also finding their place in the computing market and the Jazz Processor core from Improv Systems, Inc. [6] is one such product. It is a

configurable DSP core that can be used as a single data processing unit or several cores can be combined in an interconnect structure to provide very high bandwidth of computation. According to the computation and data path characteristics of the application, the system designer configures each Jazz processor core.

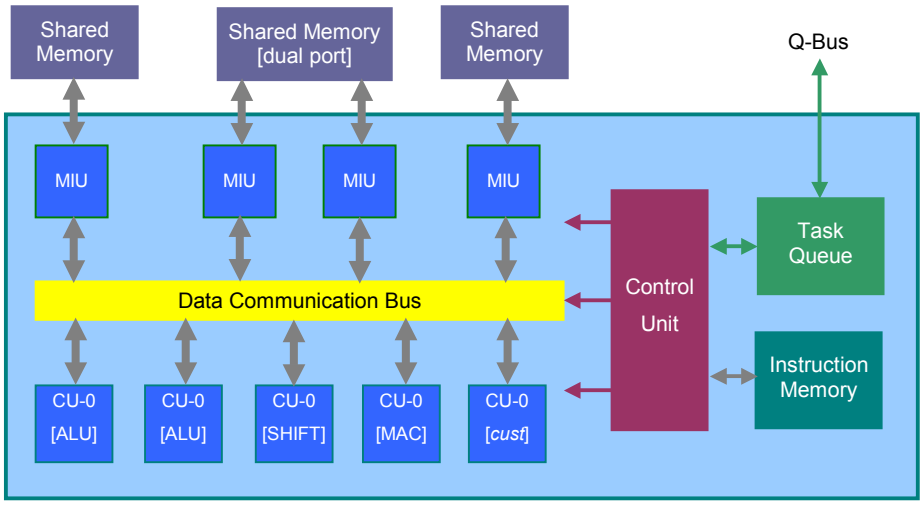

Figure 2-5 Jazz Processor Architecture

The Jazz processor has several customized ALUs and the designer can modify the operand widths, add extra computation elements (CU) and customize the instructions. Each of the computation unit has a VLIW operation with 13-16 instructions per cycle and a 2-stage pipeline. The multiplier unit has single cycle MAC operation. The CUs communicate with memory segments that are shared among various components of the processor and the external world via the Memory Interface Units (MIU) over a Data Communication Bus. A control unit exercises all the managerial tasks depending upon the instructions from a higher-level control system or from host CPU via an integration unit. Control information is transmitted on a proprietary Q-bus. The Q-bus also makes the task of integrating several Jazz processors simplified.

As shown in the Figure 2-5, multiple Jazz cores can be integrated to create System-on-chip (SoC) designs. Each of the Jazz cores can be configured separately. There is a library of standard integration blocks to integrate Jazz cores to other general-purpose microprocessors.

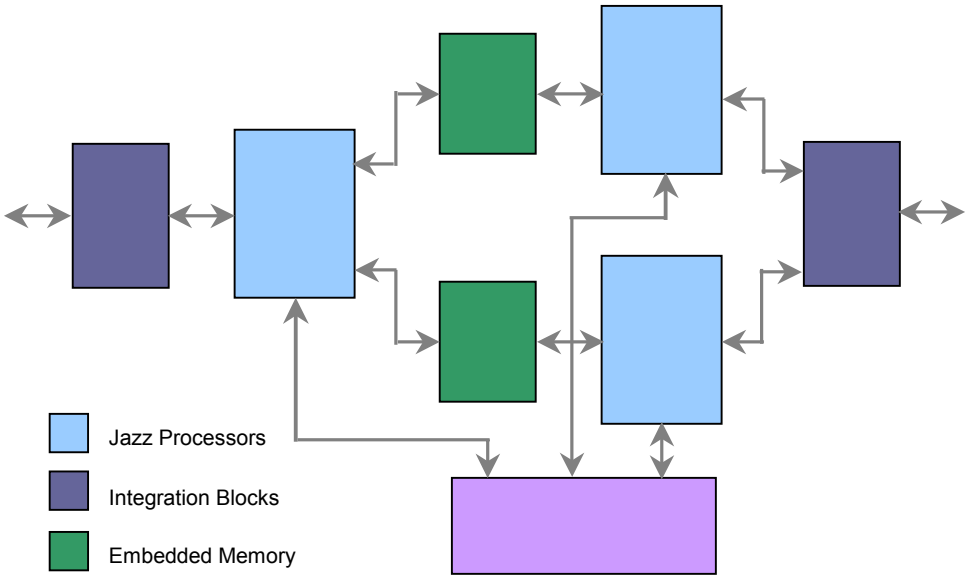

Figure 2-6 Interconnection of Multiple Jazz Processors

The concept of Jazz cores enables configurable designs at compile time. The cores are not run-time reconfigurable. However, by making the task of customizing the cores intuitive and simplified, Jazz aims to reduce design time and effort.

2.5 Chimaera Configurable Processor

The bandwidth available for data transmission between processor and off-chip configurable logic is one of the critical efficiency issues in configurable computing. At Northwestern University, Chimaera re-configurable processor (RCP) was developed to improve data transmission bandwidth between the host processor and re-configurable fabric [8]. Chimaera chip contains a microprocessor with an integrated on-chip reconfigurable functional unit (RFU). In Chimaera design paradigm, the reconfigurable logic is seen as a cache for RFU instructions. The instructions that have either recently executed or are expected to be executed soon are stored in the reconfigurable array. The application running on host processor has instruction calls to the RFU. When such an instruction is called, then a check is performed to see if it is in the RFU. If it is not in the reconfigurable array, it is brought in and some of the existing instructions may be overwritten. The inputs to the instruction being

executed are read from the (shadow) register file of the host processor and results from the RFU are sent back to processor through the register file. Every instruction opcode specifies which registers are read and written to.

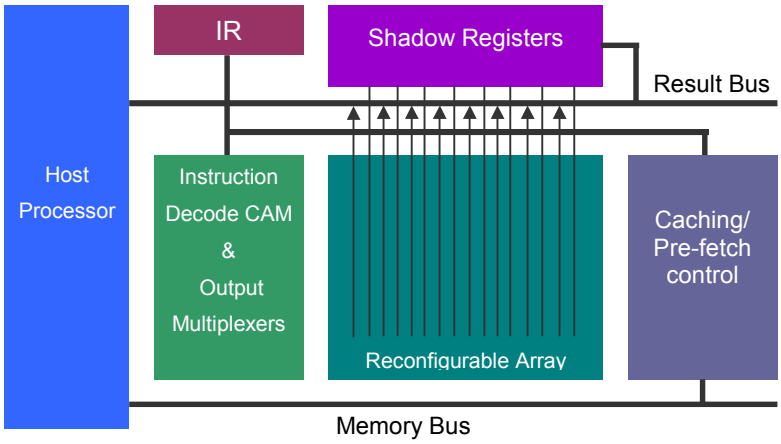

Figure 2-7 Chimaera Processor Architecture

Chimaera RFU consists of the reconfigurable array, a shadow register file, instruction decode logic, output multiplexers, instruction caching and pre-fetch control circuitry and bus units for data transfer. The main component is the FPGA like reconfigurable logic where all instructions actually get executed. The reconfigurable logic is divided in rows of logic with each row containing logic cells equal to the word size on the host processor. The logic required for each instruction is occupied in one or more of such rows. Capability is provided so that registers from data file can be read and written to by the reconfigurable array. Instruction decode CAM in Chimaera performs the check to see if the logic for the next instruction to be executed is available in the reconfigurable array or not. If it is available, the instruction is executed. Otherwise, the Caching/Pre-fetch Control logic stalls the host processor and the new instruction is configured in one or more rows causing some of the older instructions to be wiped off. The CAM logic also determines and controls the routing of output from the row corresponding to the executed instruction to the result bus.

It has been seen the Chimaera architecture improves performance of algorithms by executing complex instructions in hardware and reduces the time taken to move data to and from the host processor.

2.6 Summary

In the previous sections, several CCM architectures have been described. Each of these, with an alternative architecture, aims at improving computation performance. The PipeRench architecture is targeted at increasing the speed of pipelined DSP computations by using a configurable fabric along with an external controller. In contrast, the RCP architecture offers a standalone solution for applications that require very high computation bandwidth. The CSRC chip introduces the concept of context switching in hardware to maximize processing capacity and minimizing the configuration overheads. Also, there have been efforts to create soft cores based on CCM concepts. Jazz processor is one such product that allows custom configuration of the computational blocks at the time of compiling the HDL description. It aims to provide flexibility to the designer by bringing the configurable core to the SoC paradigm. The Chimaera architecture tries to combat the data transfer bandwidth between a host processor and FPGA type computing fabric.

In this scenario, the Stallion architecture brings forth both – capability of dynamic reconfiguration and high bandwidth of computations. The stream based approach of transmitting configuration bits and data bits in tandem merges the tasks of configuration and processing that have traditionally been treated separately. Stallion also offers high reconfiguration bandwidth as all the pins can be utilized to transmit configuration information when the chip is being programmed. By having several streams running simultaneously, Stallion processor is never unavailable for computations. Apart from that, with a large resource pool for computations, Stallion is also more capable of meeting stringent performance requirements of the new generation of complex signal processing applications like the 3G wireless communications.

Chapter 3

Stallion Architecture

Contributing to the research in the area of high-performance computing, a family of CCMs has been created at Virginia Tech with a novel design approach. This approach, referred to as *Wormhole Runtime Reconfiguration*, offers fast computations along with partial runtime reconfiguration capability. Colt, the first among these CCMs, was conceptualized and designed by Ray Bittner [1]. It was targeted towards signal processing applications and implemented the concept of stream-based data processing. Dr. Bittner had also proposed Colt's successor, Stallion, which has much larger resource pool, additional functionality and improved design. This thesis documents the task of prototyping Stallion. In order to give the reader background for the work done herein, this chapter discusses architecture of Stallion processor in limited detail.

3.1 Overview

Stallion architecture consists of three interconnect units: *data ports*, *Cross Bar* and *Meshes*. The data ports are input/output units and are the only ways to communicate to the chip. Meshes are the processing units of Stallion. Cross Bar is an interconnection network between the data ports and the meshes. Stallion has six data ports; two meshes containing sixty small processing elements called interconnected functional units (IFU) and one crossbar with 22 inputs and 38 outputs.

Stallion is based on the stream concept [1]. *Stream* is defined as the concatenation of two sets of information, the programming information in the header and the operands following

the header. Programming header configures various components inside Stallion and creates a computational path that will be followed by operands in the stream. The computational path determines what processing will be performed on the operands. As the programming header traverses inside the chip, each unit gets configured. The unit then passes the rest of the stream to other blocks inside the chip according to its own configuration. Thus, as the data path configuration progresses, the stream gets stripped off its programming header. The header no longer exists after the entire data path configuration is complete. It is important to note that the order and length of programming information is not fixed. The stream, its programming header and the operand data can be of arbitrary length.

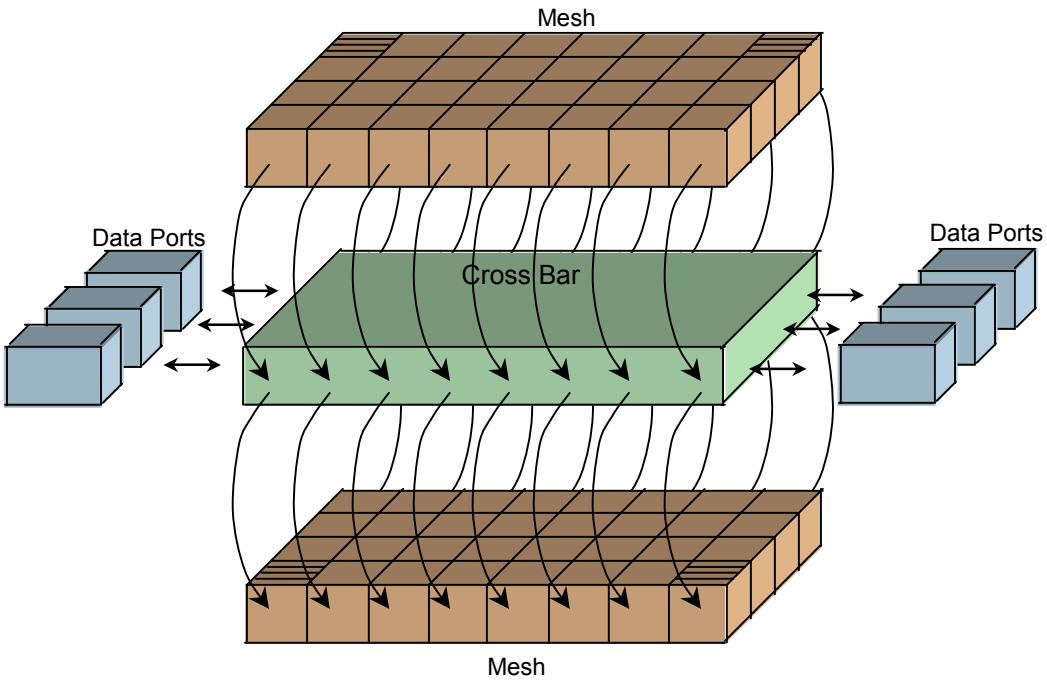

Figure 3-1 Stallion CCM Architecture

Stallion architecture supports the capability of cascading multiple chips. Thus, multiple Stallion chips can be used to create a system and scale the amount of computations that can be done.

3.2 Data Port

There are six data ports in Stallion. Data ports are used to send and receive programming headers, operands and results. Each data port is 20-pins wide, with 16 bi-directional pins for

data input/output, three bi-directional pins – *Program*, *Transmit* and *Receive* - for control and one output pin named *Write*. The *Program* pin can be pulled low from inside or outside of the chip. It is used to indicate whether or not the word on data pins is a program word. The *Transmit* pin indicates if the word is valid or not. The signal on *Receive* pin has different interpretations for program and data words. When a data port is sending out program words, it waits for the other party to pull the *Receive* pin low before sending more program words. When a data port is sending normal data, it expects the receive pin to remain high. Otherwise, when negated, data transmission is stalled indicating that the party which pulled receive line low is not ready for data reception. On the other hand, if the data port is receiving program information from the outside, it pulls the receive line low indicating its readiness. The last control pin on a data port, the *Write* pin, is an output and is used to indicate whether the data port is configured as a read port or as a write port.

The data ports have three modes of operation – Raw, Synchronize and Loop Mode. In *raw mode*, a data ports accept all data coming to the pins. In *synchronize mode,* data port uses a temporary buffer to store the current data word and signals the external circuitry that no more data will be accepted. This happens whenever the data port gets a signal from other synchronized ports that they are not ready to receive more data. Having this functionality helps in preserving valid data and protects it from being overwritten by invalid data. The third mode, *loop mode,* is useful to process computations in a lock step fashion. This is accomplished by synchronizing an output port with its input port. While operating in this mode, no new operands are accepted from outside until the current set of operands are processed and available at the output port.

The main structural components of each data port are - a state machine that controls the overall operation, an address comparator that verifies whether the programming information is to be used for configuration, a buffer to hold data in synchronize mode when the data port has to wait for processing to restart, a register to hold configuration information and tri-state logic for handling bi-directional communication.

3.3 Cross Bar

Cross Bar forms the interconnect network between data ports and the Meshes in Stallion and is the primary means of creating deep pipelines. It has 22 inputs and 38 outputs and supports 16-bit wide data paths. Of the 22 inputs to the crossbar, six come from the data ports and eight come from each of the two meshes; and of the 38 outputs, six are sent to the data ports and sixteen go to each mesh. The crossbar provides full connectivity among the data ports and components in the meshes.

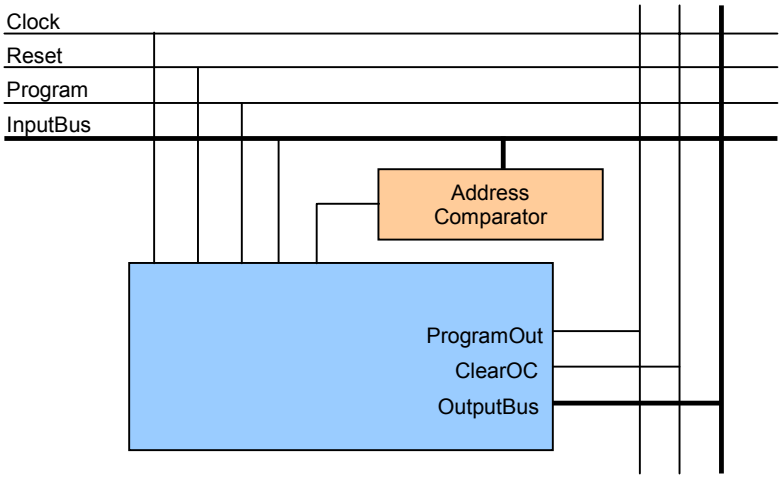

Figure 3-2 Structure of Crossbar Node

At the intersection of each row and column in the crossbar is an XBarNode which consists of a register to latch data that comes in through the row signal lines, an address comparator to check if the information is meant for this XBarNode and a finite state machine to control the operation of the XBarNode. A stream entering a row uses unique address and output of the address comparator specifies if signals on *InputBus* from the row should be transmitted on *OutputBus* in the column.

3.4 Mesh

Stallion has two separate computational meshes. Each mesh is organized as an 8 x 4 matrix consisting of 30 processing units called the *Interconnected Functional Units (IFU)* and two multipliers. The multipliers are placed on the top left and right corners of each mesh. Inputs to the mesh arrive from the outputs of the cross bar and mesh outputs are sent back to Cross

Bar.

Within the mesh, *local* and *skip* buses are used to transmit data among the IFUs. Each IFU can send data to its four nearest neighbors using the local bus and to distant IFUs using the skip bus. Skip bus provides a convenient way of fast data transfer between far-off IFUs.

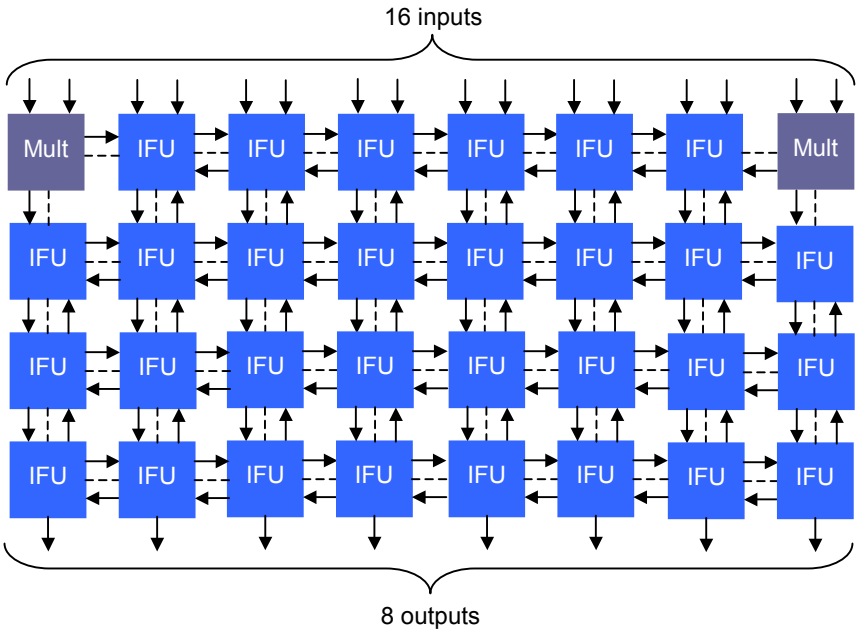

Figure 3-3 Mesh Topology

The multipliers do not have skip bus connectivity. The inputs to multipliers come from the cross bar and outputs are sent to two nearest IFUs via the local bus connectivity.

