SYSTEM-ON-A-CHIP (SOC) DESIGN AND TEST - A CASE STUDY

by

Louis Tzu-Leng Hong

B.A.Sc., The University of British Columbia, 2000

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Applied Science

in

The Faculty of Graduate Studies

Department of Electrical and Computer Engineering

We accept this thesis as conforming to the required standard

The University of British Columbia

August 2002

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical and Computer Engineering

The University of British Columbia
Vancouver, Canada

Date September 9, 2002

ABSTRACT

System-on-a-chip (SoC) and reuse of intellectual property (IP) is the emerging paradigm for integrated circuit designs. To understand the unique challenges in IP development and SoC integration, a microprocessor core and a network processor SoC were developed. The Reuse Methodology Manual (RMM) by Keatling and Bricaud was used as a guide during the development of the IP core and the SoC. This thesis presents several examples taken from the microprocessor and SoC designs that either support or counter the claims made in RMM. The problem of SoC testing is also a highly researched area. In an effort to validate the concept of an on-chip test network, a packet-switching test access mechanism (TAM) was designed and integrated into the network processor SoC. The TAM, known as NIMA, is an on-chip network that supports different types of embedded core testing. The NIMA architecture was compared with a serial TAM and a multiple-inputs TAM based on the Test Rail architecture. The three TAM designs were compared based on the total test time, area overhead, and complexity of the controlling mechanism. This thesis also discusses the trade-offs of the three TAM architectures and suggests some improvements for NIMA to reduce its area and delay overhead.

TABLE OF CONTENTS

LIST FIGURES

LIST TABLES

ACRONYMS

ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
ATE	Automated Test Equipment
BIST	Built-in Self Test
BDR	Boundary Data Register
CAD	Computer Aided Design
CISC	Complex Instruction Set Computer
DFT	Design for Test
DSM	Deep Sub-Micron
DSP	Digital Signal Processing
EDA	Electronic Design Automation
FIFO	First In First Out
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
IP	Intellectual Property
JTAG	Joint Test Access Group
LFSR	Linear-Feedback Shift Register
PCI	Peripheral Component Interface
PLL	Phase Lock Loop
QoS	Quality of Service
RTL	Register Transfer Level
SoC	System on a Chip
TAM	Test Access Mechanism
UDL	User Defined Logic
VHDL	VHSIC (Very High Speed Integrated Circuits) Hardware Description Language
VLSI	Very Large Scale Integration
VSIA	Virtual Socket Interface Alliance

ACKNOWLEDGEMENT

I would like to thank Dr. Res Saleh for giving me the opportunity to study my Master degree with him and to work in the SoC lab. Without his continuous guiding and support, the completion of this thesis would not be possible.

This research is supported by PMC-Sierra, the National Sciences and Engineering Research Council of Canada, Gennum Corporation, and Micronet. Their support is greatly appreciated.

This thesis inherits a lot of work from the NIMA project, and I thank Dr. Andre Ivanov and Mohsen Nahvi for providing valuable comments and information on NIMA. Also, I would like to thank Professor Hussein Alnuweiri and Professor Alan Hu for their advices on network processor design and system verification. I also thank Professor Steve Wilton for serving on my thesis committee.

I owe much to my family, who encouraged me to pursue the Master degree in the first place and has supported me during these years. I am also grateful to the following people in the SoC lab, who contribute to the research directly or indirectly: Ronald, Gary, Julien, Zahra, Martin, Simon, Victor, Kara, James, Laura, Roberto, and Roozbeh. It has been a great pleasure to work with this team of energetic, bright researchers.

Finally, I dedicate this thesis to Peggy. Her encouragement and her confidence in me have helped me through the difficult times.

CHAPTER 1 INTRODUCTION

The continuous reduction of device feature size ushers the possibility of integrating high-level blocks –microprocessor, DSP core, memory module, and graphic controller etc.- on a single silicon substrate, where each block was formerly a chip on a printed circuit board. The so-called system-on-a-chip (SoC) design enables higher performance, lower power, and less package cost than an equivalent system implemented as traditional integrated circuits (ICs). The realization of such complex designs, however, is hindered by the parasitic properties of the semiconductor devices in deep-sub-micron (DSM) range. Issues in DSM technology include low noise margin, signal integrity due to coupling capacitance, interconnect delay, inductive effect of the wires, and sub-threshold conduction. To resolve the DSM issues, computer-aided-design (CAD) tools must incorporate improved circuit model and advanced circuits design techniques. Models that accurately represent the 3-D structure of the interconnects, inductance loop, and power consumption are essential to capture the DSM parasitic effects. Techniques such as shielding of wires that are susceptible to coupling, buffer insertion for global interconnect, and substrate biasing for reducing leakage current should be automated by the CAD tools in order to speed up the design process.

Another limiting factor for creating a highly integrated system is the escalating cost of design, verification, and test. A typical application specific integrated circuit (ASIC) designed today consists of approximately 2-million gates in 7 RTL blocks [44]. The future system is expected to contain tens of millions gates or more. The traditional ASIC approach is not economic, if not impossible, due to the sheer amount of engineering

effort to code each block from scratch, perform multiple iterations of synthesis and integration, and generate test vectors for each blocks (not to mention the difficulty in testing the blocks once they are embedded inside the fabricated chip). In addition, the IC industry is characterized by short product life cycles and thus there is pressure to minimize time-to-market. Indeed, there is a widening gap between the engineering productivity and the IC design complexity that the technology allows [28][14]. Figure 1 illustrates the concept of the productivity gap. According to Moore's Law [59], the transistor density of the IC designs grows exponentially due to improvement in process technologies, but the productivity of the engineers (measured in gates per day) stays relatively constant. New design techniques and methods have allowed the IC design complexity to follow the growth of the manufacturing capability. Similar to how logic synthesis boosted the productivity of the design teams in the 90's, design reuse is expected to bridge the productivity gap confronted by today's engineers.

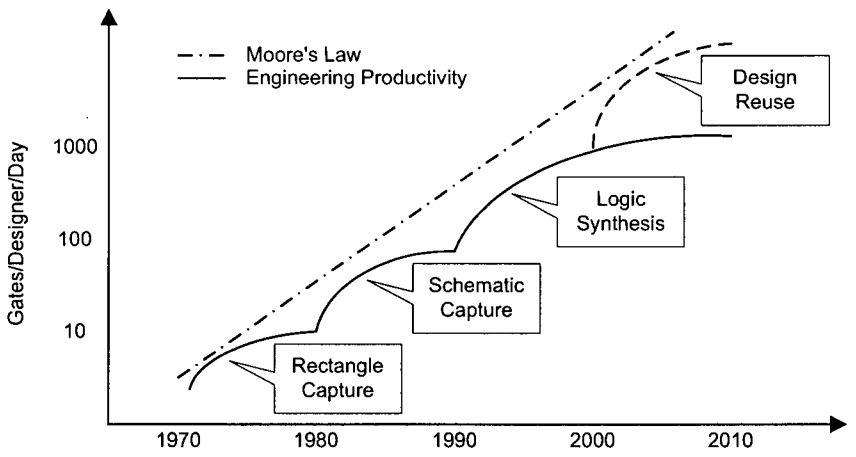

Figure 1. Productivity gap

The key concept of design reuse is to adopt Intellectual Property (IP), which could be a building block or an entire platform, from in-house design teams or third-party vendors to form the majority of the system logic. The IP vendors provide a description of the design to the system integrator, who integrate the IPs and add custom logic that differentiates the product from the competitors'. In most cases, some verification methods such as pre-computed test data or self-test capability would accompany the design description so the system integrator can exercise the tests on the manufactured end product to ensure quality. This approach leverages previous design effort, reduce risk, and shorten development cycle.

Figure 2 shows a generic SoC consisting of IP blocks, user-defined logic (UDL), embedded memory, a system communication mechanism, a core test mechanism, and the IEEE 1149.1 standard (also known as JTAG) for board testing. The IP blocks (either third-party of in-house) are self-contained, reusable, and designed to perform macro functions. Examples of digital IP blocks include a microprocessor, a DSP, and a PCI interface. The analog cores are typically phase-locked loop (PLL) modules and analog-digital converters. The UDL is specific to the application of the SoC; it may serve merely as the glue logic between the cores, or it may be the key enhancement that allows the SoC to be differentiable. The communication mechanism, typically an on-chip bus, is an important aspect of the SoC since it impacts the overall speed of the design and defines the transaction protocol that must be agreed by all blocks. The core test mechanism is for post-manufacture testing. Its design ranges from simple wires that connect between block terminals and the I/O ports to sophisticated circuitries that automated the test process. It

is obvious that putting all these components together and ensuring that they function correctly is a major engineering effort.

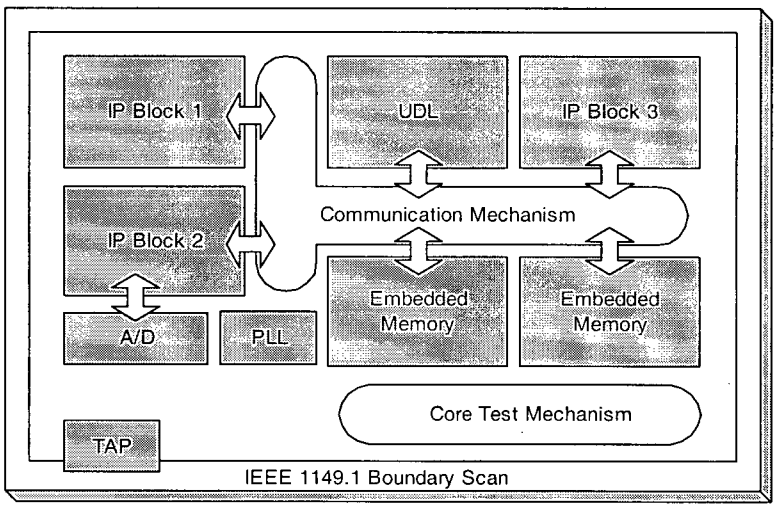

Figure 2. Block diagram of a generic SoC chip

Although promising, the SoC design methodology is still in its infancy. There are many challenges in IP authoring, embedded software design, system verification of the SoCs, and SoC testing. In addition, the industry at this moment has no standardized methods to address the problems of third-party IP delivery (where the IP could be soft, firm, or hard, with various hand-off requirements), IP security, IP library management, and integration of the IP test suits [30][48].

In particular, test integration is a significant bottleneck due to the difficulty in creating an efficient chip-level test strategy. Inefficient test may extend development time due to testability-related revisions and incur a high cost in test equipment, or compromise the quality of the product. For instance, the traditional IC verification techniques use slower

clocks for testing, which suffer from reliability problem in SoC because the higher density of devices, and wires increase the chance of failure when the chip is running at full speed. Also the controllability and observability of the embedded blocks pose an access problem since the terminals of the embedded core may not be probed directly [9].

There has been tremendous effort from the IC industry to overcome the challenges in SoC design. The Virtual Socket Interface Alliance (VSIA), formed in 1996, has been working on standards for exchanging the IP blocks at physical and functional level [62]. With uniform standards, IP block can be portable from company-to-company and technology-to-technology. If successful, IP integration will be a matter of plug-and-play. In addition to standards that facilitate portability, new design flows are needed to create reusable IP blocks. The Reuse Methodology Manual (RMM) [33] is an attempt to capture the complex process of IP creation and integration.

For SoC testing, many researchers believe that a hierarchical, built-in-self-test (BIST) scheme is the trend because of the low requirement on an external tester and the high scalability [28][9]. Two problems related to SoC testing are the standardization of IP test interface and transfer of test data from chip I/O pins to the embedded blocks. The IEEE P1500 group was formed with the goal of formulating standards for test interface and a formal language for describing a test procedure [58]. A number of research projects have been conducted in the area of test access mechanisms (TAMs) through chip I/O. Whestel proposed a technique for SoC verification by reusing the IEEE 1149.1 standard for board testing on an SoC, where the embedded cores are threaded together in a manner similar to

how the chips on a printed circuit board are serially connected [64]. Marinissen et. al. presents a structural test access mechanism known as a Test Rail that distributes the available test bandwidth among the blocks within an SoC [37]. Recently, the concept of network-on-a-chip is gaining in popularity in the research community, and [42] proposed a solution to the access problem using an on-chip network.

The motivation of this research is to investigate the design and test issues of SoCs through a case study and to propose improvements for the existing methods. The first half of the research explores the design and reuse aspects by evaluating new VLSI design flows for the SoC paradigm. The second half of the research compares several test access mechanisms for SoC verification, discusses the trade-off between the competing approaches, and suggests implementation improvements that optimize the existing methods. This document is organized as follows. Chapter 2 describes the effort to create a reusable microprocessor. Chapter 3 discusses an SoC design that is built from the reusable microprocessor presented in Chapter 2 and various other blocks. Chapter 4 presents several SoC test strategies and describes the on-chip test network in details. Chapter 5 compares three distinct chip-level test access mechanisms: serial based, bus based, and network based. It presents a comprehensive analysis for each of the three architectures and discusses their weakness and strength. Chapter 6 draws the conclusions and describes the future work.

CHAPTER 2 REUSABLE IP CORE DESIGN (HC11)

The top-down design levels of an SoC are system, IP blocks, sub-blocks, macro cells, gates, and transistors. In this thesis, the terms IP block and core are interchangeable as they both refer to design entities of the same level of abstraction. An example of an SoC is a DVD optical-disk controller, which has a microprocessor core, memory blocks, and analog-to-digital components [34]. The microprocessor core consists of sub-blocks including a finite state machine (FSM), an ALU, and a register file. The ALU sub-block has a full-adder macro cell, and the macro cell is made up of logic gates that are, in turn, implemented by CMOS transistors.

Today's IP blocks can be categorized based on their degree of modifiability. The different categories of IPs can also be viewed as the same design taken from different stages of the IC design flow. The soft IPs are designs at the register-transfer level (RTL). Soft IPs are typically delivered as Verilog or VHDL code and allow the highest degree of modifiability. The soft IPs also impose the greatest amount of work since the user needs to take the IPs through the process of synthesis, physical design, and verification. The firm IPs are designs presented at the schematic level. Firm IPs are delivered as synthesized gate-level netlist, either mapped to a specific manufacturing process or to a generic library. Since firm IPs are already in the form of gates, modification of the IP's functionality is virtually impossible, but it still allows IP users to place and route the design according to the SoC requirements (e.g. aspect ratio and shape). The hard IPs are designs in the layout form. Hard IPs are exchanged as GDS-II files and have the least degree of modifiability. Although hard IPs are already placed and routed and are thus

difficult to change, they are the most complete of the three categories of IPs and thus require the least design effort from the user [33].

The embedded memories are similar to the hard IPs in the sense that a memory block cannot be modified by the end user. But since the embedded memories are generated by memory compilers, the end user can decide the high-level specifications such as capacity and single-port versus dual-port. The memory compilers receive design parameters from the user and then generate the models of corresponding the memory block to be used throughout the design process. The actual memory block is placed on the chip by the foundry during fabrication.

The soft IPs represent 10-15% of the third-party IP market today, and they are all digital designs [47]. The firm IPs have around 5% of the market share, and are mostly analog designs. The rest of 80-90% of the IP market is occupied by hard IPs, which include both digital and analog designs [47]. For internal IPs created for in-house use, soft IP dominates because they are portable across different generation of process technologies. The kinds of IPs available in today's market ranges from interface blocks, processors, analog/mix-signal functions, to programmable IPs. Common interface IPs include Ethernet, PCI bus client, and IEEE 1149.1 TAP controller. Examples of the processor IPs include general-purpose processors from ARM© or DSP cores from Texas Instruments©. Examples of analog/mix-signal IPs include PLL for clock jitter control, A/D converter, and Bluetooth™ baseband module from Tality©. The programmable IPs such as the

solutions offered by eASIC© and Actel© are reconfigurable logic blocks based on the traditional FPGA technology [56].

The key requirement for creating an SoC design is reusable IP blocks that support plug-and-play integration. To reduce the development cost, IP integration must be automated as much as possible. A library of reusable IP blocks with various timing, area, and power configurations is also essential to SoC success as it allows mix-and-match of different IP blocks so the integrator can make the trade-offs that best suit the needs of the application. The rapid growth in the commercial IP business reflects this demand [34]. In addition, most system vendors already have internal designs that were created for one-use only in past projects. Categorizing and reworking these legacy designs for reusability also represent a major challenge. The process of creating a reusable IP block differs from the traditional ASIC design approach. Performance, area, power, and features may be sacrificed in order to ensure that the design will be reusable across different applications. To better understand the issues in creating a reusable IP block, a series of projects were initiated to create new IP blocks and to convert a one-use design to a reusable one, adopting RMM as the guide.