Structurally, the mesh has a very regular topology and contains only two types of components as discussed earlier.

3.5 Functional Unit

Functional Unit (FU) in Stallion forms the basic data processing unit. It has 16-bit left and right input registers, which receive inputs from the interconnected functional unit. It is possible to source these operands from any of the four local and skip bus values.

As illustrated in the figure, left operand passes through the barrel shifter and is sent to the arithmetic and logic unit. The right operand is sent to ALU, the conditional unit and a delay block. The arithmetic and logic unit performs various operations on the two operands. Conditional unit can make comparisons based on a conditional flag and choose one of the two inputs as its output. The delay blocks are used for pipeline synchronization and aligning execution path lengths between two or more streams. The right operand can also be directly passed on to the auxiliary output with or without introducing a delay.

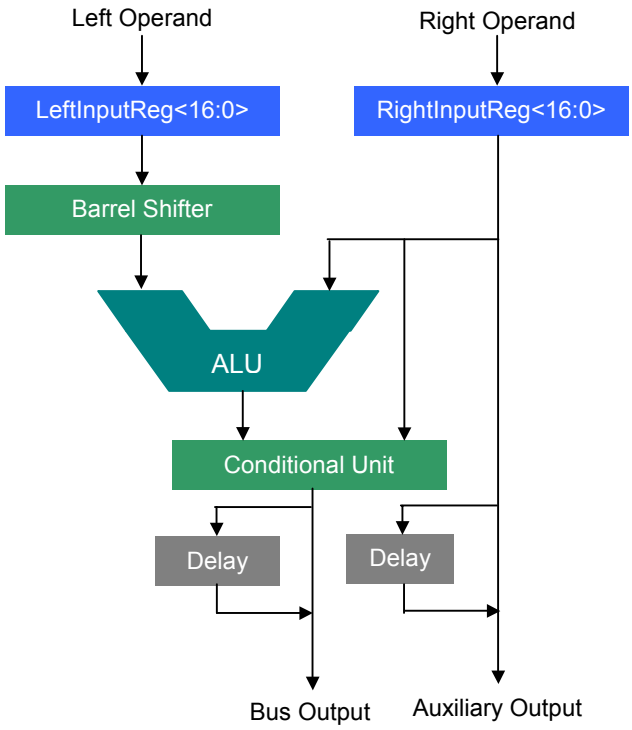

Figure 3-4 Functional Unit

The design of the FU allows limited multiplication operations also. It can be used for fixed coefficient multiplications using shift-and-add operations. The shift-and-add operations can be performed in a single clock cycle. By cascading multiple FUs and performing constant coefficient multiplications in this manner makes the mapping of digital filters on Stallion attractive.

The main components of an FU are two input registers, a barrel shifter, conditional unit logic and delay registers.

3.6 Interconnected Functional Unit

Interconnected Functional unit (IFU) is the building block of meshes. It consists of a Functional Unit surrounded by control and data buses to provide connectivity among the neighboring IFUs.

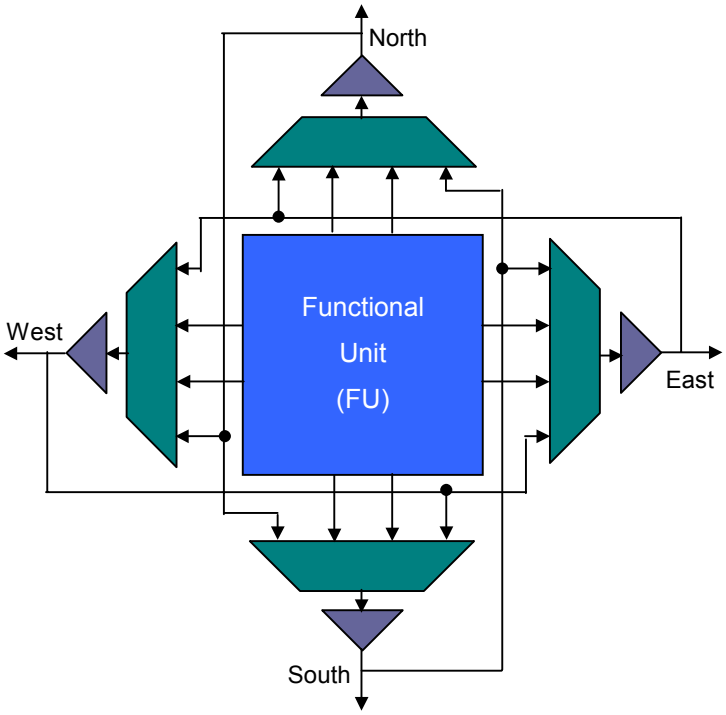

Figure 3-5 IFU Connectivity for Skip Buses

Each of the four skip buses can select direction of signal flow and act as an input or an output. If the skip bus is configured as an output, then it can output any one from the FU output, the FU auxiliary output, value of the skip bus directly above or the value of skip bus at its right side. As seen in the figure, skip buses in an IFU provide connectivity only to the IFU located directly opposite to it and to the one on its right side. However, the outputs from an FU can be sent to any of the four neighboring IFUs.

Apart from the skip bus connection, there are two local bus connections on each side of the IFU. These two connections, with the neighboring IFU, are the local output and the local input for the Functional Unit. Thus, on each side of IFU, there are three bus I/O signals comprising of the skip bus and the local buses. Apart from these, there are three general-purpose flags that are routed using the local and skip buses.

All the functionality of an IFU described above is mainly implemented using the FU block, bus multiplexers and tri-state buffers.

3.7 Multiplier

Stallion processor contains four multipliers that are located in top left and top right corners in each mesh. Designed by Tsuang-Hen Yang [9], this pipelined multiplier has also been used in Colt. It accepts two 16-bit inputs and produces a 32-bit output in two clock cycles. The inputs to multipliers come from the crossbar and the outputs are sent to two nearest IFUs.

Multiplier design can be broken down into smaller units called multiplier cells. There is an array of such cells in the multiplier. Apart from that, there are several registers and half adders in the multiplier logic. Details about the logic design of this pipelined multiplier can be found in [9].

Chapter 4

Design Methodology

The VLSI design flow has taken the form of a standard owing to the complexity of the task and the high costs of even one mistake. Most VLSI designs are a result of strict regimen of set design practices that have been laid out after years of experience and research. The computer-aided design tools have been designed to fit into the existing practices. The choice of a particular design methodology is based on the applications of the design and frequently, the nature of the design itself. The Stallion processor, being among the forerunners of new CCM architectures, adopts a full-custom physical design methodology. This approach is followed when the designers want freedom in defining all possible details from system-level down-to the transistor level. Many a times, the standard libraries are not suitable for the purpose. Stringent performance requirements also drive full-custom design flow. This chapter focuses on full-custom physical design flow, the practices followed for Stallion, associated CAD tools and how they fit in the Stallion full-custom physical design flow.

4.1 Full Custom Physical Design

The full custom design process is based on a "correct-by-construction" approach. This approach relies on the fact that the designer has finalized details of the design on a transistor-by-transistor basis. Since all the details have been taken care of, it is implicitly guaranteed that the chip design is going to be correct as long as the net-list extracted from mask data matches with the schematics.

In a typical design, following the functional specifications and system level design, all lower

level modules are designed. Before the physical design process is started, the design is frozen in form of either schematics or some type of structural description in a high level HDL. The schematics are captured in a CAD tool that has a reliable interface to the physical design tools that are going to be used for creation of layouts.

At this point, the IC design process is mainly concerned with creating layouts in chosen fabrication technology and making sure that there are no design rule errors and no net-list mismatches compared to schematics. Layout issues like power distribution scheme, parasitic capacitances, etc. are also taken care of in this phase of physical design.

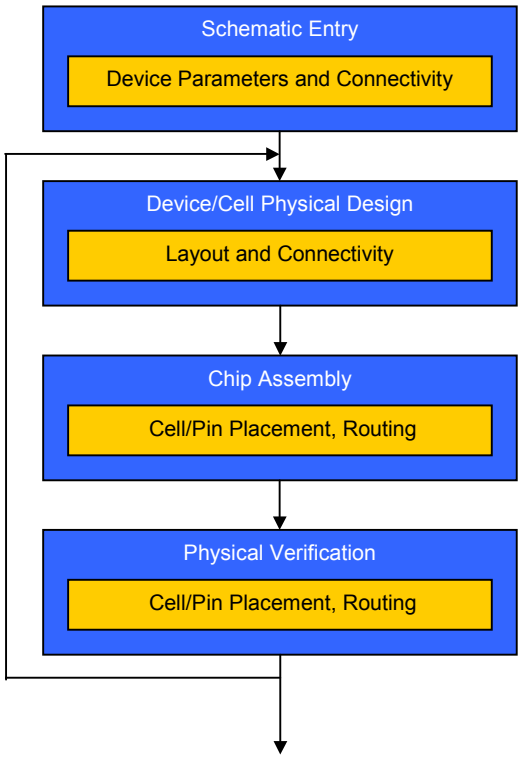

Figure 4-1 Typical Design Flow of Physical IC Development

As illustrated above, physical design mainly consists of creating layouts in accordance with the design rules. Layout work progresses to higher levels of design hierarchy until the top-level chip is assembled. At every step, checks for compliance with design rules are performed. This is essentially an iterative process. The layouts are edited until the design rules are met and structural equivalence with schematics is established.

4.2 Physical Design approach for Stallion

At the time of commencing the work discussed in this thesis, most of the Stallion structural design was available in form of schematics. The remaining parts of the design were available as functional description in Verilog. The schematics systematically partitioned the design into a multi-tier hierarchy. This hierarchical breakup made the task of creating layouts convenient and reduced the complexity of individual cells as well. The hierarchy of schematics was exactly replicated in layouts. In retrospect, this decision, made early in design cycle, considerably simplified the tasks of design and verification later on. The parts of Stallion design that were available in Verilog form were also converted into structural description. The components used in the structural description were available in Stallion design database. This structural description was then translated into schematic representation and merged with rest of the design.

Layout representations for the components (or cells) that were lowest in design hierarchy were created first of all. As has been mentioned earlier, all these cells were subjected to checks for compliance with geometrical design rules and for equivalence with corresponding cells in the schematics. Terminal/Pin information for each of these cells was also entered in the layout. This was required by the design tools to automatically identify terminals and connectivity information to match with the schematics.

At the time of starting physical design of Stallion, the layout views of some of the cells were already available as a result of work done by others in the past. However, the fabrication technology on which those cells were based had become old. In order to use these cells, it was necessary to migrate each cell from older technology to the newer one. So, the entire available library was exported in GDSII format and imported back using parameters for the new fabrication technology (TSMC25M). The back and forth translation was used to remove all connectivity information, all terminal/pin information, text and marker layer geometries and display characteristics that were based on older technology. These were overwritten by parameters of the technology file that was going to be used now. The connectivity information was recreated by extracting net-list from the new layouts. Then, the size of each of these cells was scaled down to match the design rule guidelines for λ=0.15. However, this

process of technology migration has some inherent limitations. These limitations introduced several design rule errors in the layout. All cells that had small number of errors were corrected and saved in the new library of cells being made for Stallion. The ones with large number of errors were discarded. Each of the corrected cells was also checked for errors using Design Rule Checker (DRC) and verified against schematics using Layout versus Schematic Verifier Tool (LVS). The remaining cells were then manually laid out.

After all the cells in bottom level of design hierarchy were created, efforts were expended to go up the hierarchy and re-use these cells. The higher-level cells needed one or more of lower level cells. This is a tedious task if done manually. However, if the names of components and terminals in schematics and layouts are kept identical, the CAD tools can easily create correspondence between schematics and layouts. Thus, the process of making higher-level cells can be partially automated. The CAD tools were used to automatically pick the components needed for higher-level cells, create pins in specified metal layers and also import connectivity information from the schematics. Following the automated pickup of component cells and extraction of connectivity information from schematic, the placement of cells was manually performed. The layout was then exported to another design tool to create metal routes from the connectivity information already available in the design. Every component created in this manner was then subjected to usual verification tests to make sure that it is equivalent to the schematic and follows all design rules.

At the top level of design, it is imperative that all blocks fit in a rectangle with adequate room for top level routing of signals and power rails. This needs careful planning early in the design cycle to ensure geometrical alignment at the final level in design hierarchy. Strategies for floor planning and assessment of routing resources are also necessary for efficient utilization of silicon real estate.

The placement of pins on lower level cells governs the placement of pins on top level. Therefore, the placement of the pins of lower lever blocks must be done with the location of top-level pins in mind. Moreover, to ensure logical grouping of pins in the chip package, it is necessary to identify pin placements beforehand and perform lower level placements

accordingly. The pins in Stallion were located so that routing between blocks is minimized. In the top level in Stallion design, the pins were grouped by according to data port terminals and were uniformly distributed on the four sides of the die.

At the completion of top-level of layout of Stallion, pads were added. The Stallion layout has 180 pads, 44 on each of top and bottom sides and 46 on each of left and right sides of the layout. The size of die and the number of I/O pins govern the decision of package to be used. The Stallion design has been packaged in a PGA181 ceramic package.

4.3 CAD Tools

Most of the tools used for Stallion prototyping belong to the Cadence Virtuoso family. This section briefly mentions each tool and the functionality in each that was used to create Stallion processor. More details on the use of these tools can be found in [11].

4.3.1 ICFB

ICFB is the front-end graphical user interface that is used to access the paraphernalia of Cadence VLSI design tools.

4.3.2 Schematic Composer

This tool is used to create schematics and to set properties of the devices used. It integrates easily with other tools that access schematics and extract connectivity information from them.

4.3.3 Layout Editor

This tool is used to draw layout geometries and devices used in a typical chip. The Layout Editor interface is also used to access many related tools like the net-list extraction tool, DRC and LVS. Additional menus provided by the NCSU Design Kit are also accessed from the Layout Editor GUI.

4.3.4 Layout XL (Layout Accelerator)

Layout XL is used to automatically pick and place components required to create a higher-level cell. It reads required information about the components and their interconnections

from the schematics. Templates can be used to specify exact placement of individual cells and pins of the cell that is being created. This tool also interfaces to IC Craftsman and Assura/Diva Physical Verification Tools.

4.3.5 IC Craftsman

IC Craftsman is the tool that was used to perform automated placement and routing. Design data is imported from Layout XL and exported back after placement and routing is complete.

4.3.6 Assura/Diva Verification Tools

This is the set of tools used for physical verification. Design Rule Checker (DRC) performs checks to verify that all λ based design rules are complied to. Layout versus Schematic Verifier tools performs equivalence checks between layout and schematics. Both these tools are accessible from ICFB and from the Layout Editor Window.

4.3.7 Verilog In

To translate structural Verilog description of modules to schematics, Verilog In tool is used. It is accessible from within the ICFB interface.

4.3.8 Scale

In order to scale layout data from one technology to another, from 0.5 to 0.25 micron for instance, this tool is used. It is accessible from command line only.

4.3.9 Stream In/Out

The tools – Stream In and Steam Out – belong to the set of tools meant for translation of physical design data from one format to another. For Stallion, Cadence Database format was converted to GDSII format for tape out to foundry.

4.3.10 NCSU Cadence Design Kit

North Carolina State University has developed a design kit [12] for Cadence based physical design tasks. It consists of technology files, standard cell libraries, design rule files, extra functionality for layout and schematic editing etc. A part of the NCSU design kit, called the p2m converter, is used to convert image file from JPEG to a Cadence database format. It is useful to put graphical object on silicon.

4.3.11 MOSISCRC

This is a program supplied by MOSIS [10] to calculate CRC Checksum of the mask files in GDSII or CIF format that are uploaded to MOSIS servers. CRC checksum is used to ensure correctness of file transfer over computer networks.

4.4 Stallion Design and CAD Tools

As mentioned above, several different tools were used to create layouts for Stallion processor at various stages in design cycle. It is important to be aware of the relationship among these tools and the place of each tool in a typical custom physical design flow.

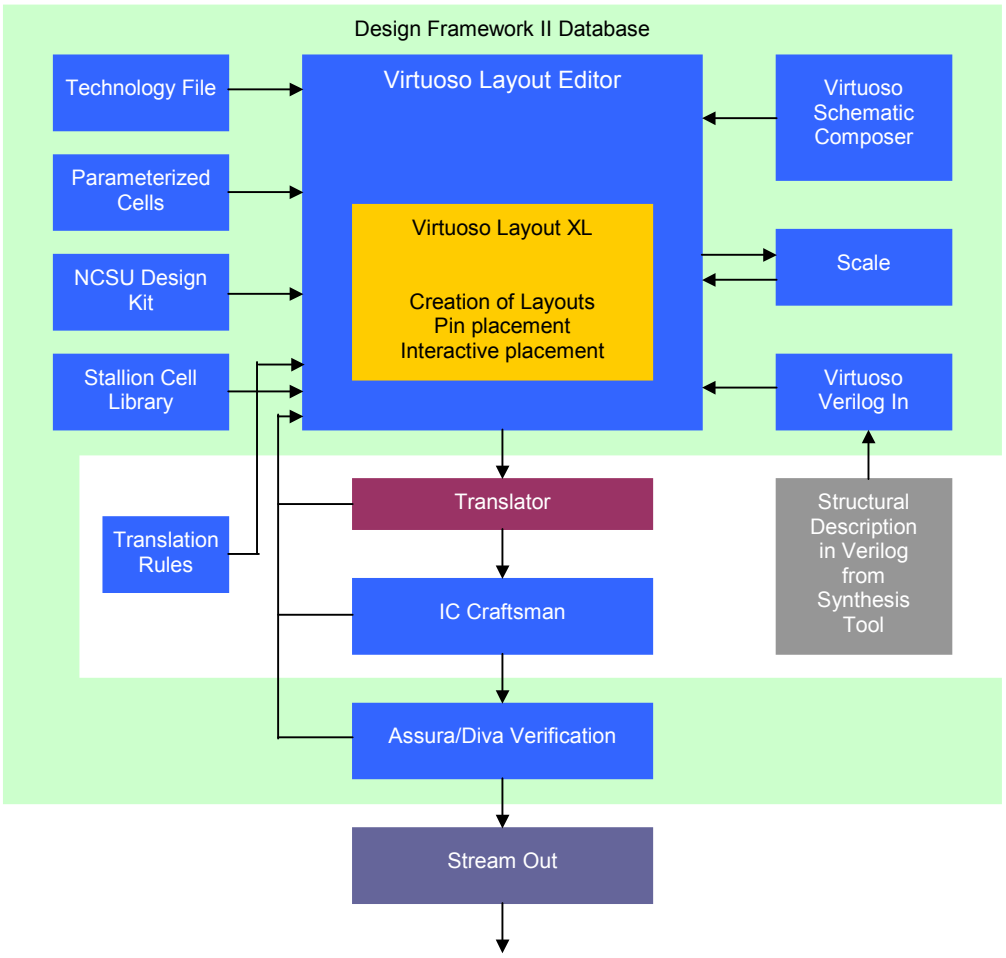

Figure 4-2 Relationship of CAD tools in Stallion design

As is evident from the figure above, physical design of a VLSI chip is an outcome of the interplay of a multiplicity of design tools. These tools simplify the task of a physical

designer considerably. However, it also takes considerable effort to make sure that the files in the design are managed properly. The size and complexity of design files keeps getting bigger as design advances up in hierarchy. Any changes made late in the design cycle need considerable time and effort to be incorporated. Similarly, bugs found later in design also have adverse impact on the design schedule.