The design chosen for reusability rework is the microprocessor core of a Motorola HC11 microcontroller. The primary reason for choosing this particular design is the availability of information. Most industry-strength designs are proprietary and require expensive licensing. Also state-of-the-art designs are complex and have legal implications that make modification and distribution of these designs difficult. The HC11 core used in this

project can be obtained easily from public domain sources [57][41]. There are several other reasons for choosing this particular design. First, a processor core is by far the most prevalent IP block, as almost every application requires one. This makes the exercise of reusability rework more valuable as the IP will likely be used in future projects. Secondly, the design is synthesizable and has been verified and implemented on an FPGA by the original developers, so there is confidence in the quality of the design. Finally, the core developers working on this project all have prior knowledge of the HC11 micro-controller from a user perspective. Their familiarity with HC11 instruction set helps to reduce the initial learning curve.

2.1 Introduction to HC11

The HC11s are popular micro-controllers for automotive control systems, robotics, consumer appliances, and other embedded applications. A typical micro-controller is composed of a central processing unit (CPU), static memory, non-volatile memory (e.g. EEPROM), and parallel and serial I/O devices, all connected using a processor local bus system. The design under investigation is the CPU for the HC11 family of micro-controllers. For the rest of this thesis, the design will be referred to as the HC11. The original code of the HC11 was written in VHDL behavioral description. A simple testbench for functional verification accompanied the original code. RTL simulation indicated that the design was functional, although the design was not created with reusability in mind. The HC11 core is a complex-instruction-set computer (CISC) that executes more than 130 instructions. It has enough complexity to make it a representative IP block and yet simple enough for the purpose of this research project.

The HC11 core is an 8-bit processor. It adopts a non-pipelined architecture and requires multiple clock cycles to execute a single instruction. Since the HC11 core is a CISC processor, the instructions have variable length depending on the addressing mode (direct, extended, indexed, immediate, inherent, or relative) and the type of data (8-bit or 16-bit). All peripherals, I/O, and memory locations must reside in a unified 64-Kbytes address space. The HC11 core performs the standard integer operations, bit manipulation, and logic operations. In addition, the processor has specialized instructions for manipulating and exchanging data between the stack pointer and the internal registers, which allows efficient implementation of complex addressing schemes for DSP algorithms (e.g. FFT). Figure 2 shows the block diagram of the HC11 core.

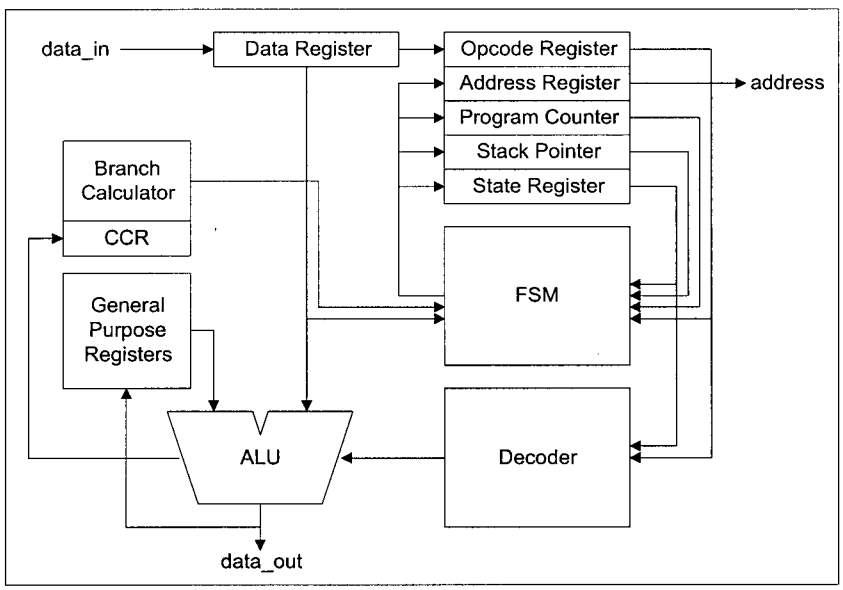

Figure 3. HC11 block diagram

2.2 Design for reuse

Since the goal of reuse is to create an IP block that can be used and possibly modified in other projects by other design teams, completeness of documentation and readability of the code are important. In this aspect, many of the techniques for reuse are simply good engineering practice. Examples of good practice suggested by RMM include:

- Hierarchical organization: partition the design into sub-designs
- Proper in-line comments and detailed design manual.
- Good Coding styles: use meaningful names, consistent naming convention, and constants instead of hard-coded values

The original VHDL description of the HC11 consists of one single, large entity. Considerable amount of time and effort was spent on re-organizing the code and breaking it down into sub-blocks. Since the description is behavioral, the partitioning is based on functionality rather than proximity of physical location or area of the logic elements. In the end, six entities were created [53]. Proper coding styles and formatting were applied during the coding of each new entity. Additional comments were added to help explain the purpose of the code. For reusability, an equivalent set of Verilog RTL code was also created. A more challenging task is to create a comprehensive document that captures the specifications of the design and records the development process so the user of the HC11 block knows what changes were introduced to the design. For specifications, the on-line manual from Motorola [41] was referenced. Information about the newly created design hierarchy, simulation results, and the chosen design-for-test (DFT) strategy were then added to complete the document [29].

Indeed, developing a reusable IP block requires more than just writing clean code and preparing good documentation. There are many technical issues that need to be addressed, as the IP developer must anticipate the application in which the IP block may be used. The RMM suggests several design-for-reuse guidelines, which include:

- Use synchronous design instead of asynchronous because current timing-driven place-and-route tools produce better result with synchronous design
- Avoid latches, as they have ambiguous timing. When latches are used, one cannot tell whether the data is intended to be latched at the beginning of the enable phase or the end, so there is no way to know whether time-borrowing is used or the path has delay problem.
- Buffer the I/O of the core to create clean timing interfaces
- Avoid tri-state-buffers because of drive strength issues. Manual transistor sizing is usually required and it makes them "non-reusable". In addition, tri-state buffers need special care during test or power-up to avoid bus contention.
- Minimizing the number of clock domains
- Thorough verification with 100-percent code-coverage testbench

The importance of thorough verification is obvious. The problem lies in developing a good testbench, which is a major design effort by itself. In the case of the HC11, testbench development takes over 50% of the total development time. RMM recommends a bottom-up strategy for which sub-modules of a block are verified entirely before being integrated into the higher-level entity. This divide-and-conquer approach breaks down the block-level verification task into sub-tasks that can be handled individually. RMM also

suggests using EDA tools to accelerate the process of testbench generation and code coverage analysis to quantify the quality of the testbench. The original testbench suit that accompanied the HC11 code was very primitive and was only designed for the integrated core. Sub-block testbenches were developed during the reusability rework project and test cases for each HC11 instruction were created to supplement the original testbench. The testbenches were all handcrafted and simulation waveforms obtained through these testbenches were manually inspected. The process is time-consuming, but it is still the most cost-effective way of catching design errors. The EDA tools cannot completely replace the manual processes; they only speed up or simplify the tasks involved. In addition, code reviewing remains a very effective method for verification and should not be overlooked.

RMM also advocates the practice of I/O buffering since it contains the timing problem within a core and provide clean interfaces for inter-block communication. This recommendation, however, has a significant impact on area and performance when applied to the HC11. After the initial addition of I/O buffers (i.e., all I/O ports were registered by edge-triggered flip-flops), the HC11 failed many instructions that involved memory access. Follow-up analysis of the state transition reveals that the data path and the control path were misaligned due to the additional cycles introduced by I/O buffers. So a memory write instruction that originally takes n cycles to complete now takes n + 1 cycles, but the state machine still assumes n cycles for the instruction and generates the control signals accordingly. Clearly this recommendation cannot be applied blindly. RMM does not warn of this potential pit-fall because it assumes the IP block is designed

from scratch and the additional cycles from I/O buffering is taken into account during the design phase. To correct the misalignment, the entire state machine had to be re-coded to include the extra cycle. With I/O buffers, the HC11 behaves very similar to a pipelined design, so numerous wait states were inserted to clear the pipeline when branching occurs. After the state machine is corrected, the average cycles per instruction increases by 25%, and the synthesized gate area of the core becomes 34% larger than before the correction. In addition, changing the code increases the risk of introducing new errors and becomes extra verification burden.

The penalty of adding I/O buffers to the HC11 is severe. Later, it was found that the problem could be mitigated by enlarging the IP boundary of the HC11 to include the interface logics and adding I/O buffers at the top-level entity. Figure 4 shows the proposed HC11 block with new boundary definition. The HC11, which is only the CPU part of the microcontroller, cannot perform any meaningful function by itself. It requires a local memory module for data storage and interface to external bus to communicate with other devices. If we consider the HC11, local memory, the local bus, and the external interface all as one single IP block, timing synchronization can be easily achieved within the IP block and we need only provide clean timing at the external interface. With the new definition, the state machine of HC11 does not need to be modified. This approach trades reuse granularity for performance, area, and redesign effort.

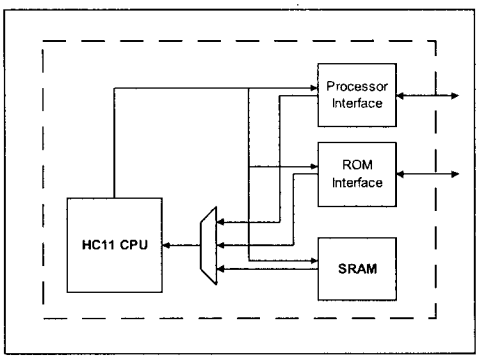

Figure 4. Refined block boundary of HC11

2.3 Coding for Synthesis

For a design to be reusable, it must be useable in the first place. One important aspect of creating useable HDL code is to ensure that it is synthesizable. The purpose of the HDL code is to describe a piece of hardware, and thus it must comply with certain rules set forth by the synthesis tool. The synthesis tool recognizes the constructs in the RTL code and translates them to equivalent gate-level netlists. However, not all RTL constructs are synthesizable. RMM suggests several guidelines for coding synthesizable designs and for avoiding mismatches between pre-synthesis and post-synthesis simulations. These include:

1. Use the standard VHDL and Verilog template for inferring random logic
2. Assign default values in conditional statements to avoid unintentional latch inference
3. Specify a complete sensitivity list
4. Use non-blocking assignments in Verilog *always* statements for sequential logic
5. The *case* statements simulate faster and can be synthesized into faster circuit, and thus is preferred over *if-then-else* statement

According to RMM, guideline #4 avoids mismatches between pre-synthesis and post-synthesis simulation, but the cause of the mismatch is not explained, nor is the usage of blocking vs. non-blocking statements in Verilog. The lack of explanation is a prevalent problem in RMM. Although RMM provides many useful guidelines, the designers sometimes can make a better decision by knowing the reason behind these guidelines. According to [11], the mismatch is caused by the scheduling policy of Verilog events, which dictates the blocking assignments to be scheduled before non-blocking assignment.

When a design is in the RTL form, the software simulators must follow the scheduling rule, as it is part of the Verilog standard. For sequential logics modeled as blocking assignments, race condition could occur (a newly assigned signal is overwritten by another assignment before the next triggering clock edge) because all blocking assignments scheduled at the same time step are processed in arbitrary order. The race condition doesn't necessarily lead to incorrect synthesis, but it always results in incorrect pre-synthesis simulation. For combinational logic modeled as non-blocking assignments, the assignment results are not updated until the end of the current time step. This requires the signal that appears in both left-hand side and right-hand side of the assignments to be placed in the sensitivity list to model the correct behavior, but doing so will trigger the sensitivity list multiple times and thus waste simulation cycles. These observations lead to the following refined guidelines:

1. Use non-blocking assignment in Verilog *always* statements for sequential logic
2. Use blocking assignment in Verilog *always* statements for combinational logic

3. Use non-blocking assignments in Verilog *always* statements that model both sequential and combinational logic

Traditionally, the flow from a high-level system model to RTL code has been done manually and the quality of the code (area and speed of the synthesized system) depends heavily on the skill and experience of the engineers. Since many of the synthesis and simulation problems can be avoided by using standard templates for the logic elements, the most efficient way of generating synthesizable RTL code, in the author's opinion, is behavioral synthesis. Behavioral synthesis generates RTL representation of a design from a higher-level description using control/data flow graph [5]. Similar to IP reuse, high-level behavioral synthesis can leverage the expertise of experienced designers and at the same time automate both coding-for-synthesis and coding-for-reuse. This method also allows the engineers to focuses on system-level design and facilitates software/hardware co-simulation. The current candidates for the standard of behavioral synthesis language include SystemC [61] and Superlog [58].

Behavioral synthesis, however, is more than just mapping C/C++ code constructs to logic elements. A behavioral synthesis tool needs to schedule the high-level operations (such as add, multiply, and move) for optimal area or speed, automatically generate the controlling finite state machine (FSM), infer the most efficient memory element (register, register file, or random-access memory), and analyze the affect on system performance versus power consumption through architectural changes. Although behavioral synthesis is used predominantly for DSP application, the large saving in design cycle (as much as 5

times has been reported) makes it very attractive for IC designers in general [7]. The continuous improvements in scheduling algorithms and tool integration have made behavioral synthesis feasible for applications other than DSP.

2.4 Design for Test (DFT)

A reusable design should also include DFT that provides testability to the block and allows ease of integration into a system-level test strategy. There are a variety of block-level DFT strategies: full/partial scan, BIST, and test path through functional logic. Full scan is the most prevalent strategy for manufacturing because of its high fault coverage and CAD tool support. Although scan has a small penalty in area and delay (signals must propagate through multiplexers), the overhead is negligible for most ASIC designs. RMM and other sources [55][10] provide details on how to create scan-compatible designs using several well-known guidelines. These guidelines are either to ensure the operation of the scan structure or are the result of tester and ATPG tool limitations. The guidelines include:

- No sequentially generated asynchronous set or reset should be used. Asynchronous control signals can render the flip-flop uncontrollable during scan shift
- Avoid more than one triggering edges in one clock domain. Dual-edge designs makes timing analysis more difficult and creates edge placement problem in the tester
- Avoid tri-state buffers to prevent signal contention during scan shift
- Avoid gated clocks. The flops using gated clocks cannot be clocked from primary input and thus impossible to scan in data
- Avoid latches because of unreliable test capture.

The HC11 adopts the logic BIST strategy. A single scan chain was inserted into the synthesized netlist of HC11 using Design Compiler™ from Synopsys©. Scan insertion is a two-stage process. First scannable flip-flops are used during mapping from RTL to technology-specific netlist. This is accomplished by the *-scan* option of the *compile* command. Then the test ports of the scannable flip-flops are stitched together by the *insert_scan* command. Since HC11 contains two separate clock domains, a lock-up latch was inserted between the two clock domains during scan stitching. Figure 5 demonstrates how the lock-up latch is inserted between scan cells belonging to two different clock domains. Shifting test vectors through multiple-clock-domain scan chain resembles a clock skew problem. Using the scan chain in Figure 5 as an example, if CLK2 is later than CLK1, flip-flop C and flip-flop B will register the same value at the same clock cycle and thus the data will be shifted one cycle too early. With the lock-up latch in place, the value of flip-flop B is available to flip-flop C only after the falling edge of CLK1. In effect, the lock-up latch delays the scan data for half a clock cycle and thus increases the scan chain's tolerance to skew.

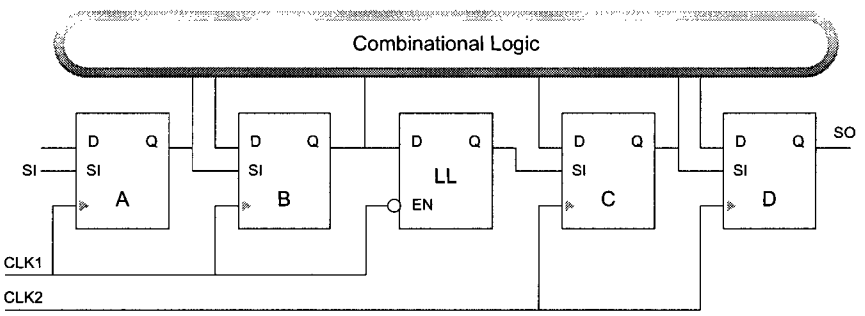

Figure 5. Using lock-up latch to combine scan cells of different clock domains

After the scan chain was in place, a BIST block was integrated into the top-level IP block as shown in Figure 6. The BIST uses a 32-bit linear-feedback shift register (LFSR) to generate pseudo-random test vectors and uses a signature analyzer to compress the test result. In order to control the HC11 during test mode, the clock to the HC11 is gated inside the BIST. Furthermore, latches are used in the BIST to prevent glitches and race condition. A state machine within the BIST generates the control signals to the LFSR and the scan counter. The LFSR and the counter must be started precisely at the same time, so latches are used to hold control signals for half clock period such that both the LFSR and the scan counter start exactly at the triggering edge of the following cycle.

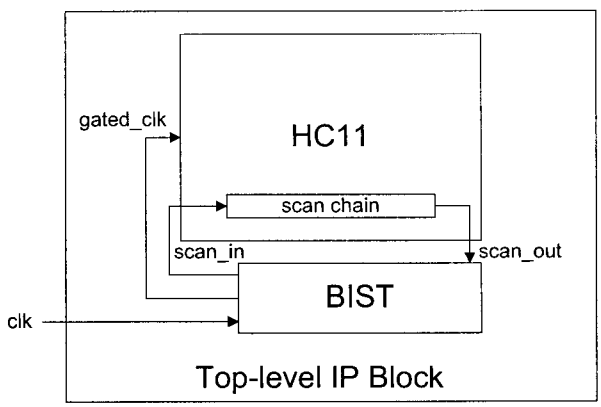

Figure 6. BIST strategy for HC11

Although the usage of gated clocks and latches is discouraged by RMM, we found both techniques necessary for the desired BIST functionality. The argument against clock gating is the danger of generating glitches, which will trigger unintentional capture of the flip-flops. In addition, the flip-flops clocked by the gated clock may not be scannable because the automatic test equipment (ATE) cannot control the gating signal, thus controllability during testing is lost. To overcome the problem, the clock gating logic in

the BIST is carefully designed so that glitches could not occur as a result of switching of the gating signal. Since the HC11 is triggered by the falling edge of the clock, the gated clock is forced to VDD when disabled so the gated clock can only monotonically rise to VDD as the gating signal switches. The argument of non-scannable flops does not apply here since the BIST replaces the ATE and takes control of the clock during testing.

The dilemma posed by the BIST is how to verify the correctness of the BIST itself before it is used to test core logic. Often a chip is discarded regardless the correctness of the BIST if the BIST generates a wrong signature. Testing BIST is useful in a redundancy scheme where multiple instances of BIST are available and a backup BIST can be activated to replace the faulty one. To address this issue, full scan test methodology is used for the BIST. Since latches are used inside the BIST, they affect the testability of the BIST. The ATPG tool expects a single edge event at the sequential logic, while the latches respond to both clock edges [14]. If the latches are driven by clocks, there is potential danger of a race condition in which scan data may be captured too early. If the latches are driven by enable signals that are generated by other logic, then ATPG may not be able to propagate faults through the latches because they do not behave as pure combinational circuits. To solve the latch problems in scan-based design, the common approaches are to either replace the latches with scannable variant during synthesis or make the latches transparent during testing. Since our standard gate library does not provide scannable latches, the approach of using transparent mode is adopted.

After successful scan insertion, a complete timing analysis of the design is required to ensure that there is no hold-time violation or race conditions in the scan chains. The problem is that there was no tool for static-timing analysis installed on-site. The best that could be done was to explore the options offered by Design Compiler for scan insertion and to perform gate-level simulation to catch any glitch and setup/hold time violations. Moreover, power analysis needs to be performed on the design with the scan chain inserted. The Power Compiler allows estimation of power consumption based on only the synthesized gates and an abstract wire-load model. This of course is a very rough estimation since the result does not include an accurate calculation the power due to interconnects and does not use the actual activity factor (which needs to be obtained through simulation). These problems need to be addressed as better tools become available.

2.5 Prototyping

Testing on the physical implementation of the IP block is necessary to guarantee the usability of the IP block before integration into SoC. This is especially true for a third-party IP vendor who must demonstrate the validity of the IP block. As a result, prototyping is an essential part of IP block verification [33]. The most common approach of IP prototyping is either an FPGA implementation or a complete implementation as an ASIC design. FPGA is suitable for debugging, but it does not have the speed and area advantages of ASIC technology. The HC11 was implemented on both as an ASIC and on a FPGA. The ASIC design uses TSMC 0.18μm technology and was carried out using Cadence back-end tools. Figure 7 shows the finished chip of the HC11 in loose-die.

Valuable experience in physical design and packaging for digital circuits had been learnt from this exercise. The FPGA on which the HC11 was implemented is a Xilinx Virtex-2000E integrated into a rapid-prototyping board with an ARMTDMI7 core. From our experience, the ASIC implementation takes 2899 logic gates and runs at 50 MHz. The FPGA, on the other hand, takes 2557 4-input LUTs and runs at a maximum frequency of 23 MHz. It is clear that the FPGA implementation cannot compete with ASIC even with a small design such as the HC11. However, the quick turnaround time from design modification to re-test and the cost effectiveness comparing to ASIC re-spin makes FPGA ideal for the initial validation.

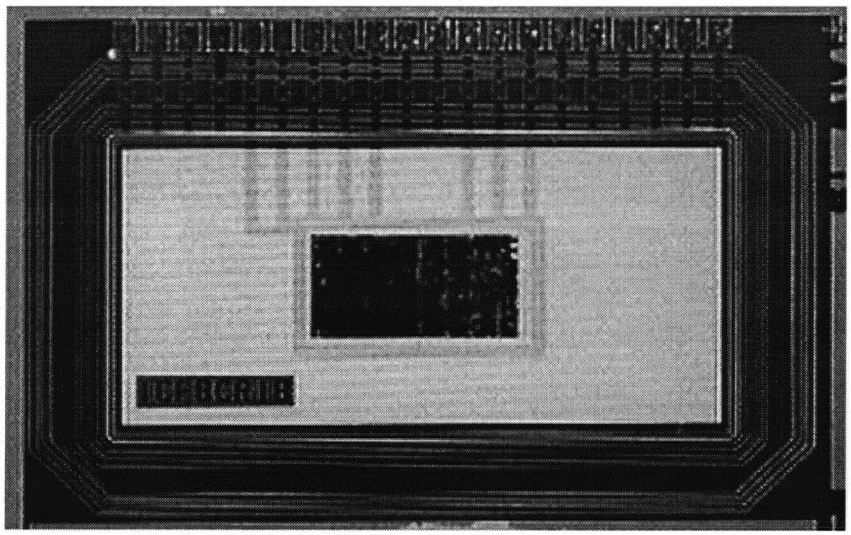

Figure 7. Die photo of the fabricated HC11 core

CHAPTER 3 SOC DESIGN (NETWORK PROCESSOR)

Many practical issues of an IP design do not emerge until it is actually applied in a real system. As a result, integration of an IP block in an SoC is an important validation step. In fact, most design managers will not use third-party IP unless it has been proven in larger SoC designs. However, IP integration poses distinct challenges in SoC design. Contrary to traditional ASIC design, an SoC combines several logic and memory blocks and often requires parallel development of hardware and software. The choice of the IP blocks and the method by which they interact are critical to the performance, power and area of the end system. In an attempt to understand the SoC design process and how reusability of the core affects integration, a simplified network processor was developed and the HC11 was used as one of its IP cores.

A network processor is a packet-processing engine that performs routing and a host of other network-related functions. The typical tasks handled by a network processor include packet classification, forwarding, header update, encryption, traffic management, and quality-of-service (QoS) [31]. In fact, any device that provides some form of network services may be claimed to be a network processor. Despite the lack of a clear definition, most network processors do have the common attribute of enhanced programmability to offer better post-manufacture flexibility than the hard-wired ASIC switches and routers. The programmability is usually built into the modules at the higher OSI stack, where protocols are constantly being refined and invented. Most network processors also provide means to accelerate tasks that must be performed on a regular basis and thus is critical to the throughput of the system. This usually applies to functions at the lower OSI

stack where the protocol is standardized and can be implemented completely in hardware. For instance, by including a barrel shifter to facilitate the parsing of IPv4 packet headers, a network processor consumes 2000 extra gates but its performance increases by 30% [38]. As a result, a network processor is really a trade-off between the ASIC solution and general-purpose CPU approach, as shown by Figure 8. A network processor is a suitable candidate for studying SoC because the functionality requires a wide variety of logic and memory components, and thus provides an opportunity to explore core integration and test issues. Moreover, the bandwidth requirement of the network traffic demands an efficient communication path between the blocks, so a network processor is an ideal test bed for high-speed bus design.

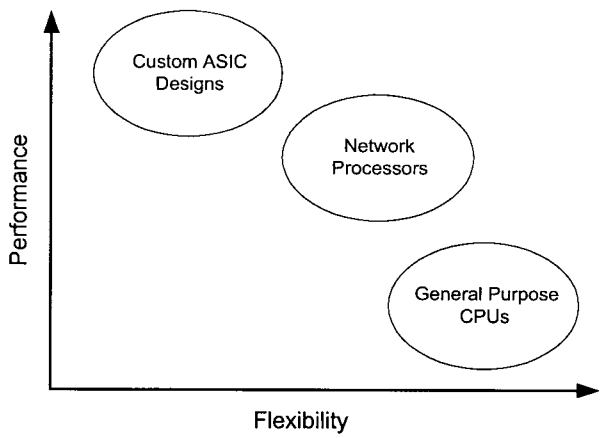

Figure 8. Performance and flexibility trade-offs of network function implementations

3.1 Architecture

An industry-strength network processor is an extremely complex system that is the fruit of years of research and development. Rather than designing such a complex system from scratch, an architecture was chosen from a commercial vender and scaled back to fit the

design capacity of the researchers. The architecture of the simplified network processor was constructed by consulting information gathered from [31][43][24][21][51], and was inspired by the Sitera network processor [50]. The simplified network processor unit (NPU) classifies and forwards IPv4 packets and has a rudimentary firewall functionality that relies on the classification result. The NPU was modeled after the Sitera architecture and has similar processing flow of classification, Layer-4-and-above functions, look-ups, and queuing. The system performs primarily OSI Layer 3 services. It is intended to be a router in local-area network for a corporate environment where the low-level protocols are stable and well defined. The design assumes that the incoming link is a 100Mbps Ethernet connection and the average packet size is 500 bytes. Then for it to achieve the proposed processing time of 4 cycles/byte, the NPU should operate at minimum of 50MHz. In comparison, a real network processor is usually targeted at gigabit links, has an average process time of 80 cycles/byte, and operates at 200~300MHz [38].

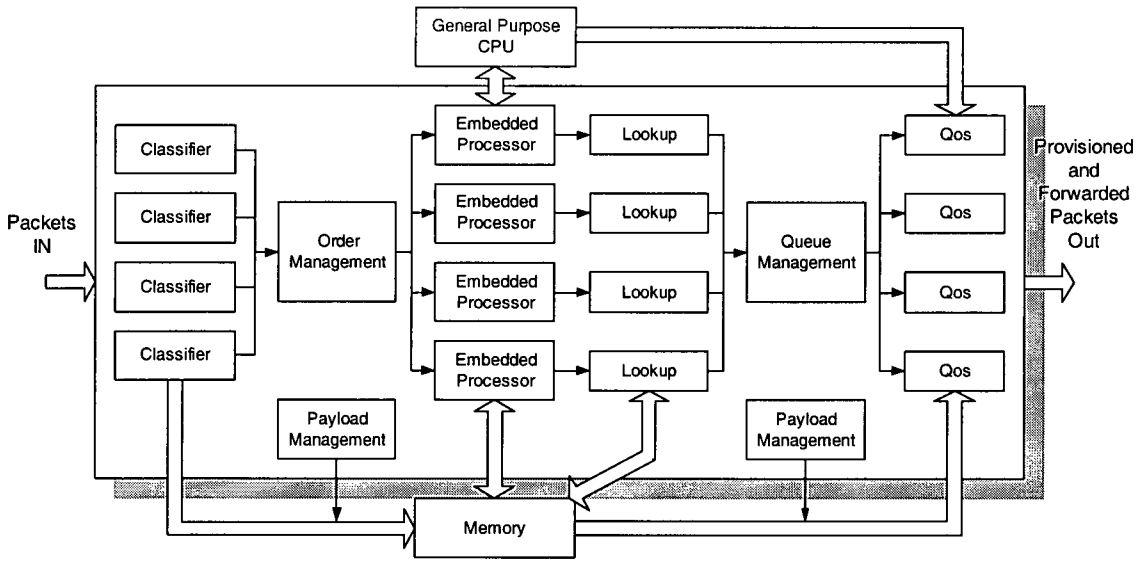

Figure 9. Architecture of the Sitera network processor

3.2 Components

The major components of the NPU include a pre-processing unit, a classifier, an embedded processor, a post-processing unit, and various memory components as shown in Figure 10. The embedded processor is the HC11 discussed in Chapter 2, and the other logic blocks are standard ASIC blocks. The embedded processor is a reusable IP block and the PCI interface is assumed to be available from a third-party vender, while the other logic blocks are considered as UDLs designed specifically for the NPU SoC. Although their functionalities are tailored for the NPU and the likelihood of reuse is low, the UDLs are still designed following the reuse guidelines. By conforming to the reuse guidelines, the blocks should have clean interfaces and good coding that help the task of integration and possibly the debugging process. The UDLs in the NPU, including the classifier, the pre-processing unit, and the post-processing unit, were all developed from scratch and designed with proper DFT for structural test. The components communicate to each other through either an AMBA^{TM}-compliant high-speed bus or through point-to-point connections. The AMBA-compliant bus was developed based on the AMBA AHB specifications publicly distributed by ARM [2]. The AMBA bus uses a pipelined design for data transfer that satisfies the bandwidth requirement of network processors. This bus will be described in more details in a section to follow.

The pre-processing unit is responsible for receiving inbound packets from an external physical layer (OSI Layer 2) device. The pre-processing unit separates the header of the received packet from the payload portion and sends only the header to the classifier. Its counter part, the post-processing unit, is responsible for assembling the header and the

payload of the packet to be sent out of the NPU. The pre-processing unit also notifies the classifier when an error occurs in the inbound packet. In the current implementation, the packet is discarded when an error occurs.

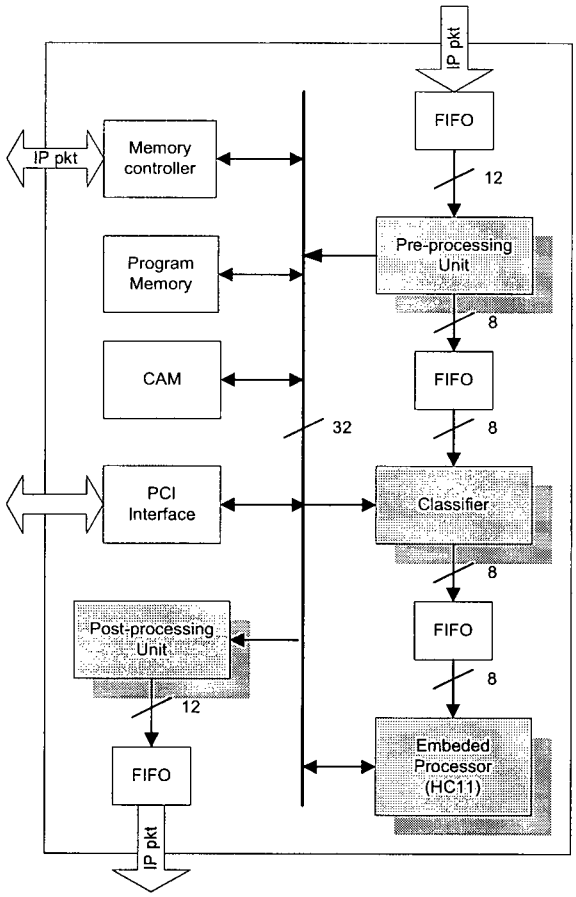

Figure 10. Architecture of the simplified network processor

The classifier accelerates the operation of the NPU by implementing the longest-prefix matching of IPv4 address in hardware. The IPv4 address in today's network is actually a combination of a network address and a host address on the network [35]. The prefix refers to the network address portion of the IPv4 address. For example, if the IPv4

address of a machine is 192.168.4.12 with a subnet mask of 255.255.255.0, then the 192.168.4 portion represents the network address and 12 indicates the host address of the machine. Adjusting the subnet mask allows the network administrator to trade off the number of networks with number of hosts on each network. Due to the masking, a given IPv4 address may match multiple networks, so the longest matching prefix is needed to resolve the aliasing. Some routers use a ternary content-addressable memory (CAM) with the IPv4 address as the CAM tag and the host address of the next hop as the data. Others use binary search trees or hash tables implemented in software [21][25]. In the NPU, the classifier narrows down the search space by identifying the flow, which is a range of IPv4 addresses. Each flow is represented by a class ID, which can be used later at the CAM for route look-up. This approach requires less CAM space and is faster then the software-based searching algorithm [25].