Chapter 5

Stallion Floor Plan and Layout

As has been discussed in the previous chapter, Stallion layouts were made using a range of CAD tools. In this chapter, the process of custom crafting each component of Stallion layout is described. The details of Stallion design hierarchy, floor plan and layout are also provided.

5.1 Design Hierarchy

Stallion design is represented as a multi-level hierarchy. Keeping the design in a hierarchical format simplifies the creation of layouts. It also improves efficiency by re-using the lower level layout cells in the cells at higher levels.

There are 166 different cells in Stallion design hierarchy. In all, these 166 cells have about 648,000 transistors. The complete design hierarchy of Stallion processor is illustrated in Appendix A.

5.2 Stallion Library Cell Layout

The cells at the lowest level in design hierarchy were created using Virtuoso Layout Editor. The schematic of each cell was used to determine connectivity and parameters of transistors. The number of metal layers used for routing was limited to two. All terminals in the cell layout were named exactly as named in the schematic to facilitate design automation. The terminals in each cell were also marked by drawing pins in the layout.

In entire Stallion layout design, scalable lambda based (λ=0.15 micron) design rules were followed. After a cell passed these design rules, the net-list was extracted and subjected to LVS equivalence check against the net-list extracted from schematics. In case of errors, the cells underwent iterations of corrections and verification tests until the layout was free of design and connectivity errors.

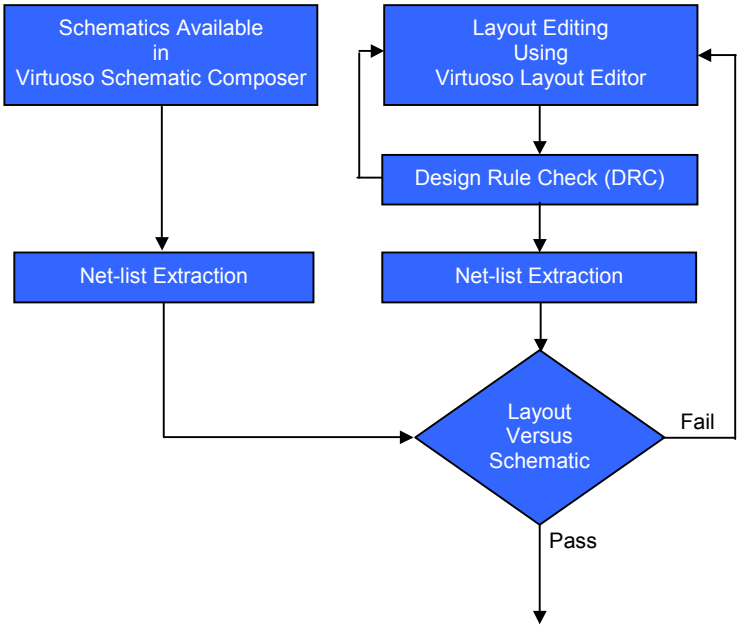

Figure 5-1 Design of a cell at lowest level in Stallion Hierarchy

Figure 5-1 outlines the scheme of operations that are performed to create the layout of a typical cell in Stallion. The details of each step are available in [11].

5.3 Creation of Higher Level Cells

Physical design for higher levels cells is similar to the design of cells at lower levels in design hierarchy. The main difference between the two methods is in the use of automatic routing and, sometimes, automatic placement. Another important difference is the routing of power and ground rails. As the cells grew in size, the width of power and ground rails was increased. At the top-most cell, the width of power rails was 120λ. The use of poly-silicon in long routes was avoided to minimize resistance. The dimensions and aspect ratio of cells was also controlled to make the top level design fit into the projected floor plan.

Virtuoso Custom Router (IC Craftsman) was utilized for automating the placement and routing of bigger cells. In IC Craftsman, the routing algorithms are designed to perform very efficient routing at the level of chip assembly. However, the routing for smaller cells is not as efficient. This caused wastage of silicon area and resulted in layouts that could have been better at silicon utilization. Nevertheless, by use of certain options in the router, attempts to mitigate this limitation were made.

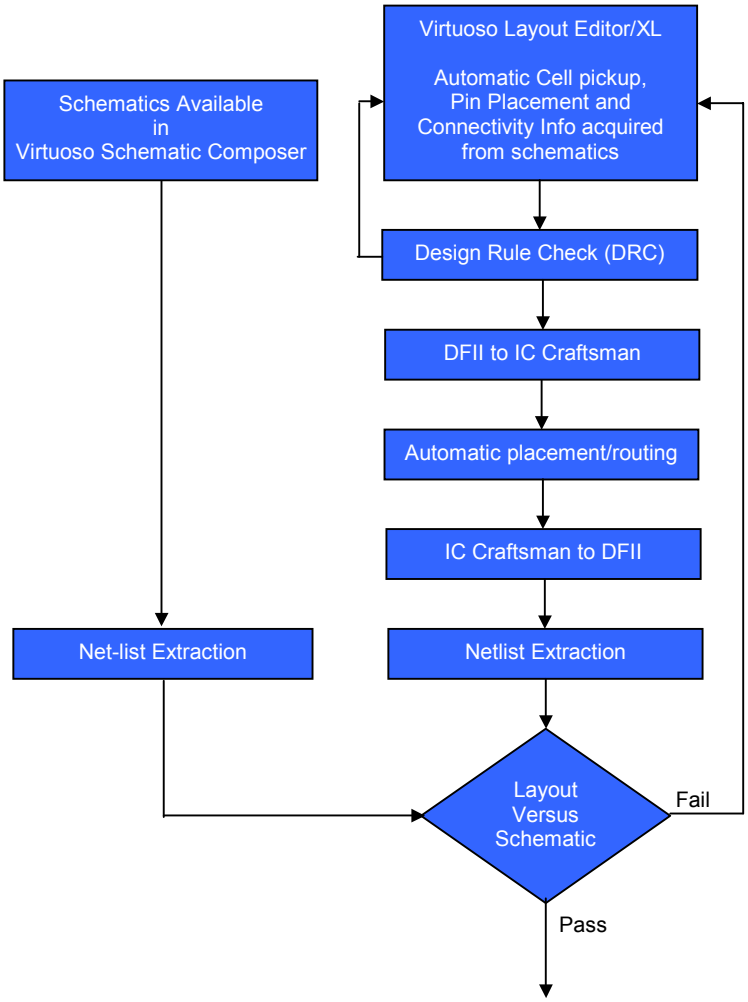

Figure 5-2 Design of higher level cells in Stallion

In Figure 5-2, the design flow used for most of the higher level cells is illustrated. The ability of Layout XL to retrieve connectivity information from schematics was probably the most time saving aspect of this design flow. Without automatic pick-up of components and connectivity information from Schematics, the complexity of prototyping Stallion would have increased manifold.

5.4 Multiplier Layout

Each of the four multipliers in Stallion is a two stage pipelined array multiplier. It was designed by Tsuang-Hen Yang for Colt and was later adopted for Stallion. However, its layout has been re-done to make it compatible with the current fabrication technology and meet the size constraints.

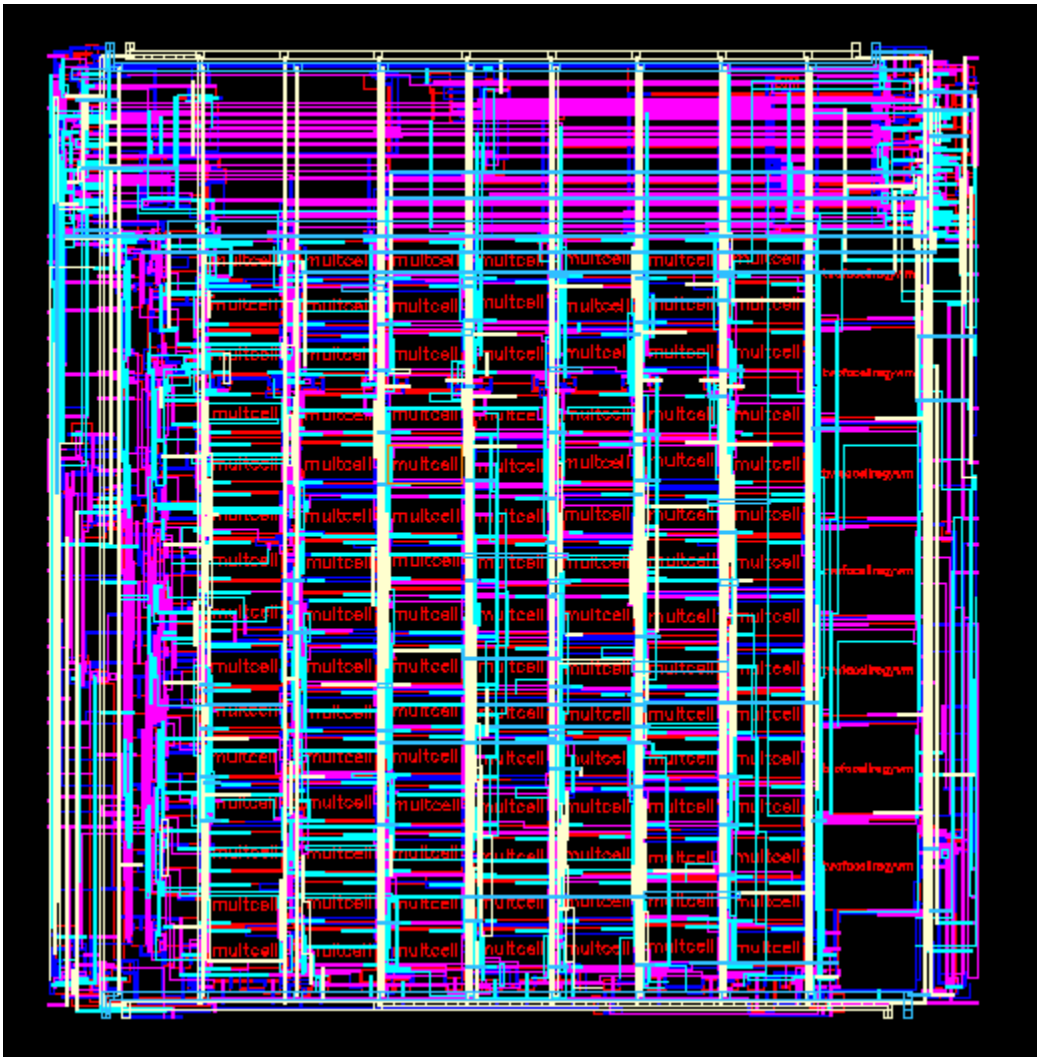

Figure 5-3 Layout of the Multiplier

The main components of the multiplier are an array of *MultCell* blocks and several Full Adders. The inputs to the multiplier are fed from the left and the output is available on the

right edge. However, it was rotated 90˚ at the time of embedding in the mesh so that the inputs are at the top edge of Mesh and outputs at the bottom. The routing in Mesh connects the outputs from bottom edge of the multiplier to the neighboring IFU on its bottom and on the left or right depending on where the multiplier is located in the Mesh.

5.5 Layout of Functional Unit

The Functional Unit is one of the important cells as its size is a crucial determinant of the size of Stallion die. The FU floor plan was intuitively guided by the functionality of its constituent blocks. The registers near the top edge are used to latch the left and right input operands. The delay register and the barrel shifter are placed just below these registers. The ALU is located towards the center in left half of the floor plan. Near the bottom edge of the Functional Unit, there are more registers for introducing delay in output of FU. The registers on the bottom left side are used to store configuration information. The floor plan of FU is as shown in Figure 5-4.

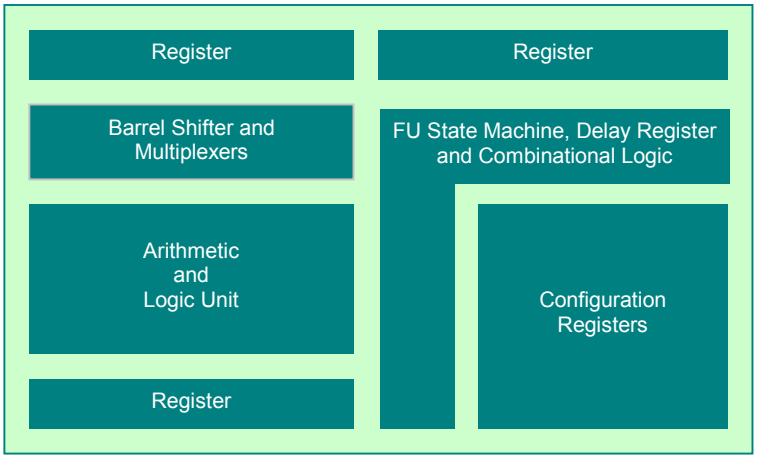

Figure 5-4 Floor Plan of the Functional Unit

Some components of the functional unit were hand laid out. The main components among these are the barrel shifter, combinational logic blocks, registers and multiplexers. Other components – ALU and FU State Machine were created using automatic routing. All these components can be seen in the Functional Unit layout shown in Figure 5-5.

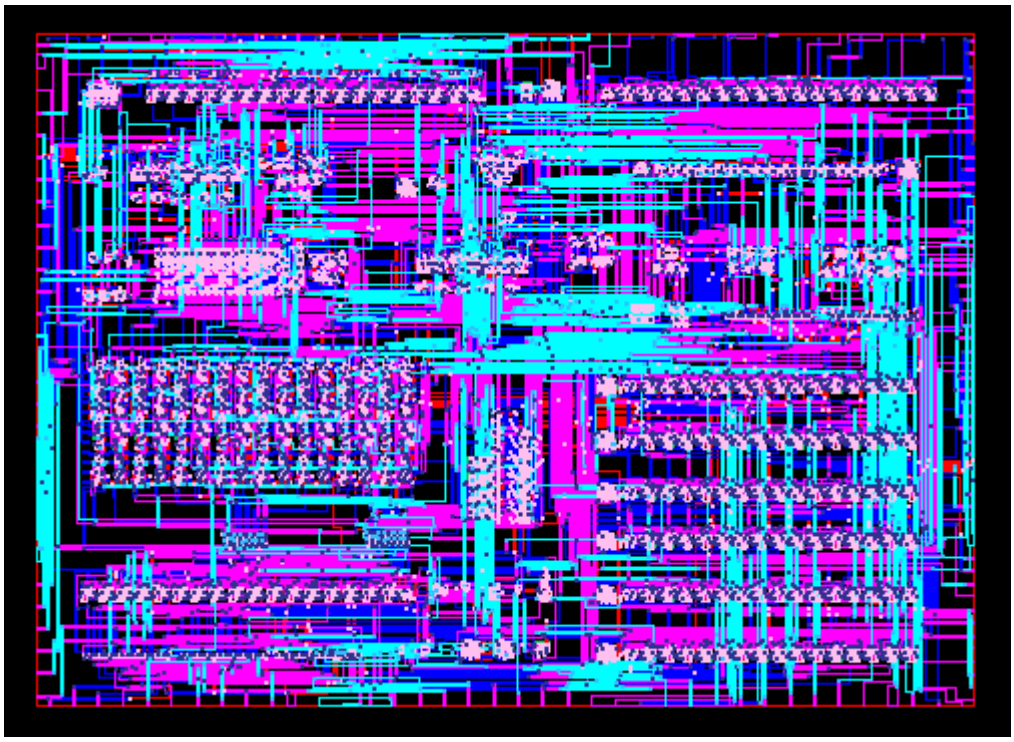

Figure 5-5 Layout of the Functional Unit

5.6 IFU layout

The Interconnected Functional Unit performs the task of transmitting signals to and from the Functional Unit in various directions according to its configuration. In order to implement this functionality, each side of the IFU has six buses – two skip buses, two local input buses and two local output buses. These buses are multiplexed to generate the input and output signals for the FU embedded inside the IFU. The terminals on each side of the IFU tap from three of these buses.

Figure 5-6 shows the floor plan of the Interconnected functional unit. The functional unit is the largest cell in IFU. Other cells, like the bus multiplexers, combination logic and configuration registers are distributed around the FU. On the top right corner, a rectangular region has been left to accommodate the address decoder. This allowed easy integration of address decoder cells in Mesh layout. A script in Layout XL was created to automate the placement of pins. Since the number of components in IFU is quite large, the cells were automatic placed. Following automatic placement, the cells were manually relocated to

improve alignment and spacing. Several iterations of routing and placement corrections had to be done before the layout of IFU achieved 100% routing with zero routing overlaps or shorts. The IFU layout was then imported back into Layout XL and routine verification tests were performed.

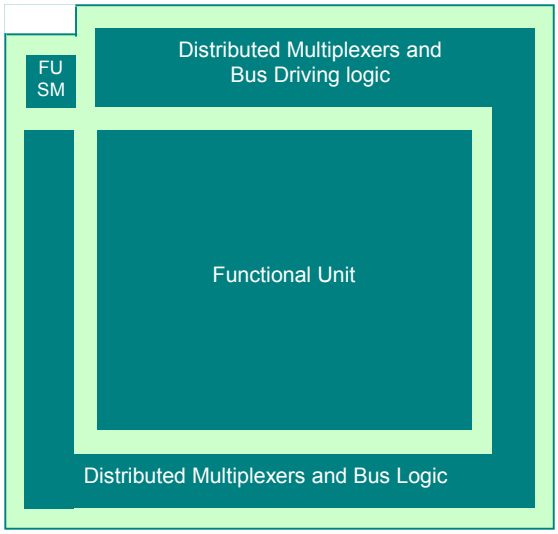

Figure 5-6 Floor plan of Interconnected Functional Unit

The layout of Interconnected Functional Unit is shown in Figure 5-7.