The classifier also allows information other than the IPv4 address to be used in making forwarding decision. In fact, the classifier allows arbitrary header field to be matched and thus enables differentiated services by matching the source/destination address and the transport protocol. Using a set of highly optimized classification rules rather a procedural language such as C to direct the operation, the classifier can perform faster then a RISC processor and the classification code is easier to debug than a C program [46].

The embedded processor in a real network processor is intended to process high-level functions at Layer 4 or higher. Examples of such functions include:

- Policy-based networking - exercises company policies such as priority definition, security enforcement, and route selection based on the user and the application
- Server load balancing - redirect the traffic among servers based on URL, server load, or user credential. An example is the proxy server that stores frequently accessed web pages
- Quality of Service - allow jitter-sensitive applications such as video and voice to have higher priority
- Network monitoring and analysis - high-speed capture of the network traffic to provide information for network planning and troubleshooting [1]

The above functions all require header information at Layer 4 or above. Currently there is no standardized protocol for any of these functions. It is unlikely that they will be standardized, since different network services providers desire different implementations of these functions to suit their business model. In the NPU, the embedded processor is only used for executing the classification results, which is explained later. The assumption is that if any high-level function is to be included in the future, it can be implemented as software and then executed by the embedded processor.

Once the components are integrated, the NPU can transport an IPv4 packet from the input port to one of the output ports based on the routing information in the packet header. The flow starts with the pre-processing unit receiving a packet in byte-wide chunks from a Layer 2 device. The pre-processing unit then stores the packet in the packet memory and copies the header portion of the packet to the classifier. The classifier processes the header according to the classification rules and produces a class ID and an action tag.

Using the action tag, the embedded processor can decide whether to further process the packet in software or simply allow/disallow the packet to flow through the engine. If the packet has the permission to flow through the engine, the embedded processor looks up the next hop by matching the class ID in the CAM and sends transfer request to the post-processing unit. One post-processing unit resides at each corresponding output port. Once a transfer request is received, the post-processing unit will transfer the packet from the packet memory to the output FIFO. The outbound packet will be retrieved by a downstream Layer 2 device and will continue to travel through the network.

Figure 11 shows a hypothetical network in which the NPU may be used. The network is designed to be such that the first 24 bits of the IP address identify the subnet, while the last 8 bits identify the individual terminals connected to the subnet. Table 1 shows a sample of classification rules for this network. The rules are to be executed one after the other in the order they are stored in memory, until one of them is matched. A rule is said to be matched when the header fields in the current packet matches all the corresponding fields in the rule. The rules in Table 1 were setup so the NPU filters out any Internet traffic (using HTTP transport protocol) from subnet 168.1.4.x to subnet 168.1.1.x and blocks any FTP requests to terminal 168.1.3.5, while allowing all other traffic to flow through the network. The NPU handles the tasks on the data path. The tasks on the control path, such as routing table update, are assumed to be performed by a host computer connected to the PCI interface.

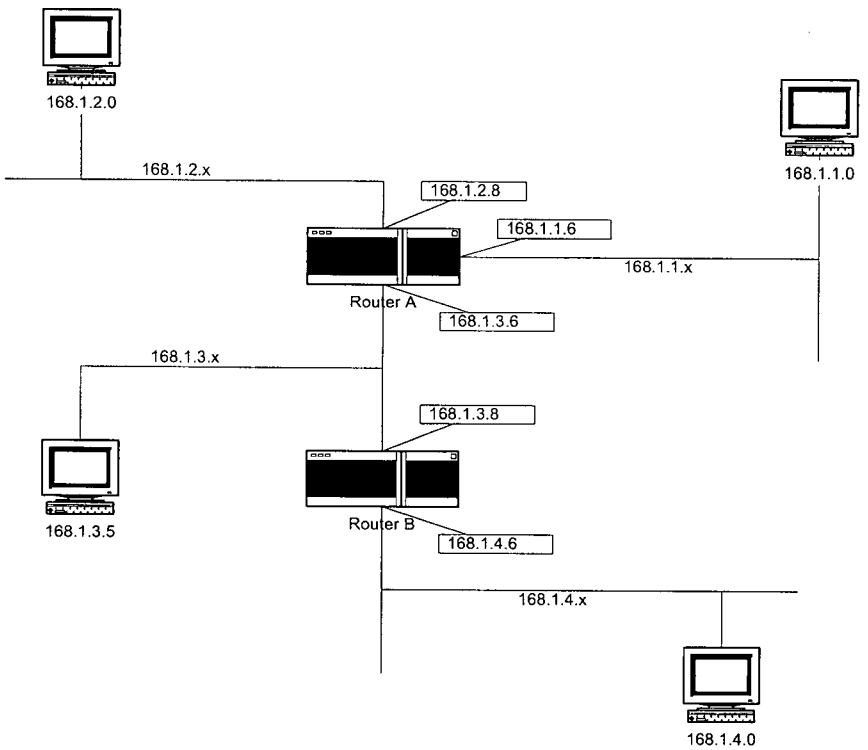

Figure 11. Hypothetical network

Table 1. Classification rules for the hypothetical network

	Layer 3 source	Layer 3 destination	Layer 4 protocol	Layer4 dest. port	Class ID	Action
Rule 1	168.1.4.X	168.1.1.X	TCP (6)	HTTP (80)	1	Deny
Rule 3	X	168.1.3.5	TCP (6)	FTP (20)	2	Deny
Rule 6	X	X	X	X	3	Permit

The X means don't care

The current design of the NPU shown in Figure 10 contains only one pre-processing unit and one post-processing unit to handle one input and one output channel respectively. Clearly a network router should be capable of handling multiple channels of inbound and outbound packets. Therefore, a real network processor should probably have multiple instances of the NPU design operating in parallel, with each instance being responsible for one input/output channel. Also the CAM module of the NPU for storing routing

information and a PCI module for connection to host computer were not available at the time the system was developed. An SRAM module is used instead as placeholder for the CAM. In addition, the HC11 core consists of only the central-processing unit and the supporting modules such as interrupt controller, local memory, and ROM controller are missing. Even if all the missing hardware were available, an embedded program still needs to be developed and loaded into a ROM accessible by HC11. As a result, the network processor is not functionally completed. Nevertheless, the current implementation is sufficient for the purpose of investigating core DFT designs and experimenting with different TAM approaches, which is the intended purpose.

3.3 System Bus

The system bus in the NPU is an AMBA-compliant, multiplexer-based bus. It is responsible for transporting functional data between the cores in the NPU. Although a multiplexer-based bus consumes more area, this type of bus is favoured in SoC design because it does not have the power-on and drive strength issues of a tri-stated bus [33]. Figure 12 shows the architecture of the AMBA bus. The architecture requires a distinction between masters and slaves. By definition, a master is a component that can initiate and respond to data transfer, whereas a salve can only respond to data transfer. Each data transfer consists of an address phase and a data phase. The bus design uses two types of multiplexers to accomplish data transfers: one for delivering address from masters to slaves, and the other for delivering data either from masters to slaves or from slaves to masters. The separation of address and data paths allows consecutive transfers to be pipelined and thus increases the bus performance [2].

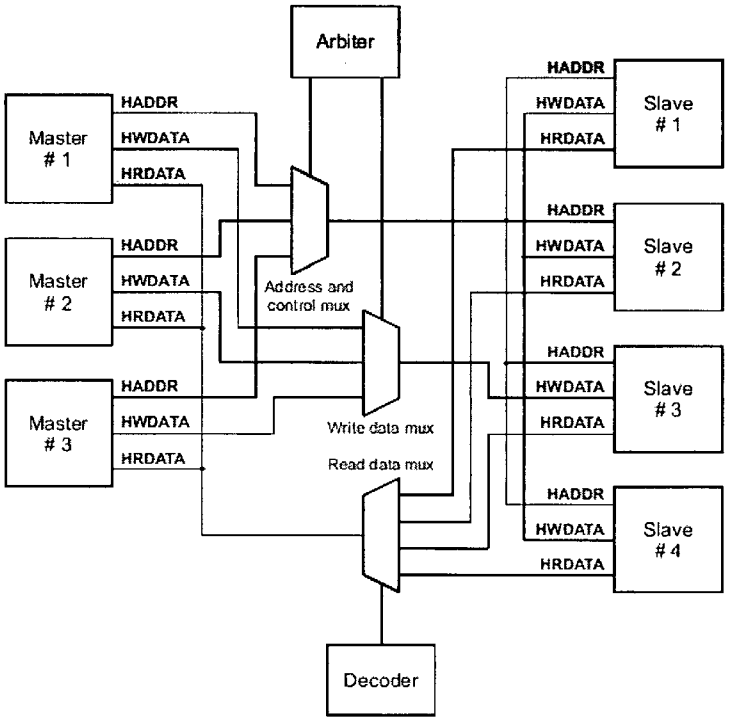

Figure 12. AMBA AHB architecture

The concept of bus adapters has been used in the integration of the cores with the system bus. The idea is that instead of connecting the cores directly to the system bus, adapters that are customized to individual cores are created to serve as intermediate interface modules between the cores and the bus. By using the adapters, the cores are shielded from the details of the bus interface and the design of the cores can be simplified. Figure 13 demonstrates the concept of the bus adapter. In the current implementation, adapters with generic interfacing rules are possible since none of the logic blocks in the NPU uses advanced AMBA bus function such as burst transfer, locked transfer, and split mode. And while the AMBA bus uses pipelined transfer, the NPU cores do not. The conversion between pipelined transfers and non-pipelined ones is handled within the adapters. In addition, the use of adapters facilitates the adoption of an alternative bus architecture in the future, as only the adapters would need to be modified and the cores could remain

unchanged. Although the development of the adapters demands more effort and may be error prone, using the adapters allows system integration and core development to proceed concurrently. The system integrator can design the adapters and test inter-block communication, while the core designers can focus on delivering the core functionality.

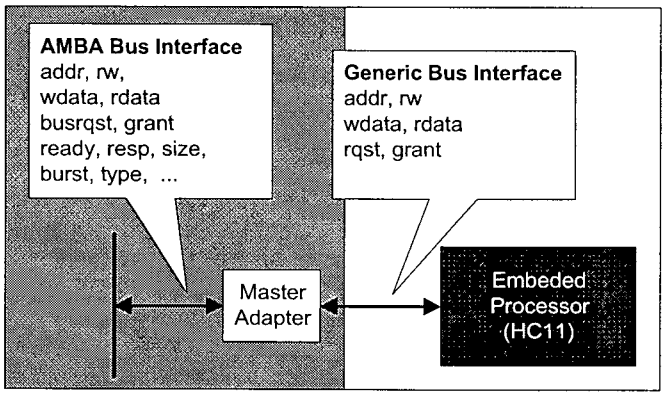

Figure 13. Concept of the bus adapter

3.4 Design for Test

For testing at the core level, full-can testing is deployed in the pre-processing unit and the post-processing unit. A single scan chain is inserted into each block, and scan vectors (generated by Synopsys Test Compiler) are to be supplied from off-chip. To create a heterogeneous testing environment, a BIST strategy is used for the other blocks. A BIST controller is implemented for the classifier and the embedded processor (HC11). When the BIST is active, it overrides the functional clock of the logic cores with a gated clock and generates the scan-enable signal from its internal state machine. The BIST can also be bypassed by asserting the appropriate control signal so the core can be tested by the full-scan method as well. Testing of the memory modules is to be performed by dedicated

memory BIST, which is different from the logic BIST used by the classifier and embedded processor. The memory BIST runs a simplified marching C algorithm and has rudimentary diagnostic support. The test data is transferred by an on-chip packet-switching network based on the NIMA concept presented [42]. NIMA will be described in a later chapter. Table 2 summaries the DFT strategies for the cores inside the NPU.

Table 2. Core DFT strategy

Core	DFT Strategy	Scan Length	Vector Count
Pre-Processing Unit	Full Scan	642	509
Post-Processing Unit	Full Scan	326	286
Classifier	Logic BIST	518	2048
Processor (HC11)	Logic BIST	181	2048
Program Memory	Memory BIST	N/A	129
Data Memory (CAM)	Memory BIST	N/A	129

For testing at the system level, P1500-compliant wrappers were developed to encapsulate individual cores and their associated test structures. P1500 is a standard under development by IEEE targeted specifically for embedded core testing [58]. Its purpose is to provide a uniform interface between the cores and the chip-level test access mechanism, analogous to how IEEE 1149.1 facilitates board-level testing. In fact, the P1500 wrapper is very similar to the legacy IEEE 1149.1 boundary scan in both architecture and operation. The most noticeable differences are the absence of TAP controller and the addition of parallel test port in P1500 wrappers. By detaching the TAP controller and providing more access ports, the serial-input constraint of IEEE 1149.1 is removed and a greater variety of test access mechanisms are supported. The wrappers for the NPU cores are developed based on the most recent information published by the

standard committee [45], with some minor adjustments to adapt to the NIMA test network.

The P1500 wrapper has four control inputs and one pair of serial data input and output as shown in Figure 14. The serial input, WSI, is used to transport wrapper instructions and test data. Instructions for the wrapper are shifted serially into the Wrapper Instruction Register (WIR) and various enable signals are generated from the control logic based on the content of the WIR and the four control inputs. The Core Data Registers (CDRs) are used to capture test results or to provide signatures to BIST and multiple-input sequence recognizer (MISR). The ring of flip-flop around the core forms the Boundary Data Register (BDR) that isolates the core's functional interface from other blocks in the SoC during testing. When exercising full-scan test on the wrapped core, the test vector is serially shifted in through WSI, and scan output is serially shifted out through WSO. If a MISR is instantiated, the scan output is also captured by the MISR and compacted to produce a signature. Both the core and the BIST support full scan, so the MISR can be used for compacting results from testing either module. Note that according to the IEEE 1149.1 standard, the serial output is stable at the falling edge of the clock. In an attempt to adhere to the established IEEE 1149.1 standard as much as possible, a negative-edge-triggered flop is added to buffer the WSO signal. The implication of using both clock edges is that the time window for data to propagate through the wrapper is reduced to half a clock period.

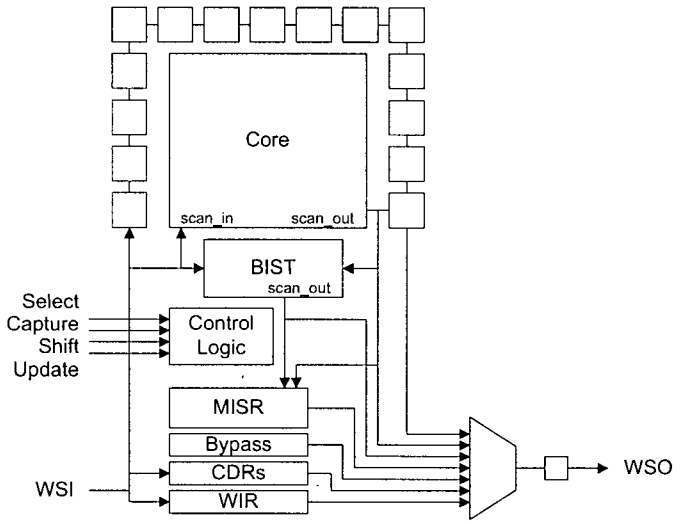

Figure 14. Block diagram of a P1500 wrapper for a core using BIST DFT

Depending on the requirement, the cell of the BDR may be simplified to reduce area overhead. The current design of the cell consists of two D-flip-flops and a multiplexer. The design of double flip-flop is derived from the IEEE 1149.1 standard [39]. In this design, the capture flip-flops provide stable data to the core while the new data is being shifted through the shift flip-flops, and thus prevent the internal state of the core from being altered unintentionally. To save routing area, reset lines are omitted for the D-flip-flops. To ensure that the core receives deterministic values from the wrapper cells, the core is connected to the functional inputs/outputs through the multiplexers during wrapper reset. The core inputs are connected to the capture flip-flops only after valid data are loaded using the wrapper PRELOAD instruction.

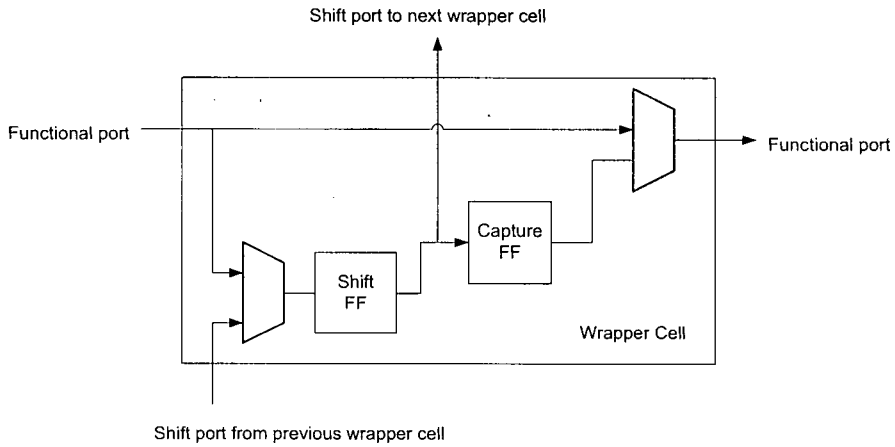

Figure 15. Structure of a wrapper cell

As shown in Figure 16, all blocks in the NPU except the bus module are wrapped by P1500. The bus module is not wrapped because the on-chip bus is not to be tested by the full scan method and also because adding BDRs around the bus will cause excessive area overhead. During testing, test data (either test vectors or commands to the BIST) is formatted into test packets and sent to the designated block. In order to translate the test packets into information that can be used by the test structure of individual blocks, a dedicated NIMA interface module is developed for each core. The NIMA interface modules, shown as square boxes attached to the P1500-wrapped cores in Figure 16, interpret the payload of the test packet and control the P1500 inputs based on the content of the payload. For instance, if the payload of the packet contains a test vector for the embedded processor, the interface module toggles the Select, Capture, Shift, and Update signals of the P1500 wrapper appropriately so the vector is shifted into the core through the WSI input.

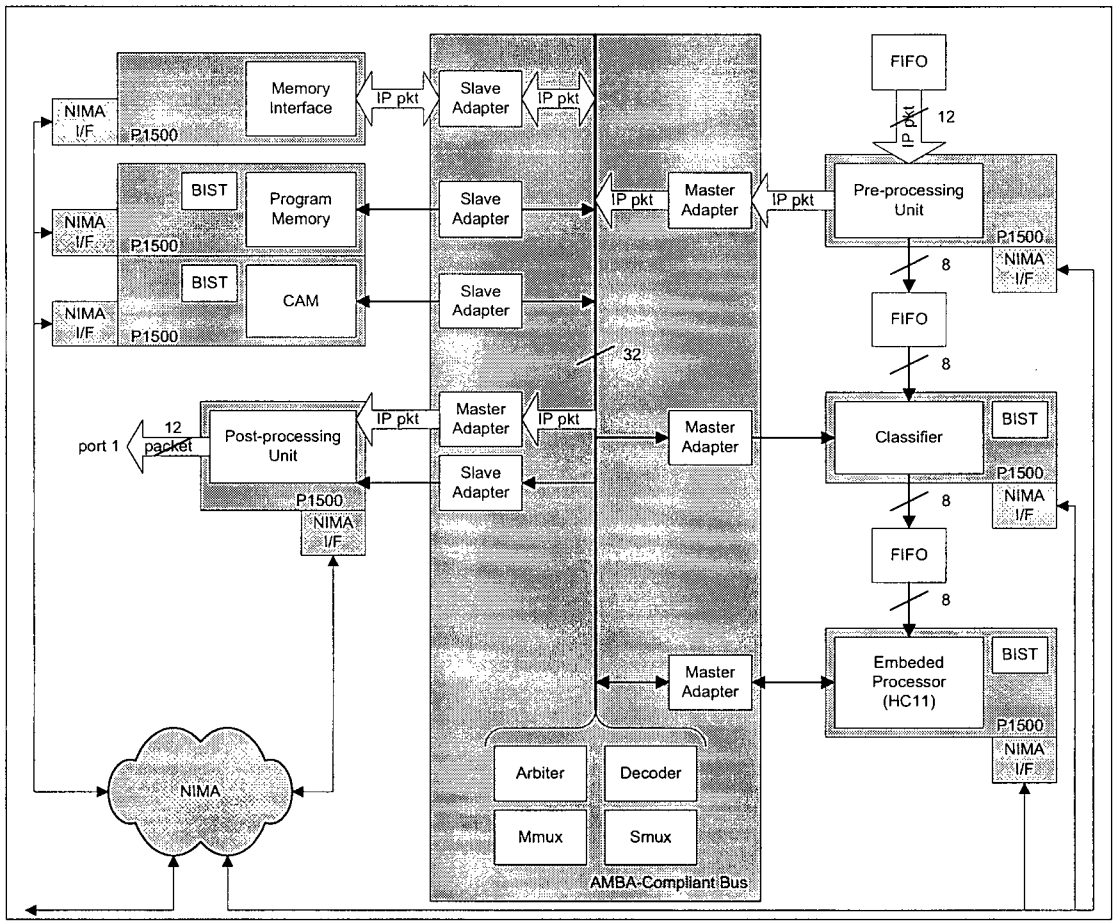

Figure 16. Network processor SoC with NIMA test network

3.5 Core Integration

Various integration issues surface during the integration process in which the network processor is constructed from the HC11 IP block and the UDLs. Since the cores are integrated as soft IPs, their RTL codes are imported into the Synopsys Design Compiler for synthesis. Proper timing and capacitive loading constraints need to be entered into the tool in order for it to synthesize the design to meet the targeted speed. For this project, area and power are not constrained and thus are not optimized. Although area and power

are not properly considered, there are still many integration problems common to today's IC designs that need to be solved. Among the important integration issues are I/O buffering of the cores, core verification, and synthesis strategy.

3.5.1 I/O Buffering

I/O buffering of all SoC cores is strongly recommended by RMM. As described earlier, adding I/O buffers has the benefit of localized timing. As the size and the speed of design increases, it becomes increasingly difficult for signals to propagate through the silicon chip within the given time window. Due to aggressive design goals, signals have to propagate through more stages of logic in a shorter clock period, and the risk of not meeting timing increases. By registering the I/O of the cores, the timing problem is localized in the sense that the delay through one core does not affect the delay through another one, as long as each core meets its own timing budget. Then a discrete number of clock cycles is allocated for the communication between the cores. This is important for reuse since the designer of one core does not necessarily have control over another core, so each core must achieve timing closure independently.

Registering I/O, however, has adverse effect on inter-core communication. Often the inputs to a core are used to drive state machines or generate control signals of a data path. By registering the input, one clock cycle of delay is introduced between the time when the inputs become valid and when the core responds to the inputs. Similarly, by registering the output of a core, the downstream core would not receive the new data until one cycle later. The extra cycle disrupts the synchronization of data between the cores

that are closely related. An example in the NPU SoC is the interaction between the FIFO and the classifier that reads from it. After the classifier reads the last piece of data from the FIFO, the FIFO sets the empty flag. However, the flag is not detected immediately by the classifier as a result of I/O buffering, so the classifier may continue to read from the FIFO and mistaken the default value as valid data. The remedy currently employed is to make the classifier read the almost-empty flag instead of the empty flag and set the empty threshold in the FIFO properly so there are enough clock cycles for the flag to propagate to the classifier. One may argue that the FIFO should be integrated closely with the classifier and thus eliminating the need for I/O buffering. The counter argument is that by moving the FIFO close to the classifier, the core that writes to the FIFO sees a longer path and their interconnect delay there could be a problem. Furthermore, since the testing strategy for the FIFO has not been defined, integrating the FIFO into the classifier complicates the DFT strategy of the core.

3.5.2 Core Verification

Inconsistent core implementation is a problem that constantly impedes the integration process of the NPU. Since different cores were designed by different engineers, inconsistency occurred when wrong assumptions were made regarding the inter-dependency of the cores or when the design specifications were misinterpreted. In other times, the inconsistency was simply the result of the design errors. Common errors that are obscure and hard to detect include:

- Errors when translating the code between Verilog and VHDL. For example, the '&' operator is valid in both HDL but have different meaning. In Verilog, it means the

logic AND function; in VHDL, it means literal concatenation. Misuse of the operator results in syntactically correct but semantically incorrect code.

- Constants that are not updated when the design specification changes. Hard-coded values are often used for length of bit vector, default values, and the mask for bit masking.

As mentioned in Chapter 2, an IP block should be fully verified before integration. However, achieving 100% code coverage is not as easy as it sounds. Generating a good set of testbench and test vectors could take longer than developing the core itself. From our experience, the extra cycle in testbench development extends the development time by 2 to 2.5 times. In addition, calculating the code coverage requires support from the RTL simulation tool. The NCSIM™ Verilog/VHDL simulator from Cadence has such a capability, but we lack the license for advanced coverage report, which is needed to accelerate testbench development. During the integration of the NPU, most of the inconsistencies caused by misinterpretation were resolved in design phase rather than during integration, thanks to close interaction between the system integrator and the core designers; 60% of the inconsistencies during integration were caused by designs that were not tested properly and thus still contained design errors when handed over. Behavioral synthesis could be useful here since it eliminates most of the manual work in RTL coding and also facilitates the creation of an executable specification that can be used in formal verification.

3.5.3 Synthesis Strategy

Developing a coherent synthesis strategy is a complicated process. Design Compiler from Synopsys is capable of many different synthesis approaches, including top-down, bottom-up, and a combination of these two approaches [13]. The selection of a synthesis strategy has a direct impact on the optimization results of the design and the run-time of the CAD tool. In the top-down approach, all design entities are read into the tool at once and the compilation process takes care of the inter-block dependencies automatically. However, top-down approach is possible only if there are adequate memory and CPU resources. The bottom-up approach synthesizes one design at time starting from the entities at the lowest end of the hierarchy. The bottom-up approach requires less memory than the top-down approach and allows timing budgeting, and thus is the suitable choice for most modern designs. Its disadvantage is that multiple iterations are usually required to achieve stable inter-block interfaces. RMM recommends a bottom-up approach for SoC design. This approach mandates a set of synthesis scripts for each core to be written so the core can be quickly re-configured and re-synthesized for use in different projects.

Currently the NPU is synthesized from the cores using the bottom-up approach. In this flow, the core and its BIST module (if present) are synthesized separately with proper timing constraints applied to each of them. For this project, area and power are not important and thus the clock definition and the I/O characteristics are the only constraints specified in the synthesis script. Once the core and BIST are synthesized, they are set to *don't-touch* to prevent further modifications, and a top design is created to instantiate the synthesized netlists. At one level of hierarchy above, another entity is created to

instantiate the top design and the P1500 circuits. The synthesized, P1500-wrapped cores are then referenced during the synthesis of the NPU. Figure 17 demonstrates the design hierarchy from core up. At the design hierarchy below the core level, the bottom-up strategy becomes inefficient in terms of chip area and code organization; collapsing all sub-blocks into a single module and synthesizing it as a whole becomes a better approach when the design is small. In this top-down approach, the timing characterization is performed on the top-level module and then propagated down to the sub-blocks by the CAD tool. Then during synthesis, the boundaries of the sub-blocks are removed, allowing better area optimization by the CAD tool.

Since the cores are synthesized using a bottom-up approach, preliminary driving and loading constraints (i.e., boundary constraints) are applied for the first synthesis attempt. The driver is assumed to be a typical two-input NAND gate, and the load is assumed to be a D-flip-flop. The correct driving and loading parameters are captured by the *characterize* and *write_script* commands during integration of the core. The core should then be re-synthesized with the correct constraints and re-integrated. This flow is described in [13]. The NPU synthesis was successful and the only timing violations were related to the clock network. Since the clock trees are inserted only in the back-end flow, the synthesis tool can only estimate the clock delay based on user inputs and the wire model from the library. By specifying an ideal clock network that has large drive strength (which is a normal assumption for pre-layout synthesis), the violations were eliminated. Timing closure could be attributed to the I/O buffers; without the I/O buffers at the core

interfaces, many long paths crossing the core boundary would have delay problem that may require iterations of synthesis to resolve.

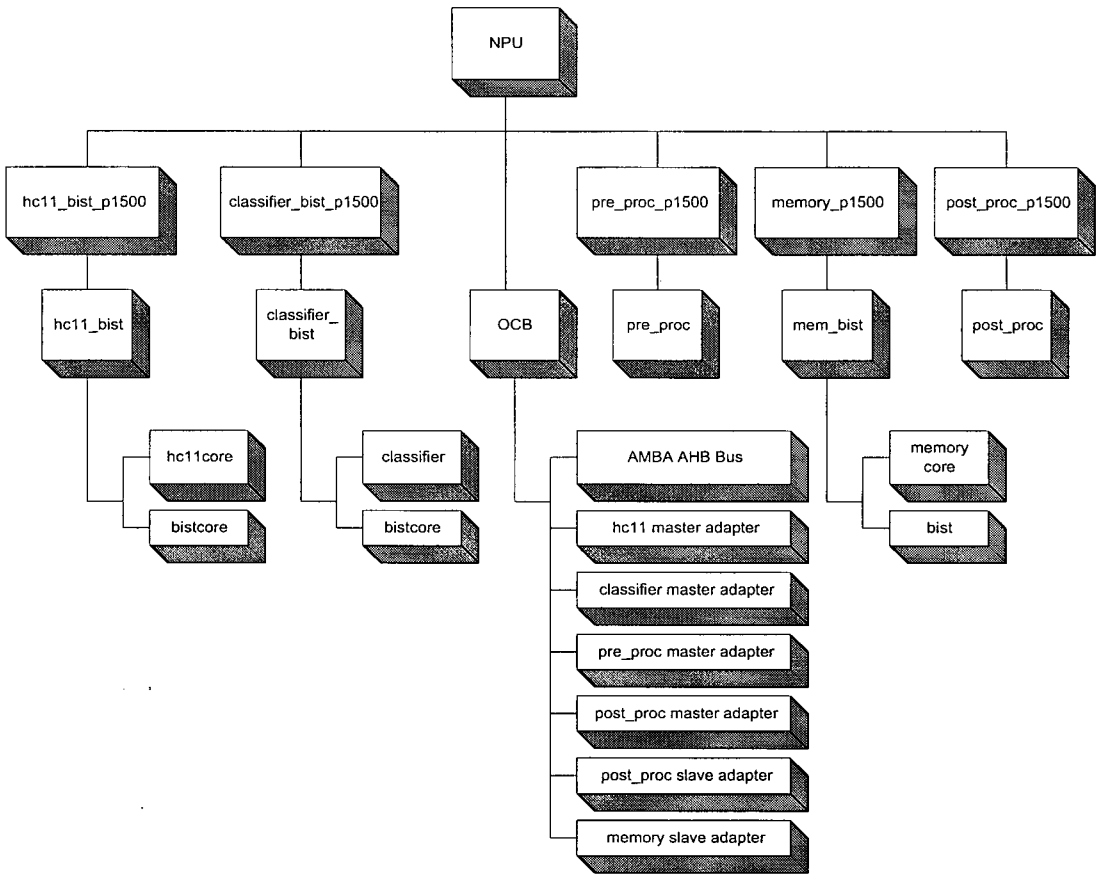

Figure 17. Network processor module hierarchy

3.6 System Verification

System verification is an important part of the SoC design process and it is perhaps the most difficult part. Today's SoC consists of many components that need to perform complex functions individually and also need to interact with each other seamlessly. The traditional verification approach for ASIC design is to apply functional test vectors as

stimulus to the input and then compare the output against some expected values [32]. This requires many test vectors to implement the different test cases of the core. Checking the correctness of the components by toggling the inputs and observing the outputs is an extremely difficult task in an SoC environment. Not only are the signals embedded (controllability and observability problems), but also the sheer number of signals makes the task unmanageable. The suggested verification approach for SoC is to abstract the operations of the cores into transactions [32][40]. For instance, the read/write signal, the bus request/grant signal, and the address bits can be grouped as a 'read' transaction on a system bus. Validating a transaction involves checking the toggling sequence of the read/write and bus control signals and determining whether the address in the transaction is within the permitted range. The transactions should be automatically checked by some monitor entities to ensure that there is no illegal transaction. For a system that uses a bus for inter-block communication, the bus interfaces are the ideal locations to implement such transaction monitors [33]. In other words, the bus interfaces can act as the control and observation points for functional verification of the SoC.