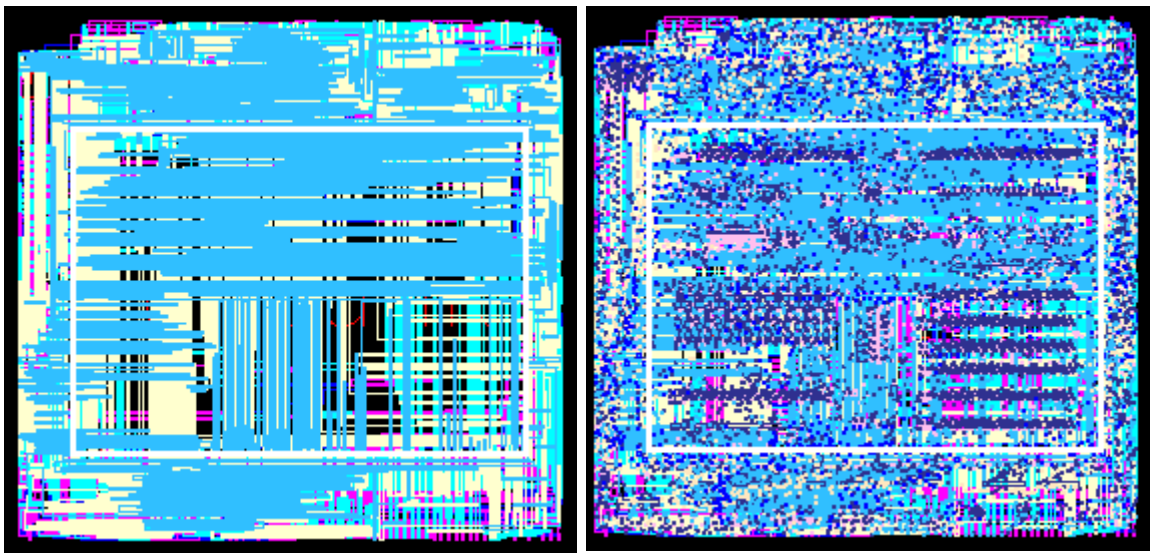

Figure 5-7 IFU Layout – single layer and all layer in layout

5.7 Mesh Layout

The Mesh in Stallion is an array of 4x8 Interconnected Functional Units with the top left and top right elements being the multipliers. The multiplier in Stallion is approximately the same size as the IFU. The address decoders in the mesh are placed in the rectangular spaces in top left corners of the IFUs. The inputs to the Mesh are fed at the top edge of Mesh and the outputs are tapped from the bottom edge. The Stallion processor has two identical meshes. These meshes are referred to as Mesh A and Mesh B.

The cells in each Mesh were manually placed. Adequate inter-IFU spacing was left to allow routing of power rails. The input and output pins were placed at the top and bottom edge of Mesh respectively. After the pins and cells were placed, the layout was exported to IC Craftsman and automatic routing of signals was performed. The Mesh layout was imported into Layout XL and the routine tests for DRC and LVS were performed to verify the layout.

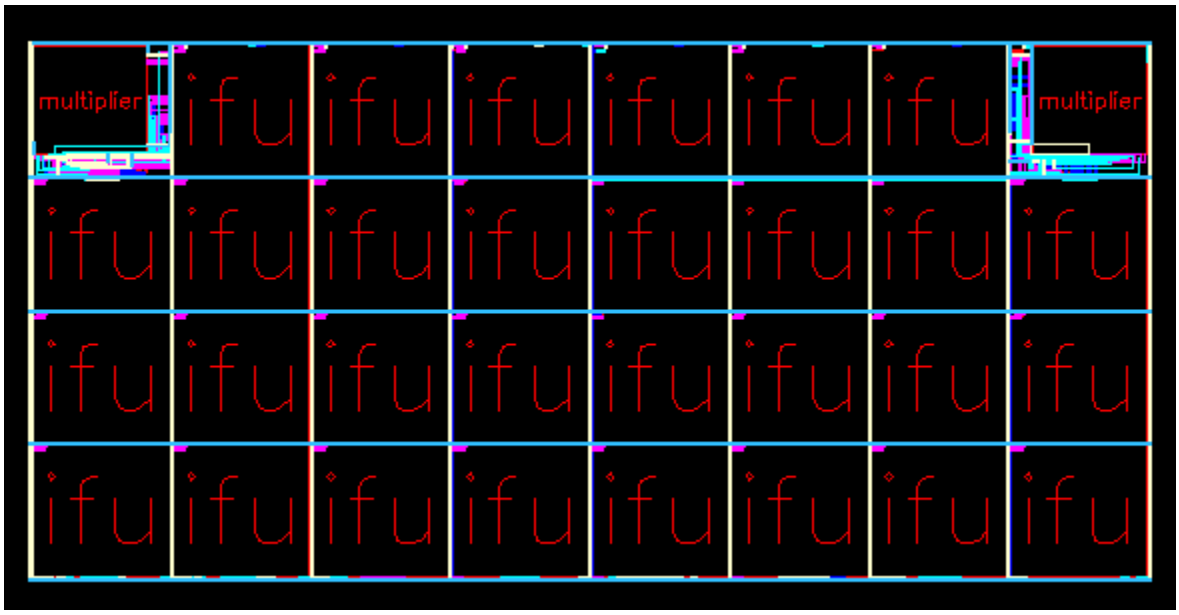

Figure 5-8 Layout of Mesh

As seen in Figure 5-8, two multipliers are located at the top corners. The routing in Mesh is limited to the channels between IFUs and Multipliers. The address decoders can also be seen on the corner of each IFU in the figure.

5.8 Cross Bar Layout

The Cross Bar in Stallion provides connectivity between the Meshes and the data ports. For the sake of simplifying design and layout, the 22x38 structure of the Cross Bar is divided into two smaller units. Each of these units, named XBarA and XBarB, is a 22x19 array of nodes. Of the 19 outputs from each XBar, three are sent to three data ports and sixteen to one of the Meshes. Similarly, the inputs to XBarA and XBarB come respectively from Mesh A and Mesh B.

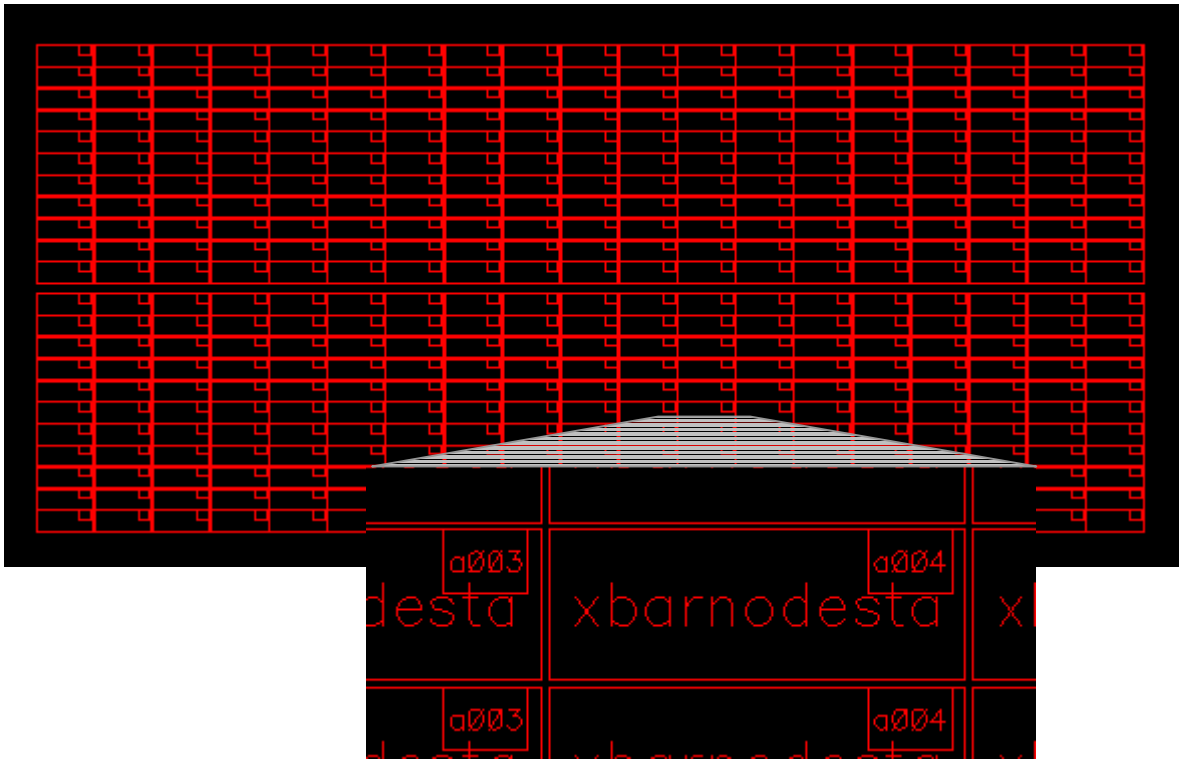

Figure 5-9 Floor plan of XBarA

Each node in the XBar has an address decoder on its top right corner as can be seen in the Figure 5-9 inset. The placement of pins on each XBar has been done to facilitate connectivity of signals between the Meshes and the Data ports such that connecting routes do not have to traverse the entire length of the chip. The outputs from XBarA shown in Figure 5-9 are available on the top edge and inputs are fed at the right edge. The layout of XBarB is a mirror view of XBarA such that the inputs are on the bottom edge and outputs on the left edge. The aspect ratio of the node Cells in XBar layout was tweaked so that the dimensions of Cross Bar would match its dimensions in the projected floor plan of Stallion chip.

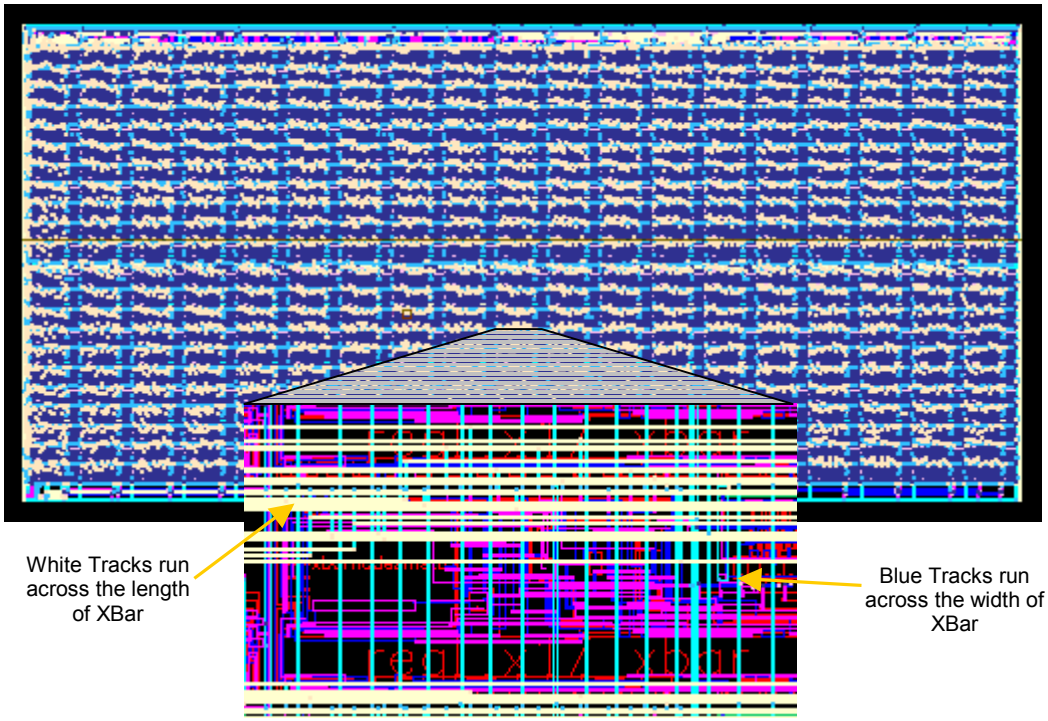

Figure 5-10 Layout of XBarB

Even though the XBar structure has regular grid like placement of cells, there is a large number of routes that span across the length and breadth of XBar unlike the routes in the Mesh. Unless a systematic sequence for routing the nets is created, it is easy to get the router stuck in its optimization algorithms. The router is unable to complete routing in such a situation. By heuristic analysis of the net-list, a sequence for routing the nets was developed to accomplish successful automatic routing. Even slight changes in sequence radically altered router performance. The order of routing the nets was tinkered with and finally, IC Craftsman was able to route all nets in the XBar.

5.9 Data Ports

Data ports provide the only way to communicate to a Stallion chip. The six data ports are placed in groups of three in two diagonally opposite corners on the die. This placement provides easy access to the I/O pads and to the Mesh and Cross Bar terminals.

Each data port consists of a Data Port State Machine, flip-flops, latches and combinational logic. Figure 5-11 shows the layout of a Data Port in Stallion.

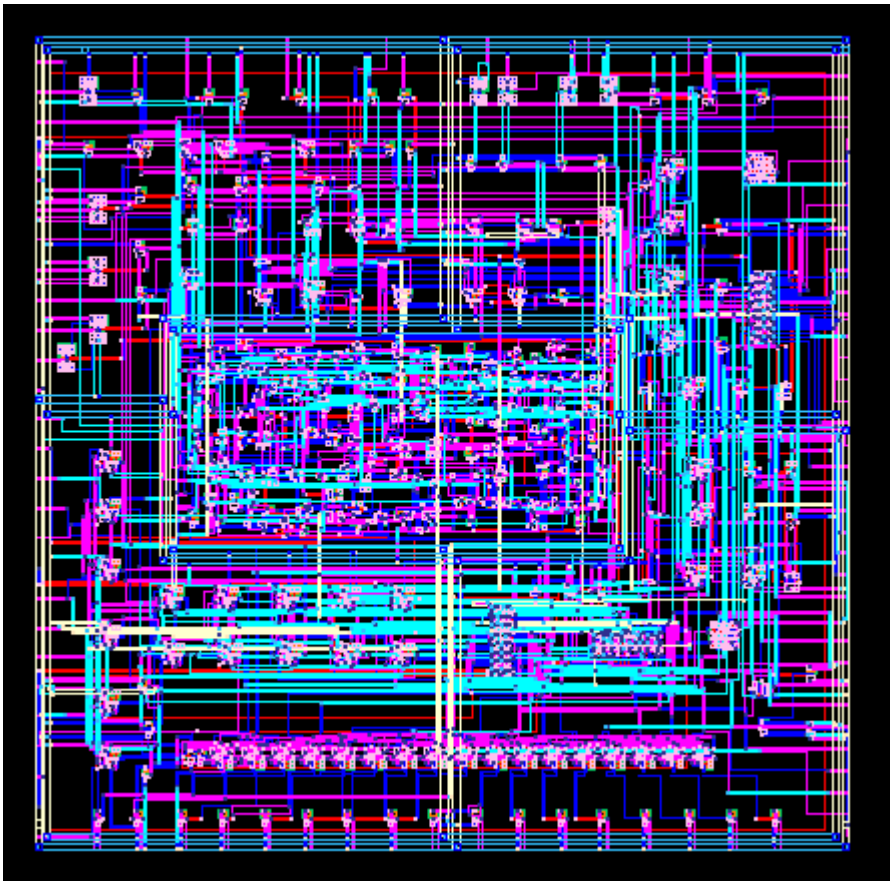

Figure 5-11 Data Port Layout

The design of Data Port State Machine was inherited in the form of Verilog functional description instead of the usual schematic form for rest of the Stallion's structural design. A significantly different approach had to be implemented to create layout of each Data Port State Machine. The functional description of Data Port State Machine was input to *Design Compiler*, a logic synthesis tool. In order to make sure that only the cells available in Stallion library are used, a small library was created using *Library Compiler*. This library contained all the gates available in Stallion database. Using this library, a structural Verilog module was generated by *Design Compiler*. All the gates that were instantiated in this module had a symbol and layout view available in Stallion design database. Then, the *Verilog In* tool was used to translate the Verilog description into schematic form. In this conversion, the schematic view is created using the symbol views of each of the gates used in the design. From here onwards, layout of the Data Port State Machine was created in the essentially the same way as for any other cell using *Layout XL* and *IC Craftsman*.

The VLSI CAD tools have several different ways to represent design data. At the same time, there is a plethora of options to migrate between these representations. Sometimes, this increases the complexity of what could otherwise be a simple task. In the case of Data Port State Machine, a significant amount of effort was invested before the technique described above actually worked. Since the exact die area needed by a data port could not be correctly assessed in time, a very optimistic figure was used while creating the Stallion floor plan. It was later found that the data ports occupied less area than what was estimated. This led to wastage of some area on the die. Nevertheless, timely completion of the physical design of Stallion processor compensated for it.

5.10 I/O Pads

There are 180 I/O pins in Stallion. Each of these pins is connected to the die via an I/O Pad. The pads that Stallion design uses have been sourced from the VLSI CAD Research group at NCSU. They used the I/O pad library available from MOSIS and scaled down the size to match with the lambda based design rules. The I/O Pads form a boundary on the Stallion die. There are several types of I/O Pads that are used in Stallion. The functionality of the pads used in Stallion is tabulated below.

###	Type of Pad	Functionality
1	Padvdd	Connection to VDD
2	Padgnd	Connection to GND
3	Paninc	Input pad
4	Padout	Output pad
5	Padbidir	Bi-directional Pad
6	Padnocon	Unconnected pad
7	Padlessspacer	Pad used to fill extra space on perimeter of the die
8	Padlesscorner	Pad used on the corners of the die

Table 5-1 I/O Pads and their functionality

In order to verify the functionality of pads, the input, output and bi-directional pads were simulated using SPICE simulator. The simulation was also necessary to ascertain whether the pads terminals required inverted or actual signal values.

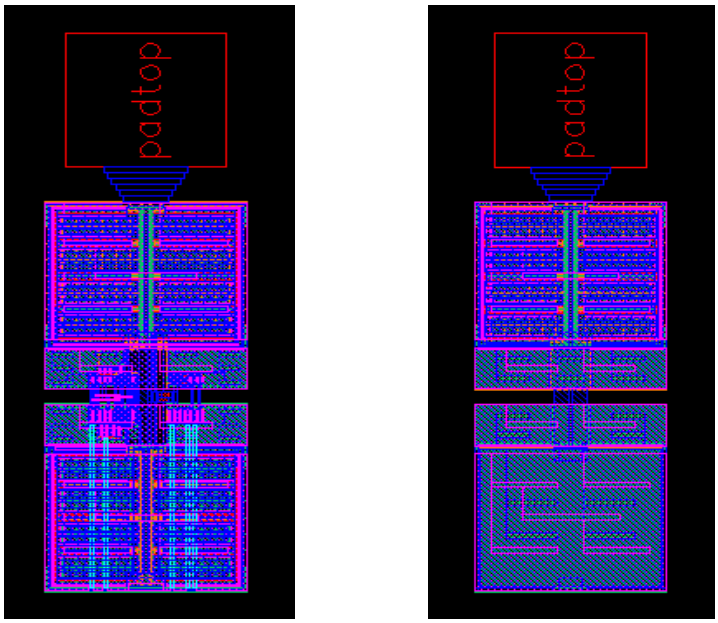

Figure 5-12 Layout of *padout* and *padgnd*

The pads were arranged around the boundary of Stallion layout. The location of input, output and bi-directional pads was according to the location of these pins on the Stallion die. Since the router cannot identify terminals on the pads unless the name of each and every terminal is specified by putting labels and pins on the pad layouts, a work around was devised to make accurate automatic routing possible. It is possible to determine the exact co-ordinates of the point where the metal routes should terminate in the pads. By calculating the exact locations, a script was made for *Layout XL* that placed the top-level pins at the specified locations. With the pins correctly located, the completely routed design just had to be placed on top of the ring of I/O pads to complete the Stallion layout.

5.11 Inserting Graphics in Layout

The NCSU design kit also provides the functionality to convert graphic files in JPEG format into equivalent layout views. A JPEG image to be placed in layout is first converted into a black and white image. Then, each black pixel is replaced by a square drawn in a metal

layer. The size of metal layer is in accordance to the design rule requirements. Using this functionality in the *p2m* tool, few images of the people who worked on Stallion layouts and the logos of affiliated organizations were embedded on Stallion die. One such image is shown in Figure 5-13.

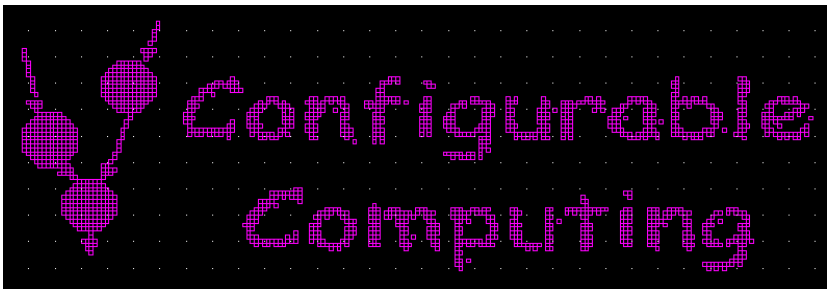

Figure 5-13 Logo of CCM Lab embedded on Stallion die

5.12 Stallion Layout

After all the blocks in Stallion design were completely laid out and verified, top-level chip was assembled. The area required for top level routing for signals and VDD/GND rails was estimated and the blocks were placed accordingly. As mentioned before, the placement of pins was done using a script so that the pins are correctly aligned with the pins in pads when the pad ring is merged with rest of the layout.

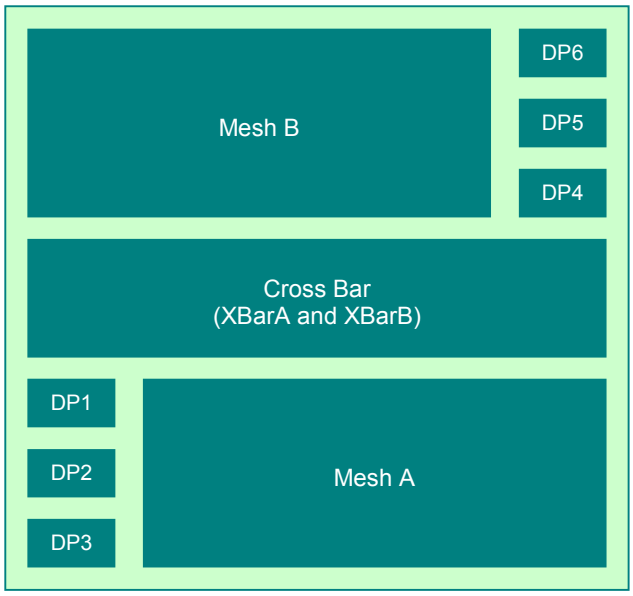

Figure 5-14 Floor Plan of Stallion Processor

As shown in Figure 5-14 , the Stallion floor plan has the Cross Bar in the middle with one Mesh on either side. The data ports are placed in two diagonally opposite corners. Apart from these blocks, there are address decoders for Data Ports and part of the logic that implements pipeline-stalling mechanism.

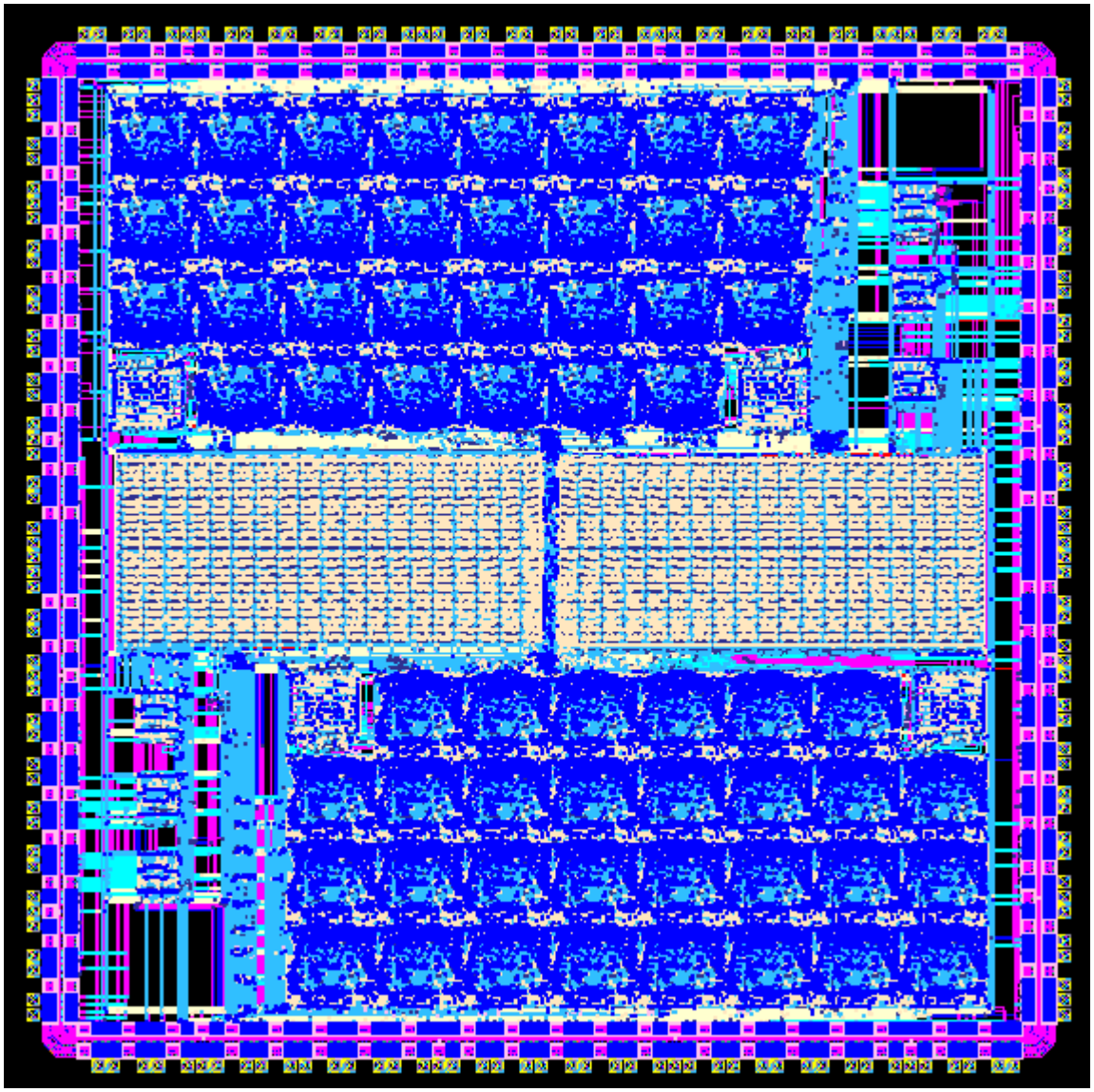

Figure 5-15 Layout of Stallion Processor

At the top level, the power rails were manually routed. The routing from these rails to smaller cells was done automatically in IC Craftsman. The definition of via arrays, width of routes and the topology of power routes were described in scripts for IC Craftsman. The

routing algorithms built in IC Craftsman have been found to work much better for chip assembly as compared to routing inside very small cells. Figure 5-15 shows the layout of Stallion chip as seen in Virtuoso Layout Editor.

After the automatic routing was completed, entire design was exported in DFII format to Layout XL. At this stage, Stallion design was a collection of over half a million transistors. On a 400 MHz Sun Ultra 10 machine, it took 96 hours of CPU time to extract the net-list of entire Stallion design and another 16 hours to perform the LVS check.

5.13 Power Distribution

In order to supply power to the entire chip, a network of power rails was created. In this distribution network, the width of the power routes was progressively increased as the design progressed higher in the design hierarchy. Figure 5-16 illustrates this scheme.

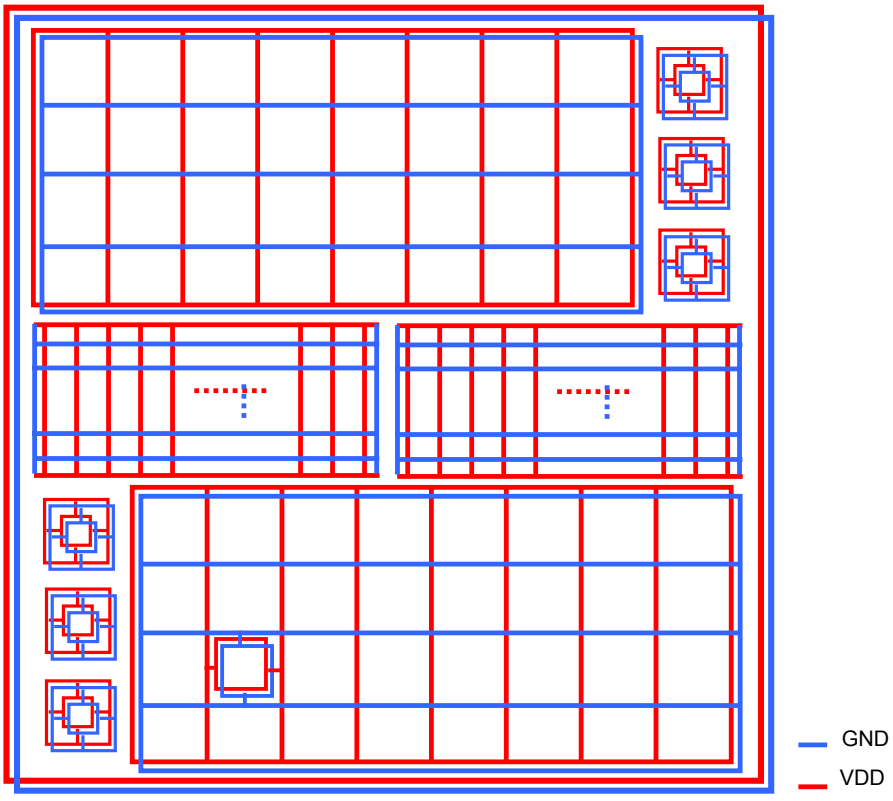

Figure 5-16 Power distribution network

In the figure, the rails for the top two levels in design hierarchy are shown. For sake of

simplicity, the connectivity between the power ring on the die periphery and internal networks is not shown. The internal routing in one of the IFUs is shown. Starting at the topmost level, the width of rails was set at 18 μm and was decremented to 6 μm, 5.4 μm, 2.4 μm, 1.5 μm and 0.45 μm for the design cells at lower levels. The top-level rails supply power to the Meshes, Crossbars and the Data Ports. Each of these components has a power network with rails of lesser width. The power routes were made narrower with the descending hierarchy levels. For cells with approximately 20 transistors, the width of power routes was limited to 4λ.

The package of Stallion die has eight pins for each of VDD and GND rails. These pins are distributed evenly around the die to ensure balanced current flow in various parts of the chip.

5.14 Clock Distribution

Stallion design has an elaborate mechanism for propagating clock signals. The clock signal is strengthened at several places across the chip. The delay in clock signal due to these clock drivers has been kept uniform for all design blocks in Stallion. As a result, the chance of losing synchronism between various blocks is low.

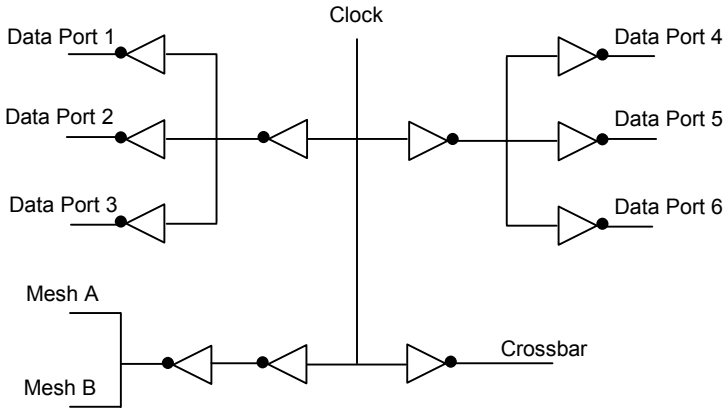

Figure 5-17 Clock Tree

In Figure 5-17, the clock signal tree at the top level of design hierarchy is shown. Inside each of these – Meshes, Data Ports and Crossbar – the clock signal is further conditioned.

The clock tree is illustrated in Figure 5-18. For each IFU and multiplier, there is one dedicated clock driver shown in the figure as an arrowhead.

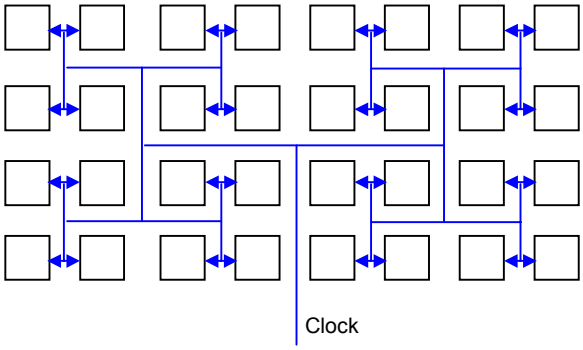

Figure 5-18 Clock Distribution in Mesh

The clock signal in the Crossbar network is distributed in a fashion similar to the Mesh. Figure 5-19 illustrates clock distribution in a Crossbar.

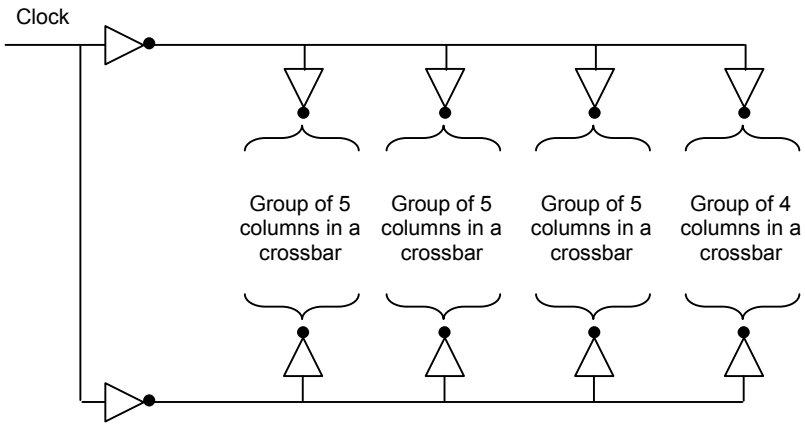

Figure 5-19 Clock Distribution in Crossbar

Each Crossbar is divided into two blocks for distributing clock signal. The scheme shown above is implemented once for each of these blocks. Thus, the scheme shown distributes clock signal in 19x11 grid of Crossbar nodes. Four of such networks complete the clock tree in the Crossbar.

Chapter 6

Conclusions

This thesis has presented the methodology of creating full custom VLSI chips. More specifically, the tools and techniques used to create custom layouts for Stallion runtime reconfigurable processor have been elaborated.

6.1 Results

The most important result of this work is the fabrication of prototype Stallion chips. The prototype chips can be used to assess the performance of Stallion architecture by running applications in hardware. The architecture of Stallion can be benchmarked against other similar computing machines developed elsewhere in industry and academia.

	Module	Number	Height (µm)	Width (µm)	Area (mm^2)	% Die Area
1	Stallion Die	1	7950	7950	63.2	100.0
2	I/O Pads	1	n.a.	n.a.	11.4	17.98
3	Mesh	2	2567	5352	13.7	21.74
4	XBar	2	1541	3257	5.0	7.94
5	Data Ports	6	300	300	0.09	0.14
6	IFU	60	628	659	0.41	0.65
7	FU	60	393	550	0.22	0.34
8	Multiplier	4	520	545	0.28	0.45

Table 6-1 Dimensions of major modules in Stallion

The physical design of a VLSI chip like Stallion can be assessed by the characteristics of layouts. The layouts are characterized by area of the layouts, timing and power consumption statistics. The area occupied by Stallion and its components is tabulated in Table 6-1. As it

can be seen, the I/O Pads, Meshes, Cross Bar and the Data Ports occupy approximately 80 percent of the die area. The remaining 20 percent is used for top-level routing of power rails and signals.

The estimation of power consumption in Stallion has also been performed. To find out the power consumption, it was assumed that the power consumed in Cross Bar and in Data Ports is low in comparison to the power consumption in the Mesh. Further, it was assumed that calculation of all signal lines at the boundaries of each IFU every clock cycle is a good estimate of power consumption in an IFU. With the number of signal lines known, the fringe capacitance of all these wires in an IFU can be estimated using the MOSIS measurement results. Since the operating voltage is 3.3 V and the operating frequency is 50 MHz, all the information needed to assess power consumption is available. Using this data and the calculations shown in Appendix D, the approximate power consumption for Stallion is estimated to be 0.7 Watts. However, since only about 60% of total transistors have been taken into account, this figure is likely to be an underestimate.

At the Mobile and Portable Radio Research Group in Virginia Tech, several algorithms have been mapped to Stallion architecture and analyzed in the Stallion simulator [13]. It has been confirmed that Stallion is particularly efficient for DSP applications. For an example application, Rake Receiver for W-CDMA was mapped on a single Stallion processor [14].

Configuration	Programming Bits	Programming Cycles	Processing Cycles	FUs Used (60 max)
Matched Filter	5632	73	5287	36
Channel Estimation	3008	69	42	22
Channel Compensation	7552	149	228	56
Maximal Ratio Combining	1536	42	86	10

Table 6-2 Stallion Implementation Statistics

The statistics of the Rake-Receiver implementation are listed in Table 6-2. The column named *Configuration* lists various parts of the algorithm that were analyzed. For each

configuration, the number of bits, number of clock cycles and the number of FUs required for programming and processing tasks are listed. It has been observed that only about nine percent of the total compute power that Stallion is capable of providing at 50 MHz is needed. Thus, Stallion processor has a potential to be a platform of choice for high-performance signal processing applications.