For the NPU SoC, the system is partially verified by simulating the bus adapters. A Verilog testbench that instantiated the bus adapters and the AMBA-compliant bus was developed. The testbench models the actual cores and sends appropriate control/data signals to the adapters to initiate transactions between the cores. The transactions were then verified by manually inspect the waveform of the signals at the receiving end of the transactions. The following transactions that will occur during the normal operations of the NPU have been verified:

- The pre-processing unit writes to the packet memory
- The classifier reads from the program memory
- The embedded processor reads from the packet memory
- The embedded processor reads from the CAM
- The embedded processor writes to the post-processing unit
- The post-processing unit reads from the packet memory

The assumption here is that the cores have been implemented correctly according to the specifications and are verified individually using core-level testbench. Due to the time constraint of the project, system level simulation involving the actual cores has not been performed. In addition, random test cases such as the transactions in arbitrary orders, and corner cases such as when the embedded processor performs a write transaction to the post-processing unit while the post-processing unit is busy, need to be covered to ensure that there is no test escapes. Furthermore, there are point-to-point communication paths between the cores that do not make use of the AMBA bus and the adapters. These interfaces need to be tested as well.

CHAPTER 4 SOC TEST STRATEGY

The problem of testing an SoC can be partitioned into two sub-problems: the provision of resources for conducting the test (i.e. the test access mechanism and the test controller) and the test scheduling. The design goal of the test access mechanism is to provide access to the embedded core from the chip I/O pins while at the same time minimizing the impact on the test performance and the performance of the SoC. The purpose of test scheduling is to devise an efficient test schedule that optimizes the test application time and also constrains the power consumption during test [36]. The TAM design and optimization is a hardware problem that involves logic design, interconnect issues, circuit techniques, and logic integration. The test scheduling, on the other hand, is largely a software problem that can be solved by mathematical theories and modeling. This section first describes the existing researches on TAM designs, analyzes their pros and cons, and then presents the implementation of the TAM for the NPU SoC described in previous chapter. This is followed by a discussion of the scheduling algorithm for the particular TAM integrated into the NPU SoC.

4.1 Test Access Problem

Most experts agree that test access is a major challenge in SoC design [36]. In traditional ASIC design, the terminals of the IC are all connected to the I/O pins of the manufactured chip, so an ATE can observe and control the signals on the design terminals. In SoC, many of the cores interfaces are buried within the system; only the cores with interface to the external system have terminals connected to the chip I/O pins. In addition, functional

and structural testing of the cores both require sending test vectors to the cores and receiving responses from the cores. Not only does the volume of test data increase due to larger number of cores integrated into the system, but also the delay problem (due to interconnect coupling of wires in close proximity and increasing length of global wire) of SoC makes reliable transportation of test data more difficult. The large number of bit streams for testing an SoC also means that the memory capacity of the ATE must be increased, which translates to higher overall test cost. Furthermore, as IDDQ testing becomes less and less effective due to increasing sub-threshold currents, alternative test method such as at-speed (AC) testing gains importance. Supporting AC testing on the traditional scan chains, however, require higher performance ATEs and printed circuit boards.

Besides the standard IC design parameters of performance, area, and power, design of the TAM also influences other factors such as test application time, quality of the test, and the reusability of the test program. Long test times translate into expensive tests, since the time the chip spends on the tester determines the test costs. Low-fault-coverage test due to flawed TAM design risks faulty designs to pass the manufacture test. The quality of the product will suffer and the defects may only be detected by expensive field testing or worse, field failures. In the reuse paradigm, the IP cores may be designed by a third party and delivered in hard form or encrypted for IP protection. In this case the core integrator must treat the IP cores as black boxes and rely on the core creator to put in the necessary DFT structures and provide the test program. Since yesterday's IC chip may be today's legacy IP, and today's SoC may be tomorrow's IP block, reusing the existing test

method/program represents significant savings in SoC development. As a result, the test time, test quality, and test reuse may significantly affect the overall cost of the SoC and its derivative products, and should be considered carefully. In addition, test integration is an important factor in the design of the TAM, as today's SoC is likely to use a combination of scan and BIST methods. Even within the scan method, there are different techniques such as multiplexed D-flip-flop and LSSD, each with different control mechanisms. There is no guarantee that a given IP core will use a particular DFT method, or if the IP core comes with DFT at all. Therefore, a TAM design must be flexible enough to support a wide variety of DFTs.

Among the TAMs that are presented in the literature, the proposed designs can be categorized based on the method of connection from the chip I/O pins to the cores. The simplest of all TAMs is the multiplexer TAM, which use multiplexers to share chip I/O pins among the cores. The multiplexer TAM suggested by [27] is easy to design and operate. On top of the multiplexers, the scheme uses AND gates for test isolation and OR gates for merging core outputs. It maps the core interface to a sub-set of the chip I/O pins, so both digital and analog test can be applied through the ATE. Debugging is straightforward since the active core can be isolated from the non-active core using the AND gates and can be directly accessed from the package pins. The disadvantage of the multiplexer TAM is that it is not scalable. As the size of the SoC increases, more multiplexers and associated control inputs are required, and this increases the routing overhead rapidly. Although at-speed test is possible, delay and/or skew caused by the multiplexers need to be taken into account. In addition, the multiplexer TAM cannot test

interconnects between the cores and may not be applicable when the number of the core terminals exceeds the number of chip I/O pins.

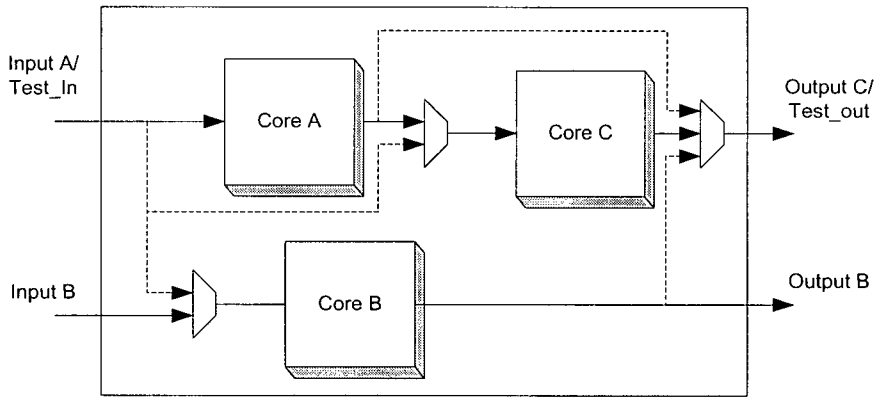

Figure 18. Multiplexer TAM. The dotted lines denote the test paths

The serial TAM is based on the established IEEE 1149.1 standard for board testing. The same IEEE 1149.1 structure is redesigned for chip testing [64]. In the proposed architecture, a scan wrapper envelops the core, and the serial output of one core is connected to the serial input of another. The same IEEE 1149.1 TAP controller can be used for core test after additional circuitry for coordinating the activities of the cores are added. The serial TAM approach has a very small routing overhead comparing to the other TAMs. Although the serial TAM reuses the IEEE 1149.1 standard and occupies very few pins for testing, it is not suitable for SoCs with more than a few cores as the test time using the serial test input would be prohibitively long.

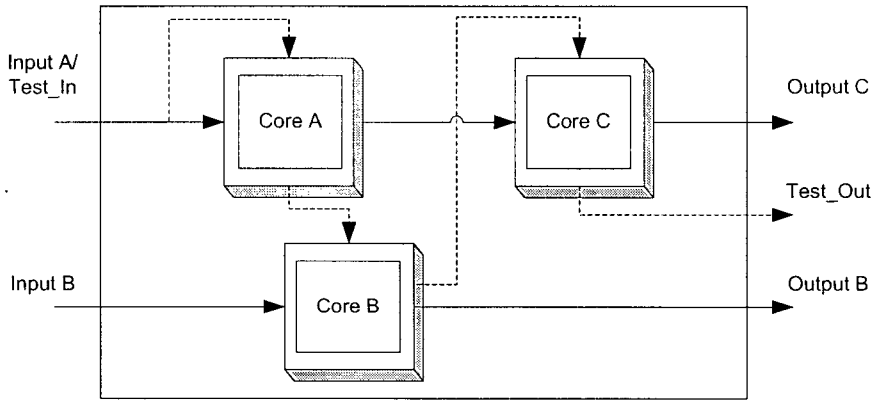

Figure 19. Serial TAM. The dotted lines denote the test paths

References [54] and [63] report different variants of the bus TAMs. The bus TAM uses an n-bit wide data bus for transferring test data to and from the chip, and an m-bit wide control bus for transferring enabling signals that determine the access of the cores to the data bus. The bus TAM essentially brings the core terminal to the chip I/O pins via the data bus. The core terminals are connected to the test bus through various access control logic that isolates the core from the bus when it is not under test. The bus TAM provides some degrees of scalability by allowing the bus width to increase with the increasing number of cores. The bus TAM also permits digital, analogue, and at-speed test. The limitation of the bus TAM is that it allows only one core to gain access of the data bus at a time. As a result, tests that require multiple cores running in parallel cannot be performed. Although it is possible to use a functional bus as a test bus as suggested in [63], the approach is ad-hoc since there is no guarantee that the width and/or direction of the functional bus would satisfy the requirement of the core test.

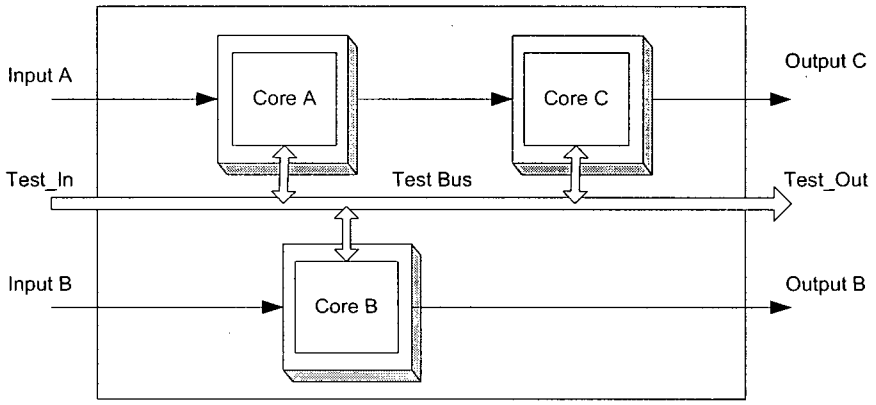

Figure 20. Bus-based TAM

Transparent TAM is based on hierarchical testability analysis (HTA) to obtain core transparency so that tests can be performed through normal functional paths [19]. The idea of HTA is to start solving the testability problem at the design phase. Code generators have been developed to produce cores whose internal nodes are easier to control and observe than cores that are handcrafted. The procedure proposed in [19] leverages the HTA code generator to obtain cores with high testability. If the result of HTA does not produce enough transparency, several techniques are described for establishing transparent paths by adding hardware such as multiplexers, observing flip-flops and latches, etc. During SoC testing, the transparent path of one core serves as transport media for the adjacent cores. The transparent TAM has low area and delay overhead, but the technique results in ad-hoc test procedures as the definition of the test ports differ from one core to the other. Moreover, the technique cannot be applied to all cores; it is particularly not suitable for the microprocessors or other data-path intensive circuits.

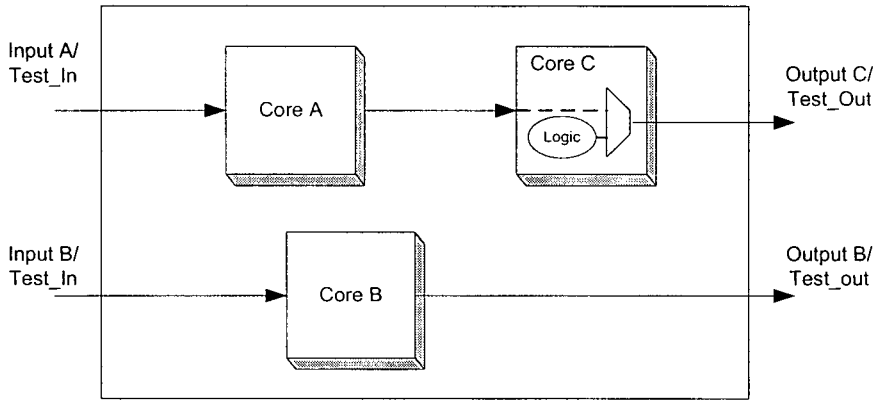

Figure 21. Transparent TAM. The dotted line denotes the internal test path

The Test Rail TAM presented in [37] is a refinement of the test bus architecture. Similar to the bus TAM, an n-bit bus is provided to transfer test data. Unlike the bus TAM, the n-bit wide bus can be split into smaller buses for cores that do not require that much bandwidth and later merged for cores that do require the bandwidth. The Test Rail thus offers more flexibility and allows core integrator to optimize the TAM either for area or for test application time. A layer of interface called Test Shell is required to surround the functional interface of the core. The purpose of the Test Shell is similar to that of the P1500 wrapper; both provide test isolation, test bypass, interconnect test function, and the capability of capturing a snap shot of the core I/O. The Test Shell is regulated by a set of dedicated control signals shared by all cores. The width of the Test Rail for a particular core is determined after considering such factors as available chip I/O pins, test application time, and silicon area. A methodological approach for calculating the minimal Test Rail width and the optimal distribution of the total test width among the SoC cores is given in [8]. The strength of the Test Rail is its large number of possible configurations. In fact, the Test Rail is the generalization of the multiple-scan-chain architecture widely used in the industry. The weakness of the Test Rail includes the extra control signals that

add to area overhead and the extra effort to calculate the optimal width and bandwidth distribution. Pre-processing and post-processing of the test data is also required in order to format the IP test vectors into a system test program and to extract the test results.

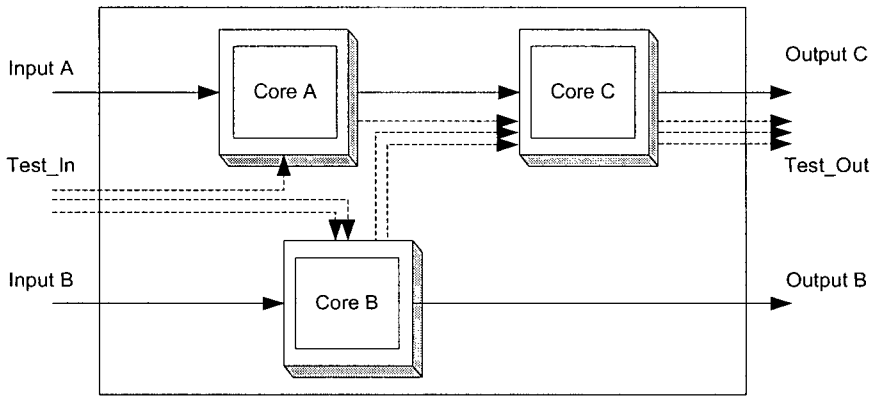

Figure 22. Test Rail TAM. The dotted lines denote the Test Tail

Finally, a novel approach using an on-chip network to deliver test data is proposed in [42]. The network-on-chip TAM, known as NIMA, adopts the packet-switching concept in computer network and applies it in SoC testing. The cores are assumed to have test harnesses such as the Test Shell for Test Rail or the P1500 wrapper already in place. The wrapped cores are then connected to the network through NIMA interface modules. The network itself consists of routers and wires that link the routers together. The on-chip test network is a promising TAM architecture for two main reasons. First, many researchers agree that layered network design is the best approach for global (long-wire) communication of future SoCs [49]. A structural network allows the electrical properties of the wires to be optimized and controlled, and also provides better utilization of the wires [4][12]. Second, the network is essentially a solution to a communication problem, so it does not make any assumption on the application that uses the communication

service. As a result, a network TAM inherently supports any kind of test strategy, whether it is full scan, BIST, scan complementing BIST, or any on-chip vs. off-chip source/sink combinations.

4.2 NIMA Design

The key concept in NIMA is to establish an indirect digital communication between the source/sink and the cores using packet-switching connections. For IC testing using this method, test vectors are converted to packets before they are sent to the cores for which they are intended. Test results can be sent to the test sink in a similar fashion. The design of the network is based on the network stack paradigm established in the telecommunication field, namely the OSI 7-layer model. This formal approach decomposes the design problem into a set of simpler, traceable, and modular tasks. Each layer of the network can be individually optimized to meet the particular power, area, and performance requirements of the SoC test network. Since the tasks involved in SoC testing are well defined and not as diverse or complex as those in computer networking, the OSI 7-layer model can be simplified to a 3-layer model consisting of a *Physical Layer*, a *Network Layer*, and an *Application Layer* as shown in Figure 23.