Stallion also compares well with the other devices in its class. Each of these devices has implemented reconfiguration at run time or at hardware compile time. PipeRench, Chimaera, Reconfigurable Communications Processor offer partial run-time reconfiguration like Stallion. On the other hand, Jazz can be configured at the time of compiling hardware. Stallion is, however, unique in the sense that not all of the three streams need to be in programming mode at any given time. Overlapping the configuration and processing tasks among various streams can be used to make Stallion available for processing at all times.

Like Stallion, other devices such as PipeRench, Context Switching FPGA and Chimaera processor operate under an external controller or need a microprocessor to interface with. This adds another component that must be designed and programmed before any of these devices can be used. The Reconfigurable Communications processor, however, can operate in stand-alone mode that can simplify the design of complex communications systems.

Stallion has relatively coarse-grained architecture that allows creation of data paths in an easier manner compared to the fine-grained architecture of the Context Switching FGPA that has CSLA cells similar to the CLBs of an FPGA. In this regard, the PipeRench architecture also offers same advantages as the Stallion architecture. The Reconfigurable Communications Processor, Chimaera and Jazz architectures look at the applications more in terms of a sequence of instructions. In these architectures, performance improvement is achieved by making the execution of these instructions fast.

Thus, Stallion compares favorably with the other devices that have been developed in academia and in industry.

6.2 Future Work

This work presented herein was directed towards creating a prototype of Stallion processor. A flow based on the Cadence full-custom physical design tool-chain was implemented to create the mask data consisting of over half a million transistors. This work effectively demonstrated how a chip of the size of Stallion is implemented. Following the fabrication, the immediate task is to verify operation of the chip, evaluate its performance and evolve the methodology to mitigate shortcomings in the design flow adopted for Stallion. This work also offers the opportunity to implement algorithms on the Stallion architecture and make assessment about it. Additionally, improvements in the Stallion design can also be identified.

Bibliography

1. Ray Bittner Jr., "Wormhole Run-Time Reconfiguration: Conceptualization and VLSI Design of a High Performance Computing System", Ph.D. Dissertation, Virginia Polytechnic Institute and State University, 1996.
2. Ray Bittner and Peter Athanas, "Wormhole Run-time Reconfiguration", FPGA97, Monterey, California, 1997.
3. Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matt Moe and R. Reed Taylor, "PipeRench: A Reconfigurable Architecture and Compiler", IEEE Computer, April 2000.
4. Matthew Myers, Kevin Jaget, Srihari Cadambi, Jeffrey Weener, Matthew Moe, Herman Schmit, Seth Copen Goldstein, Dan Bowersox, "PipeRench Manual", Carnegie Mellon University, June 1998.
5. Chameleon Systems, Inc. "CS2000 Reconfigurable Processor", CS2000 Advance Product Information, 2000.
6. Improv Systems, Inc., http://www.improvsys.com
7. Steve Scalera and Jóse R. Vázquez, "The Design and Implementation of a Context Switching FPGA", Proceedings of the 1998 International Symposium on Field-Programmable Custom Computing Machines, April 1998.
8. Scott Hauck, Thomas W. Fry, Matthew M. Hosler and Jeffrey P. Kao, "The Chimaera Reconfigurable Functional Unit", IEEE Symposium on FPGAs for Custom Computing Machines, 1997.
9. Tsuang-Hen Yang, "A Stream-Based In-Line Allocatable Multiplier for Configurable Computing", MS Thesis, Virginia Polytechnic Institute and State University, 1997.
10. MOSIS Website, http://www.mosis.edu
11. Cadence Design Systems, Inc., *Cadence Reference Manuals*, San Jose, CA, Cadence Design Systems, 2000.

12. Toby Schafer, Andy Stanaski, Alan Glaser and Paul Franzon, “The NCSU Design Kit for IC Fabrication through MOSIS”, International Cadence User Group Conference, Austin, Texas, 1998.
13. Srikathyayani Srikanteswara, “Design and Implementation of a Soft Radio Architecture for Reconfigurable Platforms", Ph.D. Dissertation, Virginia Polytechnic Institute and State University, July 2001
14. Srikathyayani Srikanteswara, James Neel, Dr. Jeffrey H. Reed, Dr. Peter Athanas, “Soft Radio Implementations for 3G and Future High Data Rate Systems”, [to be published], 2001

Appendices

A. Design Hierarchy

The hierarchy of design units in Stallion processor is illustrated in Figure A-1 to Figure A-4.

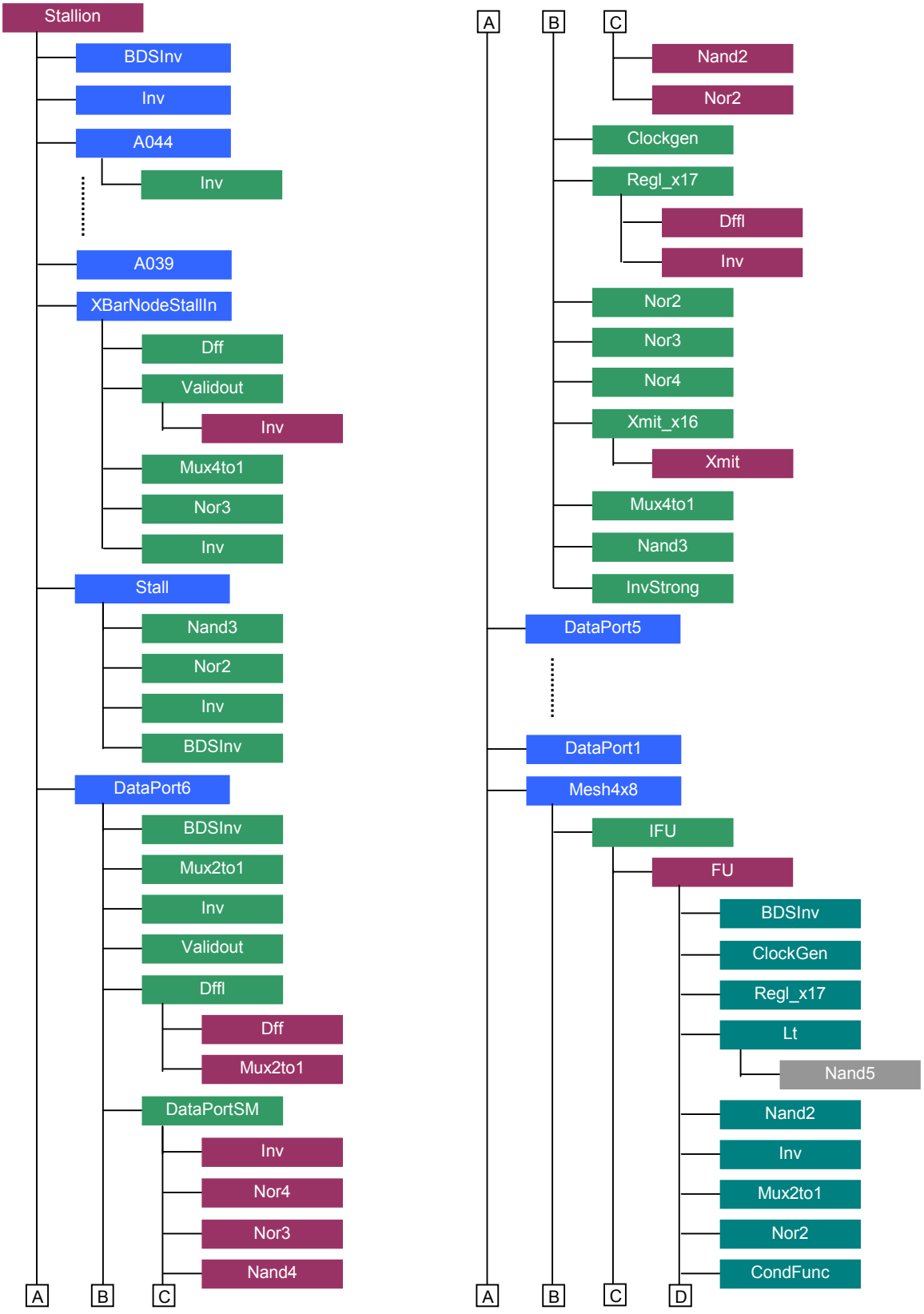

Figure A-1 Design Hierarchy

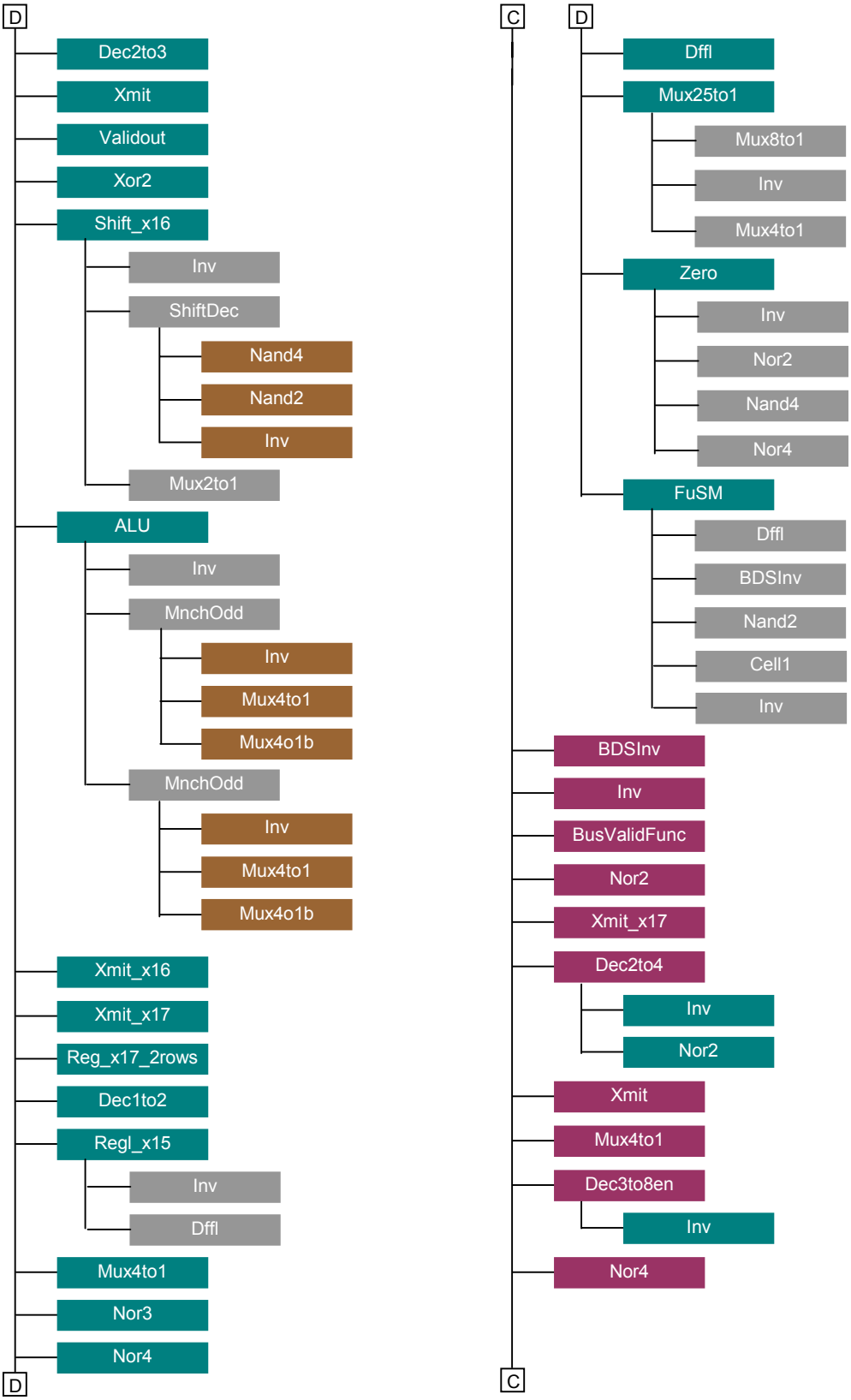

Figure A-2 Design Hierarchy (contd.)

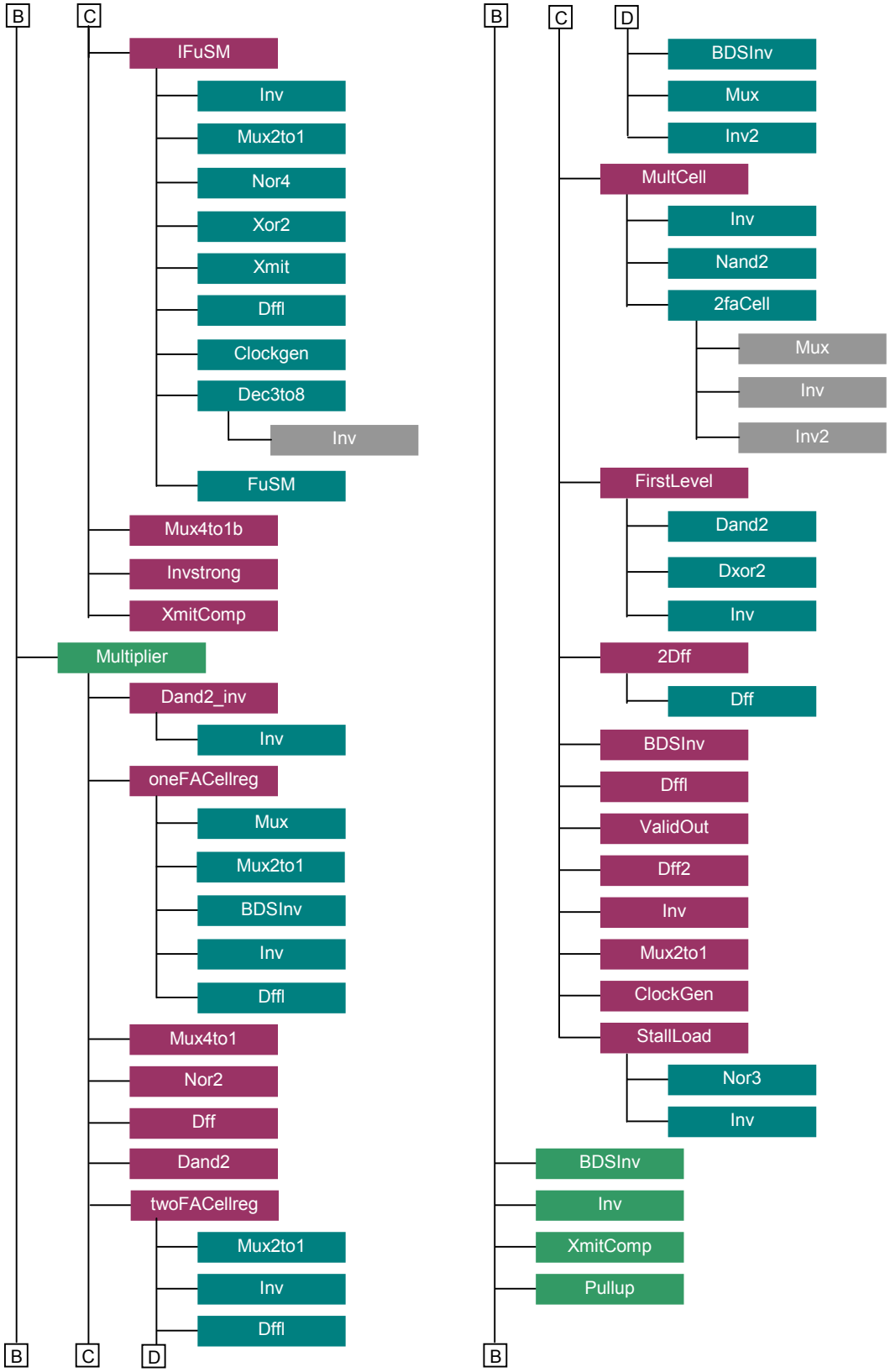

Figure A-3 Design Hierarchy (contd.)

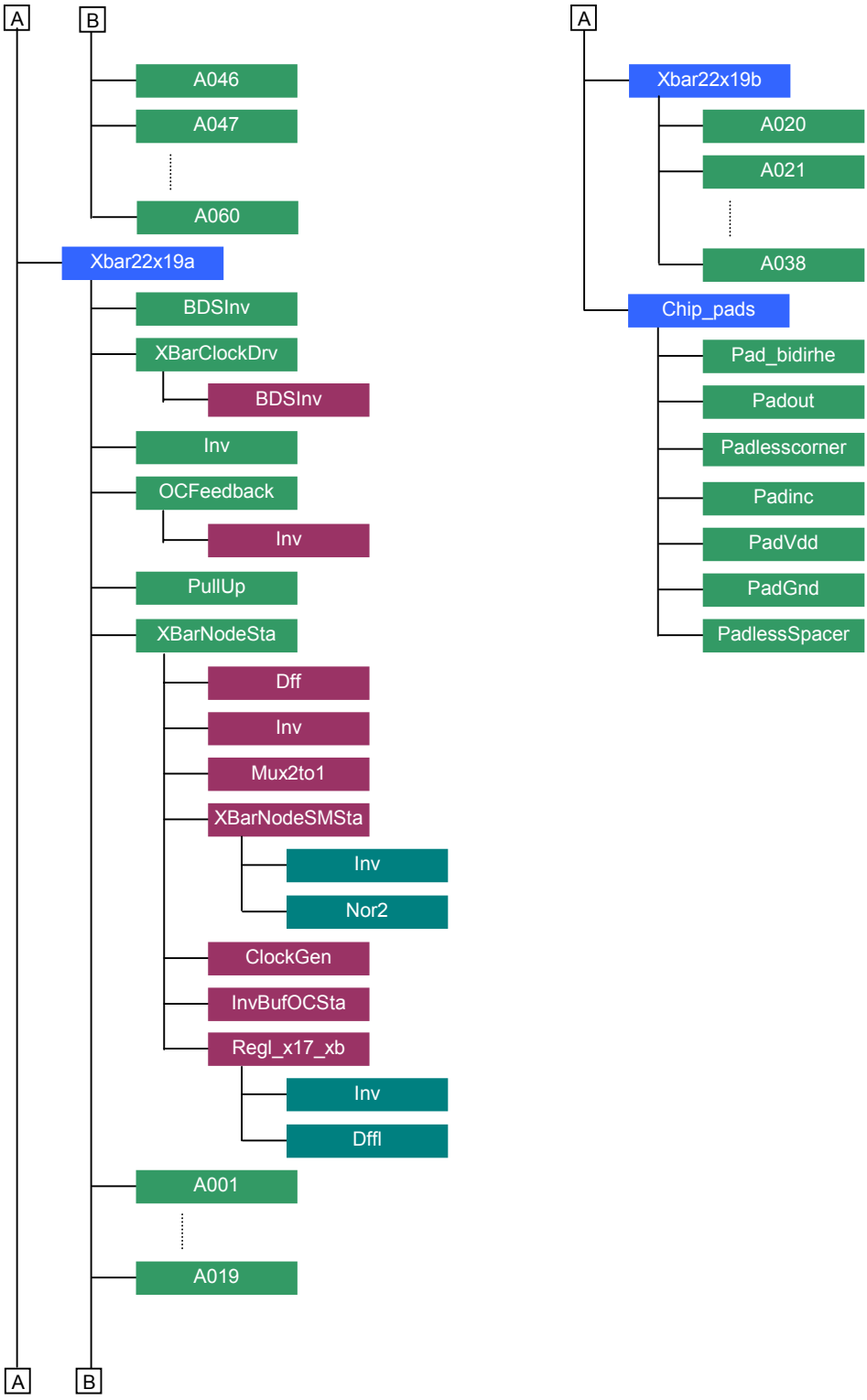

Figure A-4 Design Hierarchy (contd.)

B. TSMC25 Fabrication Process

B.1 MOSIS Parametric Test Results

MOSIS PARAMETRIC TEST RESULTS

RUN: T11Y (EPI) VENDOR: TSMC
TECHNOLOGY: SCN025 FEATURE SIZE: 0.25 microns

INTRODUCTION: This report contains the lot average results obtained by MOSIS from measurements of MOSIS test structures on each wafer of this fabrication lot. SPICE parameters obtained from similar measurements on a selected wafer are also attached.

COMMENTS: TSMC 0251P5M

TRANSISTOR PARAMETERS	W/L	N-CHANNEL	P-CHANNEL	UNITS
MINIMUM	0.36/0.24			
Vth		0.50	-0.47	volts
SHORT	20.0/0.24			
Idss		588	-268	uA/um
Vth		0.52	-0.51	volts
Vpt		7.6	-7.2	volts
WIDE	20.0/0.24			
Ids0		9.7	-3.5	pA/um
LARGE	50.0/50.0			
Vth		0.45	-0.57	volts
Vjbkd		5.9	-7.1	volts
Ijlk		-21.7	-2.1	pA
Gamma		0.44	0.61	V^0.5

K' (Uo*Cox/2)	118.2	-24.4	uA/V^2
Low-field Mobility	390.23	80.55	cm^2/V*s

COMMENTS: Poly bias varies with design technology. To account for mask and etch bias use the appropriate value for the parameter XL in your SPICE model card.

Design Technology	XL
SCN5M_DEEP (lambda=0.12)	0.03
thick oxide, NMOS	0.02
thick oxide, PMOS	-0.03
TSMC25	0.03
thick oxide, NMOS	0.03
thick oxide, PMOS	0.03
SCN5M_SUBM (lambda=0.15)	-0.03
thick oxide, NMOS	0.02
thick oxide, PMOS	-0.03

FOX TRANSISTORS	GATE	N+ACTIVE	P+ACTIVE	UNITS
Vth	Poly	>15.0	<-15.0	volts

PROCESS PARAMETERS	N+ACTV	P+ACTV	POLY	N+BLK	MTL1	MTL2	MTL3	UNITS
Sheet Resistance	4.6	3.5	4.0	60.6	0.08	0.08	0.08	ohms/sq
Width Variation (measured - drawn)	0.10	0.14	0.10		0.15	0.06	0.00	microns
Contact Resistance	5.7	4.8	4.8			3.34	6.76	ohms
Gate Oxide Thickness	57							angstrom

PROCESS PARAMETERS	PLY+BLK	MTL4	MTL5	N_WELL	UNITS
Sheet Resistance	184.2	0.08	0.04	1067	ohms/sq
Width Variation (measured - drawn)		-0.01	0.15		microns
Contact Resistance		9.94	12.96		ohms

COMMENTS: BLK is silicide block.

CAPACITANCE PARAMETERS	N+ACTV	P+ACTV	POLY	MTL1	MTL2	MTL3	MTL4	MTL5	N_WELL	UNITS
Area (substrate)	1792	1893	106	39	18	13	8	8	63	aF/um^2
Area (N+active)			6027	52	21	14	11	10		aF/um^2

Area (P+active)			5797					aF/um^2
Area (poly)				63	18	10	7	6 aF/um^2
Area (metal1)					39	15	9	7 aF/um^2
Area (metal2)						37	14	9 aF/um^2
Area (metal3)							37	15 aF/um^2
Area (metal4)								40 aF/um^2
Fringe (substrate)	406	344		23	60	56	41	24 aF/um
Fringe (poly)				74	41	31	25	21 aF/um
Fringe (metal1)					67	37	28	24 aF/um
Fringe (metal2)						51	36	29 aF/um
Fringe (metal3)							53	38 aF/um
Fringe (metal4)								61 aF/um
Overlap (N+active)			610					aF/um
Overlap (P+active)			656					aF/um

CIRCUIT PARAMETERS			UNITS
Inverters	K		
Vinv	1.0	1.03	volts
Vinv	1.5	1.11	volts
Vol (100 uA)	2.0	0.22	volts
Voh (100 uA)	2.0	2.07	volts
Vinv	2.0	1.17	volts
Gain	2.0	-17.36	
Ring Oscillator Freq.			
DIV1024_T (31-stg,3.3V)		206.05	MHz
DIV1024 (31-stg,2.5V)		285.24	MHz
Ring Oscillator Power			
DIV1024_T (31-stg,3.3V)		0.09	uW/MHz/gate
DIV1024 (31-stg,2.5V)		0.06	uW/MHz/gate

COMMENTS: DEEP_SUBMICRON

B.2 SPICE Parameters

```
T11Y SPICE BSIM3 VERSION 3.1 PARAMETERS

SPICE 3f5 Level 8, Star-HSPICE Level 49, UTMOST Level 8

* DATE: Feb 23/01
* LOT: T11Y                  WAF: 05
* Temperature_parameters=Default
.MODEL CMOSN NMOS (                                LEVEL   = 49
+VERSION = 3.1            TNOM    = 27             TOX     = 5.7E-9
+XJ      = 1E-7           NCH     = 2.3549E17      VTH0    = 0.4113021
+K1      = 0.4212301      K2      = 0.0107813      K3      = 1E-3
+K3B     = 2.0111046      W0      = 5.03895E-7     NLX     = 2.135081E-7
+DVT0W   = 0              DVT1W   = 0              DVT2W   = 0
+DVT0    = 0.2160075      DVT1    = 0.1444576      DVT2    = -0.1362042
+U0      = 328.7872174    UA      = -8.02255E-10   UB      = 1.945003E-18
+UC      = 1.808991E-11   VSAT    = 1.098276E5     A0      = 1.3001133
+AGS     = 0.2645939      B0      = 2.390393E-8    B1      = -1E-7
+KETA    = 7.558555E-3    A1      = 5.303728E-4    A2      = 0.6003119
+RDSW    = 120            PRWG    = 0.5            PRWB    = -0.2
+WR      = 1              WINT    = 2.804916E-9    LINT    = 2.474805E-9
+XL      = 3E-8           XW      = 0              DWG     = 8.685122E-10
+DWB     = 7.704994E-9    VOFF    = -0.1257833     NFACTOR = 0.0195742
+CIT     = 0              CDSC    = 2.4E-4         CDSCD   = 0
+CDSCB   = 0              ETA0    = 8.17751E-3     ETAB    = 1.126094E-3
+DSUB    = 0.0878912      PCLM    = 1.6922564      PDIBLC1 = 1
+PDIBLC2 = 5.240401E-3    PDIBLCB = -0.1           DROUT   = 0.9410143
+PSCBE1  = 7.996789E10    PSCBE2  = 1.457183E-8    PVAG    = 0
+DELTA   = 0.01           RSH     = 4.6            MOBMOD  = 1
+PRT     = 0              UTE     = -1.5           KT1     = -0.11
+KT1L    = 0              KT2     = 0.022          UA1     = 4.31E-9
+UB1     = -7.61E-18      UC1     = -5.6E-11       AT      = 3.3E4
+WL      = 0              WLN     = 1              WW      = -1.22182E-16
+WWN     = 1.2127         WWL     = 0              LL      = 0
+LLN     = 1              LW      = 0              LWN     = 1
+LWL     = 0              CAPMOD  = 2              XPART   = 0.4
+CGDO    = 3.11E-10        CGSO    = 3.11E-10        CGBO    = 1E-12
+CJ      = 1.808639E-3    PB      = 0.99           MJ      = 0.4628536
+CJSW    = 3.660419E-10   PBSW    = 0.99           MJSW    = 0.3167326
+CF      = 0              PVTH0   = -0.01          PRDSW   = 0
```

```
+PK2      = 3.267316E-3    WKETA    = -6.564238E-3   LKETA    = -0.031274)
*

.MODEL CMOSP PMOS (                                 LEVEL    = 49
+VERSION = 3.1            TNOM     = 27             TOX      = 5.7E-9
+XJ       = 1E-7          NCH      = 4.1589E17      VTH0     = -0.5979778
+K1       = 0.5862081     K2       = 0.0114948      K3       = 0
+K3B      = 8.6450158     W0       = 1.458288E-6    NLX      = 1E-9
+DVT0W    = 0             DVT1W    = 0              DVT2W    = 0
+DVT0     = 1.7706541     DVT1     = 0.4452494      DVT2     = -0.0759572
+U0       = 150.0872115   UA       = 2.473146E-9    UB       = 1E-21
+UC       = -7.73644E-11  VSAT     = 2E5            A0       = 0.6745826
+AGS      = 0.0437272     B0       = 1.661543E-6    B1       = 5E-6
+KETA     = 0.0217103     A1       = 5.390741E-4    A2       = 0.7107435
+RDSW     = 929.504272    PRWG     = 0.0656674      PRWB     = -0.5
+WR       = 1             WINT     = -2.057265E-9   LINT     = 2.583385E-8
+XL       = 3E-8          XW       = 0              DWG      = -1.79913E-8
+DWB      = 1.692346E-8   VOFF     = -0.1306214     NFACTOR  = 0.614634
+CIT      = 0             CDSC     = 2.4E-4         CDSCD    = 0
+CDSCB    = 0             ETA0     = 0.2636802      ETAB     = -0.0760326
+DSUB     = 0.8425492     PCLM     = 1.187932       PDIBLC1  = 0
+PDIBLC2  = 0.0189656     PDIBLCB  = -1E-3          DROUT    = 1
+PSCBE1   = 2.902357E10   PSCBE2   = 8.368983E-9    PVAG     = 4.1641349
+DELTA    = 0.01          RSH      = 3.5            MOBMOD   = 1
+PRT      = 0             UTE      = -1.5           KT1      = -0.11
+KT1L     = 0             KT2      = 0.022          UA1      = 4.31E-9
+UB1      = -7.61E-18     UC1      = -5.6E-11       AT       = 3.3E4
+WL       = 0             WLN      = 1              WW       = 0
+WWN      = 1             WWL      = 0              LL       = 0
+LLN      = 1             LW       = 0              LWN      = 1
+LWL      = 0             CAPMOD   = 2              XPART    = 0.4
+CGDO     = 2.68E-10      CGSO     = 2.68E-10       CGBO     = 1E-12
+CJ       = 1.895793E-3   PB       = 0.9859519      MJ       = 0.4680019
+CJSW     = 3.348085E-10  PBSW     = 0.7271758      MJSW     = 0.3060725
+CF       = 0             PVTH0    = 4.727645E-3    PRDSW    = 27.8542201
+PK2      = 2.591311E-3   WKETA    = 2.368072E-3    LKETA    = -0.0152222)
*
```

C. Packaging Information

C.1 Package

Stallion is packaged in a Ceramic PGA181 package. It is a Kyocera KD-P84141-C model. This package has a 472 mil square shaped cavity. The external measurement is 1.675 inch on each side. The pins are arranged in a 15x15 grid with inter-pin spacing of 0.1 inch. There are four rows of pins on each side of the package. The PGA181 footprint is depicted in Figure C-1.

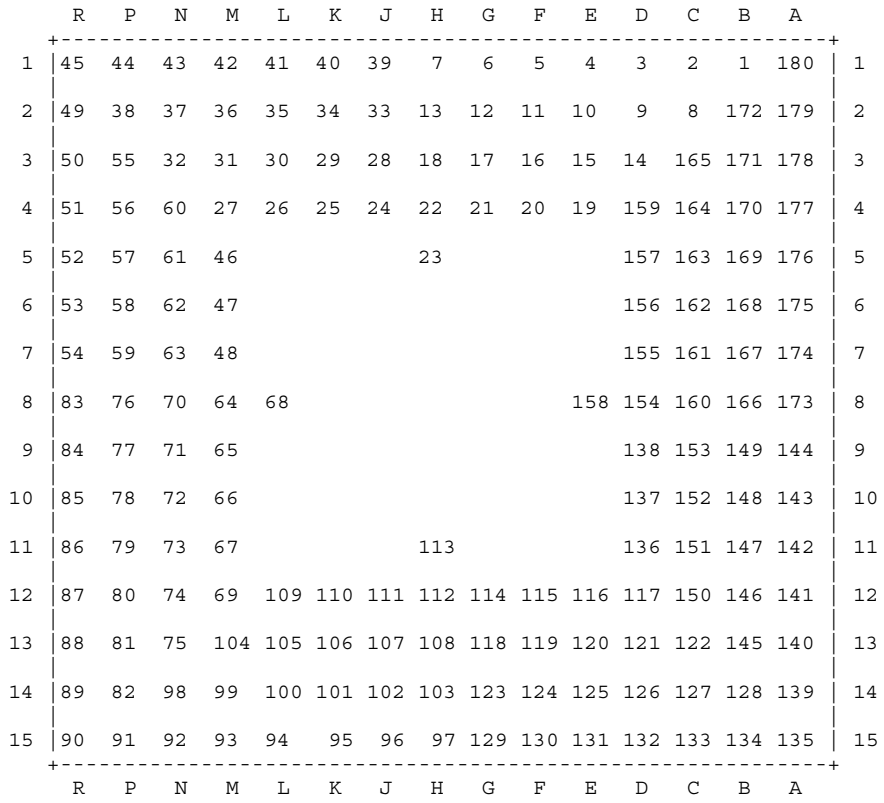

Figure C-1 Pin locations in a PGA 181 package

C.2 List of Pins

##	Pin Name
1	Port6WritePinOut
2	Port6ReceivePinOut
3	Port6ReceivePinIn
4	Port6TransmitPinOut
5	Port6TransmitPinIn
6	VDD
7	Port6ProgramPinOut
8	Port6ProgramPinIn
9	DP6Q1
10	DP6Q2
11	DP6Q3
12	Port6ZBus00
13	Port6ZBus01
14	Port6ZBus02
15	Port6ZBus03
16	Port6ZBus04
17	GND
18	Port6ZBus05
19	Port6ZBus06
20	Port6ZBus07
21	Port6ZBus08
22	Port6ZBus09
23	Port6ZBus10
24	Port6ZBus11
25	Port6ZBus12
26	Port6ZBus13
27	Port6ZBus14
28	VDD
29	Port6ZBus15
30	SyncBus3
31	SyncBus4
32	SyncBus5
33	Port5WritePinOut
34	Port5ReceivePinOut
35	Port5ReceivePinIn
36	Port5TransmitPinOut
37	Port5TransmitPinIn
38	Port5ProgramPinOut
39	GND
40	Port5ProgramPinIn
41	DP5Q1
42	DP5Q2
43	DP5Q3
44	Port5ZBus00
45	Port5ZBus01
46	Port5ZBus02
47	Port5ZBus03
48	Port5ZBus04
49	Port5ZBus05
50	VDD
51	Port5ZBus06
52	Port5ZBus07
53	Port5ZBus08
54	Port5ZBus09
55	Port5ZBus10
56	Port5ZBus11
57	Port5ZBus12
58	Port5ZBus13
59	Port5ZBus14
60	Port5ZBus15
61	GND
62	Port4WritePinOut
63	Port4ReceivePinOut
64	Port4ReceivePinIn
65	Port4TransmitPinOut
66	Port4TransmitPinIn
67	Clock
68	Port4ProgramPinOut
69	Port4ProgramPinIn
70	DP4Q1
71	DP4Q2
72	DP4Q3
73	VDD
74	Port6ZBus00
75	Port6ZBus01
76	Port4ZBus02
77	Port4ZBus03

78	Port4ZBus04
79	Port4ZBus05
80	Port4ZBus06
81	Port4ZBus07
82	Port4ZBus08
83	Port4ZBus09
84	GND
85	Port4ZBus10
86	Port4ZBus11
87	Port4ZBus12
88	Port4ZBus13
89	Port4ZBus14
90	Port4ZBus15
91	Port3WritePinOut
92	Port3ReceivePinOut
93	Port3ReceivePinIn
94	Port3TransmitPinOut
95	Port3TransmitPinIn
96	VDD
97	Port3ProgramPinOut
98	Port3ProgramPinIn
99	DP3Q1
100	DP3Q2
101	DP3Q3
102	Port3ZBus00
103	Port3ZBus01
104	Port3ZBus02
105	Port3ZBus03
106	Port3ZBus04
107	GND
108	Port3ZBus05
109	Port3ZBus06
110	Port3ZBus07
111	Port3ZBus08
112	Port3ZBus09
113	Port3ZBus10
114	Port3ZBus11
115	Port3ZBus12
116	Port3ZBus13
117	Port3ZBus14
118	VDD
119	Port6ZBus15
120	SyncBus0

121	SyncBus1
122	SyncBus2
123	Port2WritePinOut
124	Port2ReceivePinOut
125	Port2ReceivePinIn
126	Port2TransmitPinOut
127	Port2TransmitPinIn
128	Port2ProgramPinOut
129	GND
130	Port2ProgramPinIn
131	DP2Q1
132	DP2Q2
133	DP2Q3
134	Port2ZBus00
135	Port2ZBus01
136	Port2ZBus02
137	Port2ZBus03
138	Port2ZBus04
139	Port2ZBus05
140	VDD
141	Port2ZBus06
142	Port2ZBus07
143	Port2ZBus08
144	Port2ZBus09
145	Port2ZBus10
146	Port2ZBus11
147	Port2ZBus12
148	Port2ZBus13
149	Port2ZBus14
150	Port2ZBus15
151	GND
152	Port1WritePinOut
153	Port1ReceivePinOut
154	Port1ReceivePinIn
155	Port1TransmitPinOut
156	Port1TransmitPinIn
157	ResetBar
158	Port1ProgramPinOut
159	Port1ProgramPinIn
160	DP1Q1
161	DP1Q2
162	DP1Q3
163	VDD

164	Port1ZBus00
165	Port1ZBus01
166	Port1ZBus02
167	Port1ZBus03
168	Port1ZBus04
169	Port1ZBus05
170	Port1ZBus06
171	Port1ZBus07
172	Port1ZBus08

173	Port1ZBus09
174	GND
175	Port1ZBus10
176	Port1ZBus11
177	Port1ZBus12
178	Port1ZBus13
179	Port1ZBus14
180	Port1ZBus15

D. Packaging Information

The calculations to estimate power consumption in Stallion are provided below:

Capacitance Parameters for Metal 2:
Area (substrate) = 18 aF/um^2
Fringe (substrate) = 60 aF/um

Area Capacitance:
Area of a wire (M2) over IFU: 650 um x (3 x 0.15 um) x 18 aF/um^2 = 5265 aF
So, area capacitance is 5.3 fF for one wire in metal 2 that is 3λ wide and spans across the IFU

Fringe Capacitance:
Fringe (M2-substrate) capacitance = 60 aF/um
#(toggling wires in metal 2 on IFU boundary) = n
Total fringe capacitance = n x 60 aF/um x 650 um = 39000n aF
So, fringe capacitance is 39 femto Farad times the number of toggling signals every clock.