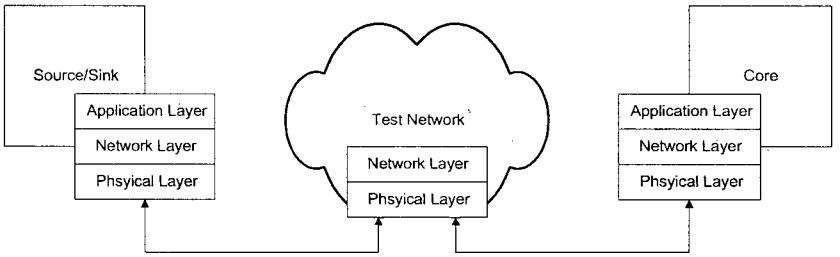

Figure 23. The 3-Layer Model of NIMA

4.2.1 Physical Layer

The physical layer deals with the actual medium for interconnection. For contemporary ICs, the physical medium is the metal wire that is routed between the SoC blocks. The physical level specifies the voltage level of the signal on the wire, the timing of the signal events, the signaling techniques, and other physical properties of the link, such as protection measures against crosstalk. In the end, the physical layer presents the data exchanged between the SoC blocks as a stream of 1's and 0's to the upper layer.

In the OSI 7-layer model, a data link layer sits above the physical layer to handle error detection and access control. We omit this layer in our design because we believe the reliability of the physical link can be maintained with proper circuit techniques (e.g., buffer insertion to mitigate delay problems and shielding to protect victim wires against aggressor crosstalk noise). For access control, the problem can be alleviated to one of packet scheduling in software since the data traffic in SoC testing is predictable. In other words, the SoC designer has complete control over when a block should have access to the network during testing.

4.2.2 Network Layer

The network layer dictates the details of how data is transmitted across the network. Information in this layer includes the switching technologies and network topology. In our design, the network layer is based on virtual indirect connection provided by micro-routers. To simplify the network layer, packets are sent in a particular order and are required to reach their destination in that order. To reduce transmission delay in the network, we adopt wormhole routing in which the packet is forwarded to the output as soon as the destination is known, without waiting for the tail of the packet to arrive [22]. To minimize the need to maintain routing information, the packets are routed using source routing, meaning the route is predefined. In addition, the network layer specifies the synchronization scheme of the packets. Asynchronous packet transfer is possible and is likely to be the way of future as centralized communication scheme becomes increasingly difficult to achieve in a large SoC design [4]. We opt for a synchronous approach instead as it is still applicable in today's SoC and has better support from the existing CAD tools. The synchronous scheme implies that the routers must operate within the same clock domain.

The network architecture assumes a single source for sending out test packets and a single sink for collecting the returning packets from the cores as in the original proposal suggested by [42]. However, the NIMA packet format has been modified from the original proposal in order to support a scalable implementation of the NIMA network. The original NIMA has only a width of one, and it is deemed insufficient for handling large test traffic. The improved version used in this project increases the network

bandwidth by changing the width from 1 to an arbitrary number. The packet format has been changed accordingly, and the current format is depicted in Figure 24.

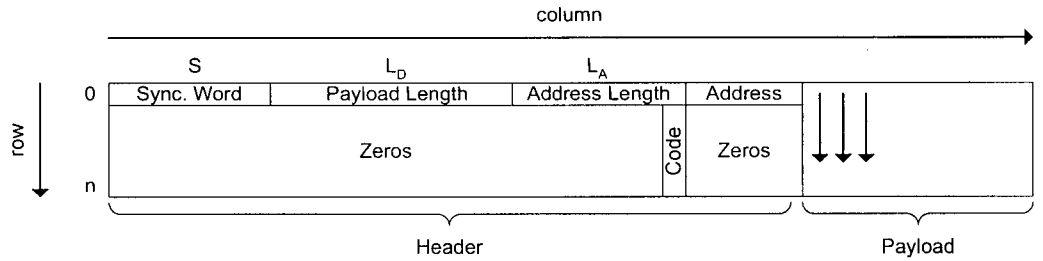

Figure 24. Revised NIMA test packet format

The packet is best described as a two-dimensional array. The row represents the width of the packet, and the column represents the length of the packet. The synchronization word, the payload-length field, the address-length field, and the address field must reside in row 0 of a packet. As shown in Figure 24, S, L_D, and L_A denote the length of their corresponding fields. These values are the dominating factor in determining the size of the packet header, and should be designed carefully to minimize overhead. The unused spaces in the packet header are filled with zeros. The following is a brief description of each field in the header:

- Synchronization Word – signals the beginning of a packet
- Payload Length – stores the length of the payload. The addition of this field to the original format to allow the payload to have an arbitrary length
- Address Length – stores the length of the address. Since the address must be a multiple of two bits (related to the fanout of NIMA router), the address length is actually the number of bit pairs.

- Address – destination address of the core for that packet
- Code – binary code used to generate the channel-ready signal in the last router

The payload portion of the packet contains test information such as control bits to the NIMA interface module, instruction to the P1500 wrapper, test vector to the embedded core, and control bits to the BIST, etc. The bits of the payload must fill up each column from row 0 down to the last row before the next column can be used. This rule is required because the decoding logic in the NIMA interface module reads the packet in a column-by-column manner. The rows of the packet must be properly aligned so that row 0 is applied to the least-significant bit of the test port, row 1 to the second-significant bit, and so forth until row n is applied to the most-significant bit. This rule is derived from the fact that the NIMA router assumes the useful header information to reside only in row 0.

4.2.3 Application Layer

The application layer in our design is an aggregation of the 4th to the 7th layers in the OSI model. It defines the protocol for accessing the network. At this level, the data is comprised of test vectors and DFT control signals that have to be converted to/from packets. The network architecture allows arbitrary bit width for the channel, so the packet could be a block of two-dimensional bit arrays instead of a one-dimensional bit stream. As the result, the blocks that wish to use the network must have the capability to scale the packet payload into the bit width suitable for its application. For packets wider than 1 bit, special ready signals are needed to indicate which bits are valid, as the data may not completely fill the bit array.

Since the cores could come from different design team at different organizations, a standardized test interface is critical. As mentioned before, the P1500 is currently under development by the IEEE with the intention of being the *de facto* standard. In this approach, core tests are to be carried out using P1500 instructions. Custom P1500 instructions are required to support full scan test as well as any other DFT strategies. Furthermore, a mechanism is needed to generate the P1500 control signals in the correct sequence at the core side. A set of wrapper control flags was devised to serve as instructions for this control mechanism. This results in two additional hierarchies in the message that is to be "packetized".

4.3 NIMA Implementation

To validate the concept, the network TAM NIMA was integrated into the synthesized NPU design. NIMA is a layered design that offers modularity and flexibility. However, redesign of the original NIMA was necessary to achieve a scalable solution. The NIMA network as proposed in [42] uses a single test port and the test packets are serially transported across the chip. It was found that the serial design can limit the bandwidth of the NIMA network and does not allow the full potential of NIMA to be realized. Fortunately, the NIMA architecture is highly flexible and it is fairly straightforward to extend the width of the network to multiple bits. The only complication is that with the increase in bandwidth, buffers need to be introduced.

For the *Physical Layer*, we use the traditional metal interconnect as the physical medium. The voltage on the wire is expected to swing between 0V and 1.8V, which is a typical range for the targeted 0.18um technology. The power consumption of the SoC during testing is well below the budget, based on the estimated calculation done by Power Compiler, so no special signalling technique is used. For process technology 0.18μm and below, the coupling capacitance between neighbouring nets dominates over the ground capacitance [52][23]. As a result, crosstalk is an issue that needs to be considered. Instead of implementing guarding wires around the NIMA nets, we decided to route the wires as part of a standard design flow and use the built-in capability of the routing tool to check for crosstalk violations. The target chip has sufficient real estate and the wires can be spaced apart to reduce crosstalk.

For the *Network Layer*, the analysis of total test time and scheduling showed that a NIMA width of 4 is sufficient for the current NPU implementation. The NIMA router is the basic component of the network. The current router implementation supports one input channel and four output channels (fan out of 4). Based on the number of cores within in the SoC and the length of test vectors for the cores, the parameters of the packet header were set as following: $S = 6$, $L_D = 10$, and $L_A = 6$. These parameters were hard-coded in the RTL code of the router. To integrate the NIMA network with the NPU SoC, a top-level design named NPU_NIMA was developed and it instantiates the NPU, the NIMA interface modules, and the NIMA routers. The NIMA_NPU design has the routing configuration shown in Figure 25. Note that the cores using full scan DFT require large amounts of test traffic from an off-chip supplier, so they are connected to the highest-

level router to minimize delay through the network. The current implementation does not have a network for transporting test traffic from the embedded cores to the off-chip test equipment (i.e. the sink), so the wrapper scan output of the six cores are connected directly to the chip I/O for observing the activity during test.

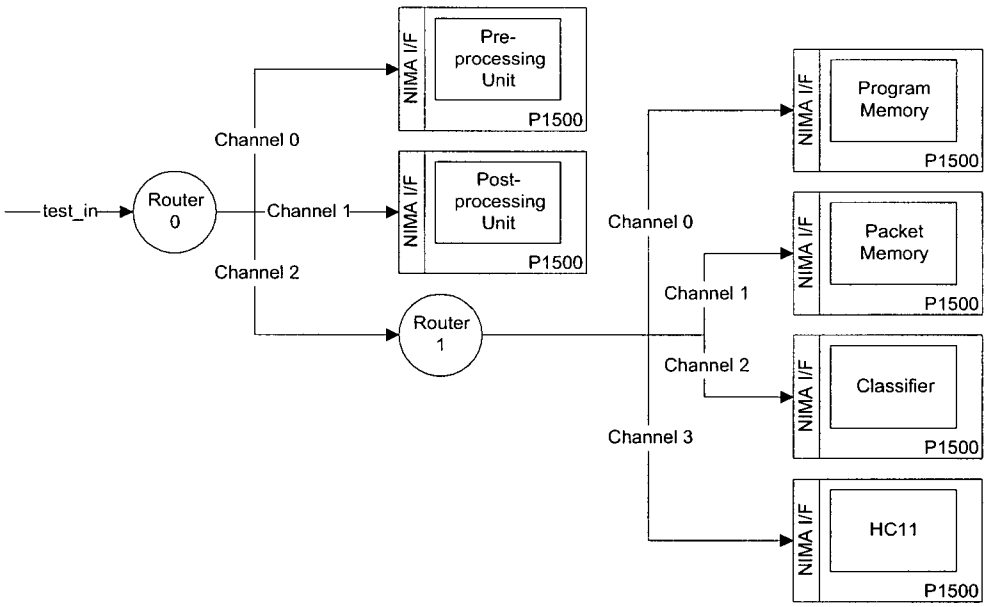

Figure 25. NIMA network configuration

To support a scalable NIMA network, a funnel device is added in the NIMA interface module (residing at the *Application Layer*) to buffer the test packet. The funnel device is essentially a special FIFO that has different input and output data width. The input width of the funnel is the width of the NIMA network, and the output width is the number of scan chains in the core. Currently implementation of the core in NPU has only one scan-chain, thus the funnel output is always a single bit stream.

4.4 NIMA Scheduling

Scheduling is another application-layer function. The performance of NIMA depends strongly on the scheduler, which determines when to send a particular test packet from the source. The scheduler needs to ensure that proper initialization sequences for testing individual cores are applied and that test results are not overwritten before they are retrieved. The scheduler also must prevent conflicts of the network resource usage and at the same time minimize the test application time. During the actual test, simultaneously activating all core DFT may result in power dissipation that exceeds the chip junction or package heat tolerance, so the scheduler needs to control the activity of the DFTs based on a given power budget. Moreover, the test data from NIMA network may arrive at a faster rate than can be consumed by the core, so a buffering scheme is incorporated to ensure the P1500 wrapper gets the data only when it is ready to accept it. To prevent buffer overflow, constraints are applied to the time interval between consecutive packets destined to the same core. One more issue that is starting to become important is the noise of simultaneously switching circuits. In DSM technologies, the wires are narrow but tall, so there is stronger coupling between neighbouring wires on the same layer. In addition, IR drop could be a problem as large amount of transistors switch at the same time. These constraints also need to be built into the scheduler, although the power issue is perhaps most important.

Before we can construct the scheduler, we need to understand the operation of the NIMA TAM in more details. The time for a packet to pass through the SoC is consisted of two parts: the delivery time - which is the time required by the NIMA TAM to deliver the

packet to the designated core - and the process time - which is the time to decode the packet and process the payload. The delivery time through NIMA is made shorter (through increase in NIMA bandwidth) then the process time so that NIMA can serve another core while the core(s) that already receive the packet is busy processing the payload. After NIMA has transported a packet for a particular core, it cannot send another packet to the same core until the first packet has been completely processed. If all cores in the system are busy processing their packets, then NIMA will be idle until one of the cores in the system has completely processed its packet. It is also possible that a core may sit idle after it completes its current packet because NIMA is busy transporting the packet for another core. The NIMA idle time is to be minimized through architectural design. The goal of the scheduler is to minimize the total test time by avoiding idle cycles at the cores.

The NIMA scheduling problem is very similar to a class of well-studied problem in the domain of single-process task scheduling. The problem, known as Sequencing with Intervals, is a decision problem described as follows:

> *Given a finite set of tasks T, and for each task t from T, there is a release time r(t) ≥ 0, a deadline d(t) ≥ 1, and a length l(t) ≥ 1, does there exist a feasible schedule for T that satisfies the release time and deadline constraints for all the tasks in T?* [18]

It is easy to see that the Sequencing with Intervals problem becomes a decision problem of NIMA scheduling by replacing the tasks by packets, the release time r(t) by the process time of the previous packet to the same core, and the deadline d(t) by a constant value K for all tasks. If the decision problem can be shown to be NP-complete, then so is

the optimization problem of finding the schedule with the minimum K. The Sequencing with Intervals problem has been shown to be NP-complete, and the same proof presented in [18] can shows that the decision problem (and thus the optimization problem) of NIMA scheduling is also NP-complete. This provides enough incentive for us to look for a heuristic algorithm rather than an exact one.

The current scheduling algorithm, shown in Figure 26, is based on the observation that if a long packet is scheduled first, it may be possible for NIMA to deliver the shorter packets while the core with long packet is still processing the first packet, and thus minimize the idle time of the cores. As a result, the algorithm sends the packet for the first-ready core. The ready time for a core is calculated in the algorithm based on the time to deliver the packet and the time to perform the test. If more than one core is ready at the same time, the core with larger scan chain (resulting in larger payload) has higher priority. The algorithm also looks ahead in time so that a non-ready core with a longer scan chain may preempt a ready core with a shorter scan chain. This heuristic sometimes reduces the idle time of the core DFT and thus decreases overall test time. It is not know at this moment whether this schedule produces the minimum total test time. Using heuristic methods to derive the optimal scheduling algorithm is the subject of future research.