Operating Voltage (V) = 3.3 V, Frequency (f) = 50 MHz

For rough estimates, if n=1000 for one IFU, C= 39 pf (Area capacitance ignored)
For 64 IFUs (considering Multiplier ≡ IFU) power dissipation is
64 x (½ CV^2f) = 64 x ½ x (39 x 10^{-12}) x (3.3 x 3.3) x (50 x 10^6) = 0.7 Watts

E. CAD Tools

In this section, the procedure of creating a typical cell as done for Stallion processor is described. It is assumed that the reader has a working knowledge of Cadence Virtuoso set of tools. In this description, the schematic view of a cell is the starting point and from that, layouts are created using Layout XL and IC Craftsman. The procedures for physical verification and creation of mask data are also provided.

E.1 Outline

The description given herein follows the following sequence of steps:

1. Perform automatic pickup of components and create pins using a template Virtuoso XL template.
2. Place the components and pins in Virtuoso XL. Perform DRC.
3. Export the layout to IC Craftsman
4. Perform automatic routing and check for completeness.
5. Import the layout into Virtuoso XL
6. Perform DRC and LVS.
7. Export the GDSII file.

E.2 Procedure

In this section, the exact way of performing the tasks listed in previous section is explained.

E.2.1 Component Pickup

Open the schematic view in Schematic Composer. It is necessary that the layout view of all the cells in that are used in the schematic are available in same library with identical names.

- Select *Tools* → *Design Synthesis* → *Layout XL*

- In the *Startup Option* dialog box, select *Create New* option and click *OK*.
- Another dialog box *Create New File,* select the name for the cell to be created and click OK.
- ICFB spawns the Layout XL tool. The schematic window and the Layout Window are connected to each other for sharing design information with each other.
- In the Layout XL menu bar, select *Design → Generate from Source*. In this case, the schematic view forms the source.
- After a small delay, the layout views of each of the components in the schematic are placed in the layout view. A window *Layout Generation Window Option* pops up. In this Window, all the pins found in the schematic are listed. The geometrical attributes of the pins can be set in this window. Also, the size of layout view can be set so that a rectangular boundary for the cells is automatically drawn. From the *Pin Placement* option in this window, the location of pins can be specified. Alternatively, a template file can be used to set all these options outside of the GUI. The sample template file is shown in Appendix E. After selecting all options in *Layout Generation Window Options* window, click *OK*.
- The tool completes placement of pins as specified and draws the boundary according to given dimensions.
- Using the commands as in Layout Editor, place the components manually. The components can also be placed automatically with IC Craftsman.

E.2.2 Export to IC Craftsman

The design data is exported to IC Craftsman as described below:

- From the menu bar in Layout XL, select *Route → Export to Router*
- A pop-up form *Export to Router* shows various options for exporting the design information to IC Craftsman. All these options can also be set in a file and then, only the filename needs to be specified using the *Load Defaults* option. Additionally, IC Craftsman needs to know about properties of the layouts and the layers used in the design. A file containing these settings can be created using

Route → Rules → New Rules. The already existing rules file can be edited using *Route → Rules → Open Rules.* With the options set, click *OK.* All the design information is exported to the directory specified in options form.

E.2.3 Automatic Routing

- After exporting the design data, start IC Craftsman from command line. A form pops up in which the path to newly exported design files is specified.
- Browse the files and select the design file that was exported. The design files have a *.dsn* extension. Click *Start Router.* The IC Craftsman window would appear and automatic placement and routing can be performed.
- Before the tools can do the routing, constraints settings and routing guidelines must be specified. These settings are specified in the IC Craftsman menu items *Rules* and *Define.*
- For sake of simplification, all these settings have been saved as a file. These files are essentially a list of IC Craftsman commands. When this file, also called the *do file*, is executes, all the commands that would otherwise be executed using the graphical interface are executed through the script. Separate scripts may be needed for cells at different levels in a design hierarchy. A sample *do file* has been given in Appendix E.
- After all the settings are made, global routing is done by *Autoroute → Global → Global Route.* Global routing is necessary for detailed routing to be successful. In global routing, the router determines rough estimates of how the nets would be routed.
- If global routing is not successful, then the errors are first removed and the global routing is repeated to ensure no errors exist. Typical errors can be of overlapping cells or inaccessible pins. These errors must be manually corrected.
- Detailed automatic routing is accessed from *Autoroute → Route.* By default, the tool runs 25 passes of auto-routing efforts. Routing can finish before all passes are over. Sometimes, the router algorithms can never finish routing. In such cases, routing has to be forcibly stopped. Then, the already created routes might

have to be ripped and auto-routing has to be restarted with relaxed constraints. Detailed description of these options and settings can be found in Cadence Reference Manuals [11].

- After auto-routing, the design is saved as a session (*.ses)* file using the menu option *File → Write → Session.*

E.2.4 Import from IC Craftsman

After the session file is available from IC Craftsman, the session file needs to be brought back into the Virtuoso Environment. To accomplish this, following steps are needed:

- In Virtuoso XL, open the cell for which routing information is to be imported.
- Choose *Route → Import from router.* In the form that pops up, the name of the session file created in IC Craftsman must be specified. The session file is analysed and the routes are drawn in the cell view open in Virtuoso XL.

E.2.5 Physical Verification

For verifying that the layout has no errors, DRC and LVS are performed.

- From the menu bar in Layout XL, select *Verify → DRC.* The name of the DRC rules file is required in the DRC form. According to the rules, the layout is checked for correctness. If there are any errors, they must be either corrected manually or through another run in the automatic router tool.
- Following DRC, both schematic and layout net lists must be compared. The schematic net list is extracted from schematic by *Design → Check and Save* option in Schematic Composer. For layout, the option *Verify → Extract* in Layout XL is used. As cells grow in size, extraction of net list from layout can be a very long procedure. Adequate compute power and swap space on machines is necessary.
- After the net lists are extracted, LVS tool is invoked from Layout XL using *Verify → LVS* option. In the LVS form, the library, cell name, view name, LVS rules file and LVS options are set. Then LVS check is started. The net lists are compared and results are logged in a file specified in the LVS options. The net

list files might have to be analysed to find out any layout related bugs that cause LVS failure. The bugs should be corrected and the entire physical verification process should be repeated until the cell passes both tests. The design is then complete.

E.3 Sample Files

In this section, samples of the *template file* used in Virtuoso XL and the *do file* used in *IC Craftsman* are being given.

E.3.1 Sample template file for Virtuoso XL

```
;; Template file created on Nov 13 2000
;; Maneesh Soni
IO_section(
   (type "geometric")
   (layer ("metal2" "pin"))
   (shape (rectangle width 0.450000 height 0.450000))
   (multiplicity 1)
   (pin "AddressCompare"             (position (left) (order 15)))
   (pin "ChipProgramIn"              (position (left) (order 14)))
   (pin "ChipProgramOut"             (position (right) (order 14)))
   (pin "ChipStallIn"                (position (left) (order 13)))
   (pin "ChipStallOut"               (position (right) (order 13)))
   (pin "ChipValidBitIn"             (position (left) (order 12)))
   (pin "GND!"                       (shape (rectangle width 0.600000 height 0.600000))
                                     (position (bottom) (order 0)))
   (pin "PinBusDir"                  (position (right) (order 12)))
   (pin "POut"                       (position (left) (order 0)))
   (pin "ProgramPinIn"               (position (left) (order 11)))
   (pin "ProgramPinOut"              (position (right) (order 11)))
   (pin "Q1"                         (position (left) (order 10)))
   (pin "Q1p"                        (position (right) (order 10)))
   (pin "Q2"                         (position (left) (order 9)))
   (pin "Q2p"                        (position (right) (order 9)))
   (pin "Q3"                         (position (left) (order 8)))
   (pin "Q3p"                        (position (right) (order 8)))
   (pin "RegBusDir"                  (position (right) (order 6)))
   (pin "RW"                         (position (left) (order 7)))
   (pin "RWLoad"                     (position (right) (order 7)))
   (pin "RWState"                    (position (left) (order 6)))
   (pin "StallIn"                    (position (left) (order 5)))
   (pin "StallLoad"                  (position (right) (order 5)))
```

```
   (pin "StallOut"                    (position (right) (order 4)))
   (pin "StallRegOutEnable"           (position (right) (order 3)))
   (pin "StartOfPacketMarker"         (position (left) (order 4)))
   (pin "SynchronizationReadyIn"      (position (left) (order 3)))
   (pin "SynchronizationReadyOut"     (position (right) (order 2)))
   (pin "TransmitPinIn"               (position (left) (order 2)))
   (pin "TransmitPinOut"              (position (right) (order 1)))
   (pin "ValidBitOut"                 (position (right) (order 0)))
   (pin "VDD!"                        (shape (rectangle width 0.600000 height 0.600000))
                                      (position (top) (order 0)))
)
```

E.3.2 Sample *do file* in IC Craftsman

```
rule IC (pin_width_taper up_down (max_length 0))
set same_net_checking on
grid via 0.075 (direction x) (offset 0)
grid via 0.075 (direction y) (offset 0)
local_direction layer_panel
cost layer metal5 free  (type length)
cost layer metal4 free  (type length)
cost layer metal3 free  (type length)
cost layer metal2 free  (type length)
cost layer metal1 free  (type length)
cost layer poly high  (type length)
cost layer poly high  (type length)
view groute_blocked_pins on

rule net Clock (pin_width_taper up_down (max_length 0))
rule net Clock (width 0.6)
circuit net Clock (priority 235)

rule net VDD! (pin_width_taper up_down (max_length 0))
# rule net VDD! (width 0.6)
circuit net VDD! (priority 255)
circuit net VDD! (use_layer metal5 metal4 metal3 metal2 metal1)
circuit net VDD! (use_via  M5_M4 M4_M3 M3_M2 M2_M1 M1_POLY)
rule net VDD! (via_on_pin on (grid off) (fit off (via_center_enclosed off)))

rule net GND! (pin_width_taper up_down (max_length 0))
rule net GND! (width 0.6)
circuit net GND! (priority 245)
circuit net GND! (use_layer  metal5 metal4 metal3 metal2 metal1)
circuit net GND! (use_via M5_M4 M4_M3 M3_M2 M2_M1 M1_POLY )
rule net GND! (via_on_pin on (grid off) (fit off (via_center_enclosed off)))
```

F. Stallion Snapshots

This section contains pictures of the Stallion die taken through an optical microscope with an attached digital camera.

In the first photograph, shown in Figure F-1, all major subcomponents of Stallion – Mesh, Cross Bar and Data Ports – can be identified.

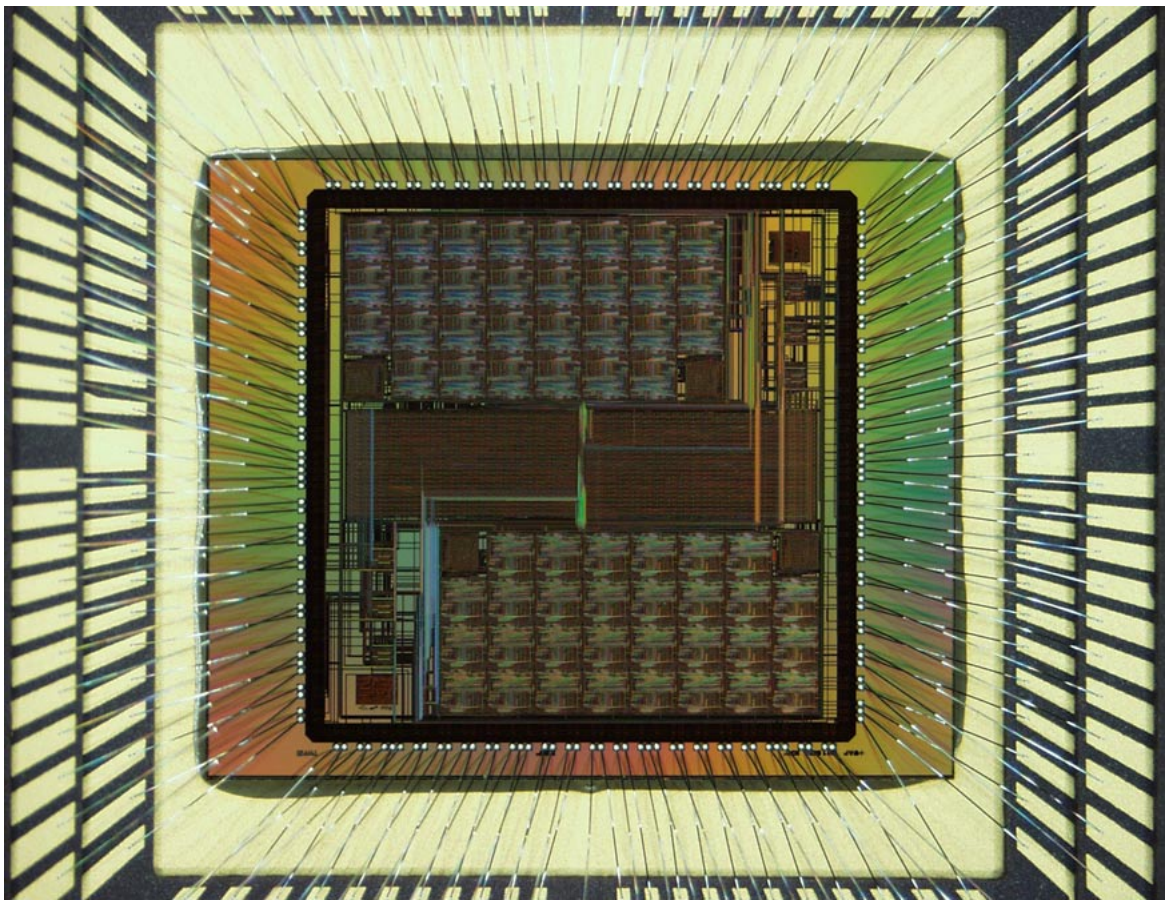

Figure F-1 Photograph of the Stallion die

In the second photograph, the JPEG images embedded on Stallion die can be seen. The photograph has the author seated on steps. Right next to this photograph on the top edge are -

the logo of Configurable Computing Lab, author's autograph and logo of Virginia Tech. The JPEG images were inserted in Virtuoso layouts using the *p2m* tool availed from the Cadence Research group at NCSU. This photo also shows the bond pads, bond wires, a data port and top-level routes.

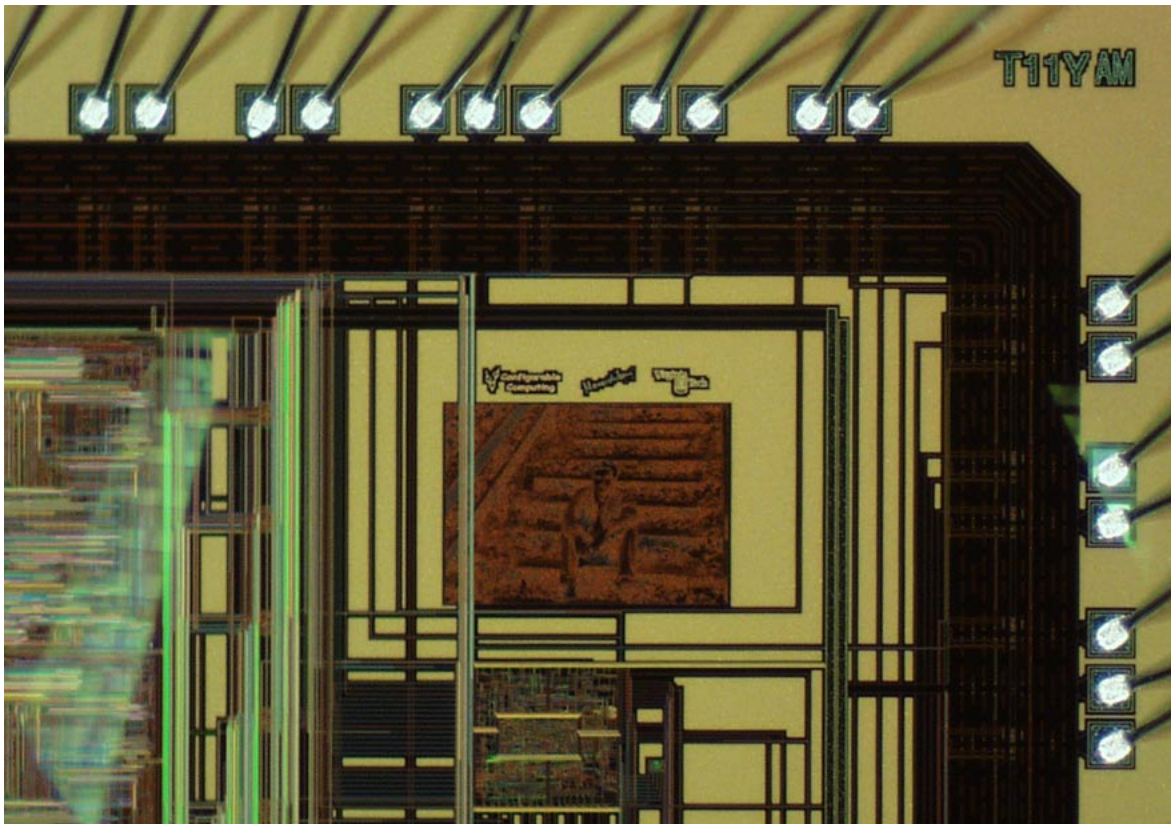

Figure F-2 Photograph and logos embedded in Stallion using the *p2m* tool

Vita

Maneesh Soni was born on October 20, 1976 in the *City of Lakes*, Udaipur, India. His early education was obtained at several different schools, the last of which was St. Paul's School in Udaipur, India. At the completion of high-school education, he went to the Indian Institute of Technology in New Delhi, India. After a period of four years, he was conferred a Bachelor of Technology in Electrical Engineering. Following a stint in the professional world for a year and a half, he returned to academia to pursue graduate studies at Virginia Tech in Blacksburg, USA. He plans to graduate with a Master's degree in Computer Engineering in June 2001. In the fall of 2001, he would go back to industry and contribute to the technological advancements in wireless and digital systems engineering.