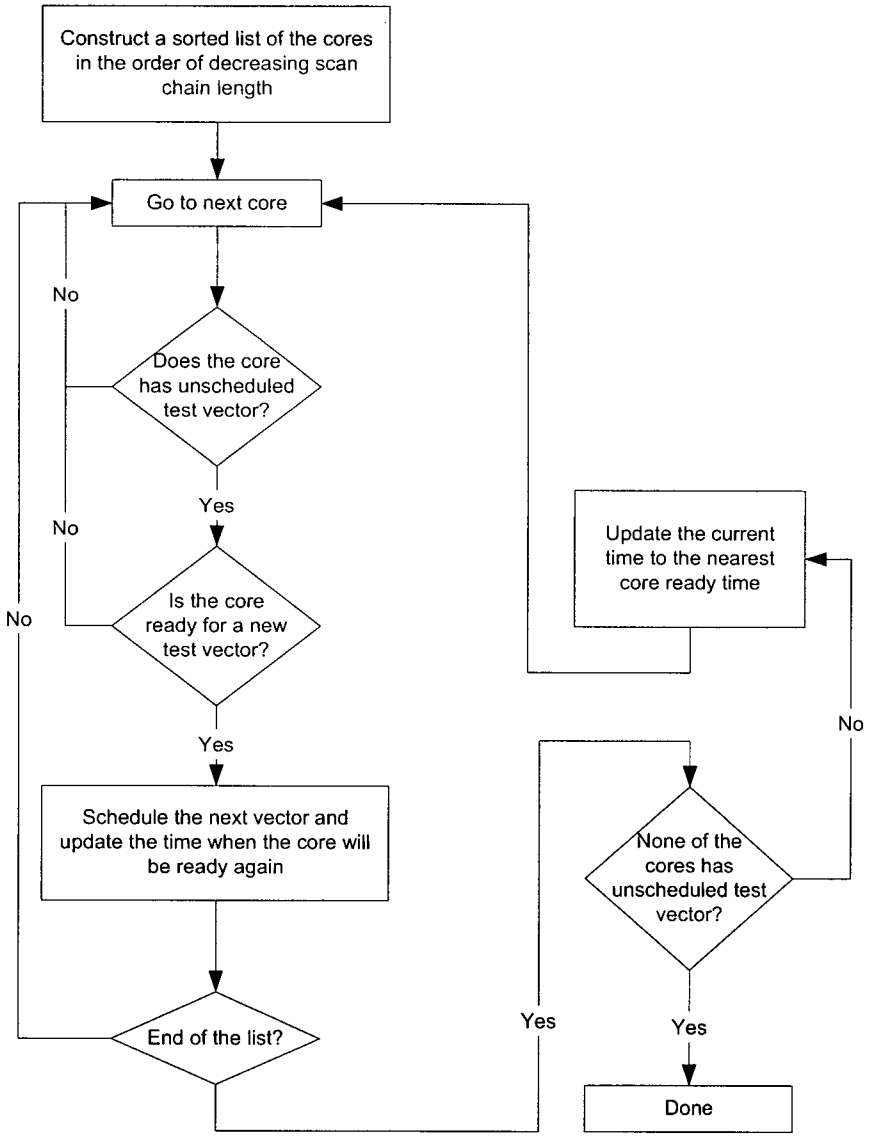

Figure 26. Flowchart of the NIMA scheduling algorithm

CHAPTER 5 TAM COMPARISON

In order to understand its advantages and disadvantages, the NIMA TAM was compared with other TAMs integrated on the same platform. Two TAMs, the serial TAM and the Test Rail, were implemented on the same NPU SoC platform on which the NIMA TAM was implemented. Figure 27 shows the conceptual diagram of the three TAM architectures. The serial TAM serves as the baseline approach since it has the least area overhead but the worst test application time; the improvement of the other two TAMs can be realized by comparing to the serial TAM. The Test Rail is one of the state-of-art TAM architectures that is often used as a comparison for new TAM designs. This chapter describes the implementation of the serial and Test Rail TAMs and presents the comparison results of the three TAMs.

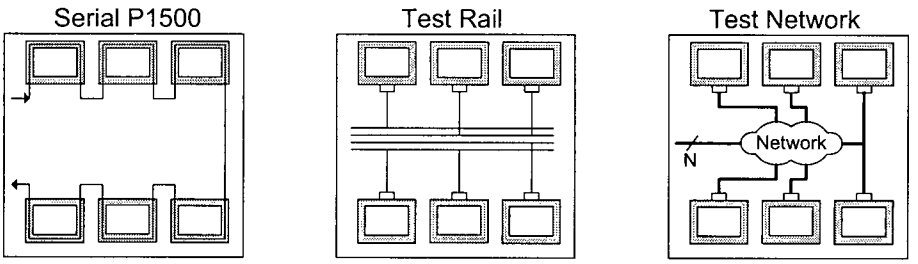

Figure 27. Conceptual diagrams of the three TAMs

5.1 Serial TAM Implementation

The serial TAM is created by simply threading the serial test output (provided by P1500 wrapper) of one core to the serial test input of another. Since the serial TAM was intended to be the TAM with bare minimum hardware, no TAP controller is added.

Instead, the control signals to the wrapper were directly controlled off-chip. The P1500 wrapper provides a bypass register to allow the packets to bypass unintended cores. This feature allows the BIST-ready cores to have their BIST modules initiated, and then the wrappers of these cores can be put to bypass so the cores that use full scan can be tested while the BISTs are running. In addition, for cores with multiple scan chains, the P1500 wrapper also allocates parallel test ports that can accommodate shifting of multiple scan chain simultaneously.

5.2 Test Rail Implementation

The other TAM used in the comparison is a variant of the Test Rail approach. The Test Rail TAM is often used as the reference design in SoC test research because it is a very generic architecture that can be highly optimized for area or test time. For our implementation, time-division multiplexing is used to control the access [16]. A core on the shared lines automatically assumes control over the bus when an internal counter reaches some predetermined value, each associated with a different time slot. This eliminates the need for an address bus. A TAP controller and an access controller are attached to each P1500-wrapped core. The access controller is composed of a 32-bit counter and some clock-gating logic. The 32-bit counter is implemented as a shift register. Before the test starts, a mask value is loaded into the shift register. During testing, the mask value is cycled through the shift register to produce the desired effect of a periodic counter. An arbitrary bit is chosen as the enable signal to the clock-gating logic. When the enable signal is asserted, the clock to the TAP controller and the P1500-wrapped core runs freely and the blocks function normally. When the enable signal is de-

asserted, the activities within the blocks are suspended. One exception is the BIST within the P1500 wrapper. It is allowed to run on its own regardless of the enable signal in order to minimize test time. For scheduling, a genetic algorithm is used to partition the test bandwidth among the cores and calculate the best bit-sharing configuration, if some bits in the bandwidth are to be shared [15].

5.3 Results

The SoC and the integrated TAMs were simulated at the gate-level. The details in the timing cycles were used to build a C/C++ model for each TAM architecture to allow simulation of different test scenarios. There were two types of DFT used in the network processor: full scan and BIST. The BIST approach has the advantage of lower test traffic than full scan. In addition, a BIST approach avoids off-chip vector storage and management, thus simplifying the ATE setup. However, the current BIST algorithm is based on pseudo-random pattern generation, which typically suffers from reduced fault coverage. To mitigate the problem, we require BIST to use more vectors (two to ten times more) than full-scan test. This of course results in longer test time. Consequently, it is important to explore the impact of various BIST and full-scan DFT combinations when considering the choice of TAM architectures. The TAM architectures were compared in six different scenarios. In the first scenario, all cores but two use BIST for testing. In each of the following scenarios, more cores that use full-scan DFT are added, which results in a larger volume of test data that need to be transferred onto the chip. Power estimation indicates that the power budget is not exceeded even when all DFT structures are

operating concurrently. As a result, the DFT structures are allowed to run in parallel whenever possible

5.3.1 Test Performance

Figure 28 shows the scan test time of serial TAM vs. NIMA. The vertical axis is the total test time (in clock cycles) to exercise core scan chains, and the horizontal axis indicates the total test data (in bits) transferred, which is calculated from the number of the test vectors for the NPU and the size of these test vectors. The longest BIST execution time is approximately 530,000 cycles for all six scenarios, and it is chosen as the threshold for total test time. The graph clearly suggests that the serial TAM is not practical. The total test time of the serial TAM exceeds the chosen threshold after the second scenario. The NIMA TAM is able to scale down the total test time by using larger bit width (the graph shows the scan test time of NIMA of width from 1 bit to 32 bits). NIMA with a width of 1 is performing worse than the serial TAM due to the extra overhead introduced by the packet header and the NIMA interface modules. NIMA with a width of 2 or above performs better than the serial TAM in all scenarios. As the number of full-scan cores increases, larger NIMA bit widths are required to keep the total test time bounded by the chosen threshold. In the last scenario where all logic cores in the SoC use full scan, an 8-bit NIMA is sufficient to keep the total test time below the chosen threshold of 530,000 cycles.

The graph also shows that the test time is a linear function of the data that needs to be transferred. So by moving toward the BIST-oriented DFT (moving toward the left hand

side of the x-axis), the test time is decreased linearly. Whether the saving in test time justifies the extra logic for BIST is a trade-off that must be analyzed by the chip designers. However, the technology scaling has made the area overhead less a concern as the designers can afford to implement more logic, whereas test time is a growing problem due to the increasing number of IP blocks that are integrated. According to [28] and [9], the trend of the industry is to adopt BIST and other test automation schemes for SoC testing. For a system that uses BIST as the primary DFT method, the design of the TAM architecture makes little difference in test time as evidenced by the graph. However, the reality is that a TAM is required more than just to transport scan vectors. A TAM is needed to send test vectors for diagnostic purposed and for functional verification of the embedded blocks. As a result, an efficient TAM design is essential to SoC testing.

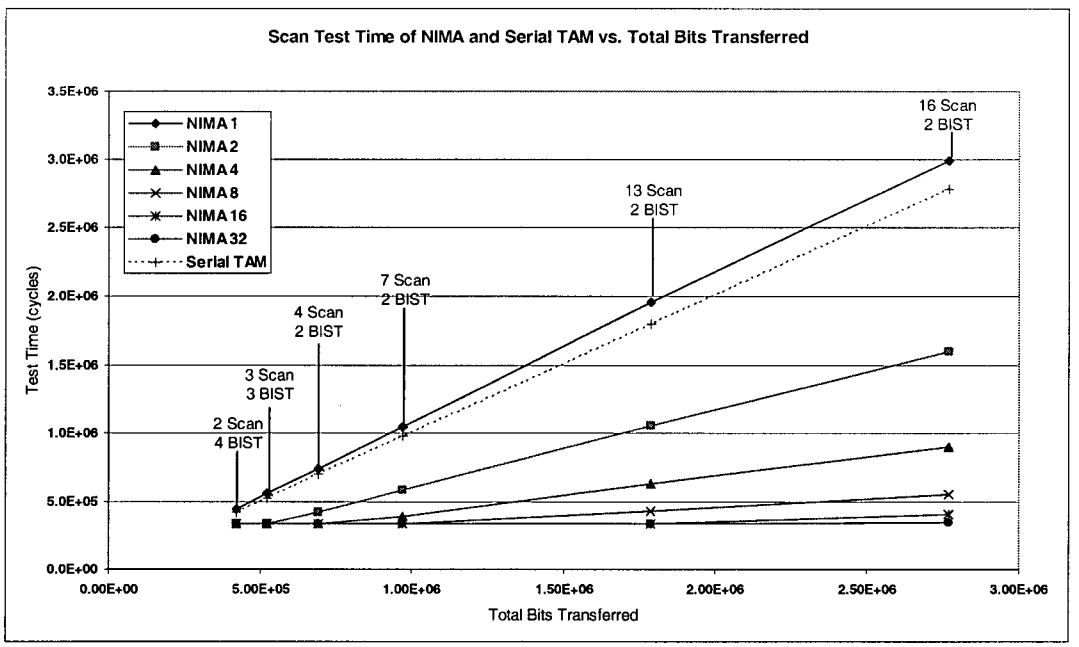

Figure 28. Test time of the serial TAM vs. NIMA

Figure 29 shows the Test Rail and NIMA test time of exercising full-scan DFTs using different test widths (from 1 bit to 5 bits). The results suggest that NIMA performs better or comparable to the Test Rail. The low performance of Test Rail with a small test width is caused by the limitation of the time-division multiplexing scheme of the Test Rail TAM. When the width is small, there are fewer wires to be shared among the cores, so the multiplexing scheme becomes less efficient. When the Test Rail TAM cannot distribute the available resources according to the data requirement of the cores, delays increase as cores with small data loads occupy the resources that could otherwise be used by the cores with large loads. The graph also shows that as the bit width of the TAM increases, the improvement in test time is diminishing. This is true for both NIMA and the Test Rail. For instance, there is a 40% improvement in test time when the bit width is increased from 1 bit to 2 bits, but the improvement is less than 15% in the case of from 4 bits to 5 bits. This observation is best described by the relation between total scan test time, total test bits of the SoC, and the bit width of the TAM, which can be abstractly formulated as

$$\text{Total scan test time} = \frac{\text{Total test bits}}{\text{Bit width of the TAM}} \quad (1)$$

When the bit width of the TAM is 1, the total scan test time in cycles is the same as the number of test bits to the SoC. By increasing the bit width of the TAM above 1, we parallelize the test process of the SoC. Since the total scan test time is inversely proportional to the TAM bit width, larger bit widths yield less improvement.

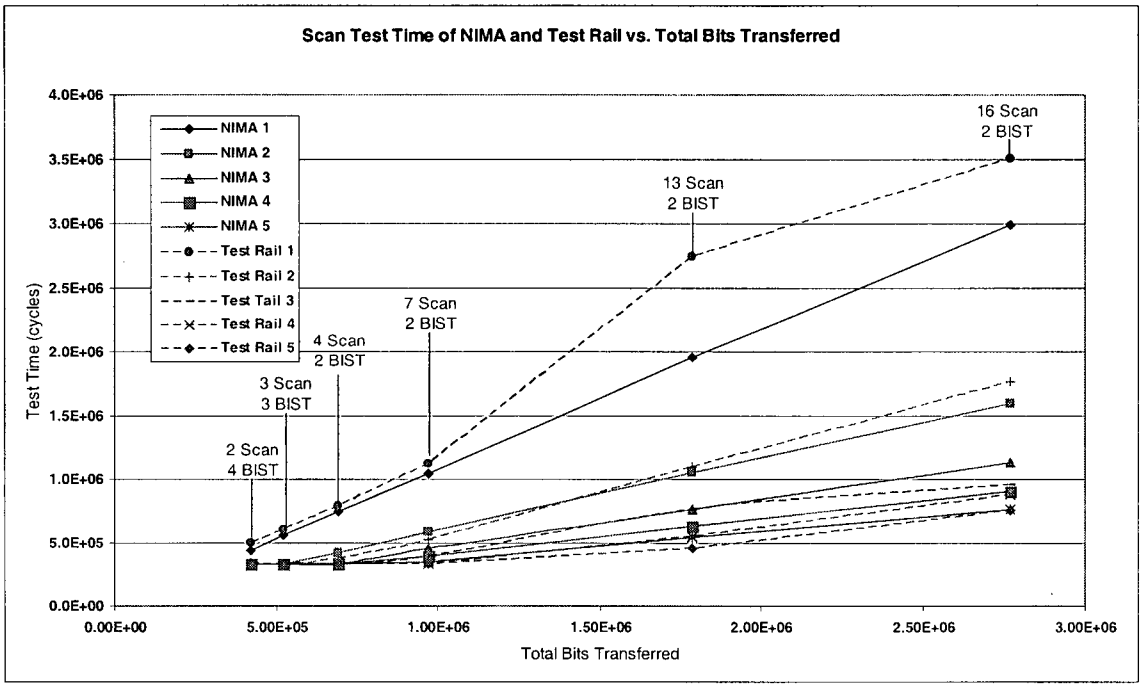

Figure 29. Test time of the Test Rail TAM vs. NIMA

Regardless of which TAM is used, the scan test time of the SoC has a lower bound, which is the longest test time of the cores. Using Equation 1, the optimal TAM width can be derived as the ratio of the total scan test time to the longest core test time, i.e.

$$W_{opt} = \text{round}\left[\frac{\sum_i t_i}{t_{max}}\right] \quad (2)$$

where t_i is the test time core i, and t_{max} is the longest test time of all cores. Then the theoretical best total test time is either the time to send in all the test data using the optimal TAM bit width, or the maximum BIST run time if the BIST run time dominates. Equation 3 gives the theoretical best test time in cycles.

$$T_{opt} = \max\left\{\frac{\sum_i t_i}{W_{opt}}, t_{BIST\_max}\right\} \quad (3)$$

Equation 3 assumes that all BIST-based cores can run their BIST simultaneously. The equation is only an approximation because it does not include the cycles for scan capture nor the delay through the TAM. The network TAM cannot achieve the theoretical best time because of the packet header overhead. The Test Rail cannot achieve the theoretical best time because of the granularity problem similar to the finite arithmetic error in digitized data. Specifically, the mask value of the Test Rail cannot perfectly match the ratio of the test data between the cores, so the Test Rail cannot achieve the optimal test bandwidth distribution.

5.3.2 Area Overhead

Aside from test time, there are many other factors that need to be evaluated for thorough comparison. Area overhead is one of the important design factors. Table 3 shows the gate area overhead of the three TAM architectures. The serial TAM requires no additional logic, and thus has zero gate area overhead. The Test Rail TAM requires a time-division multiplexer for each core, so the gate area overhead is derived from the multiplexer logic, and it has been shown to be approximately 1% of the total NPU gate area. The NIMA requires interface module at each core and routers to connect the network. The gate-area overhead has been calculated to be around 70% of the NPU gate area. For a real SoC design, area overhead this large will certainly be unacceptable. A more detailed analysis shows that 80% of this area overhead is attributed to the buffer memory used in the

interface modules. The overhead could be reduced by optimizing the buffer memory design, such as by using smaller memory cells or minimize the buffer size requirement.

Table 3. Gate area overhead comparison

TAM	Gate Area (um^2)	Percentage of NPU
Serial TAM	0	0%
Test Rail TAM	18882	1%
NIMA	1343192	70%

A more effective method for reducing area overhead of NIMA is reuse of system memory. If the NIMA TAM could use the functional memory modules already integrated in the SoC as the buffer memory, then the area overhead would decrease dramatically. As a result, the NIMA TAM will benefit from a distributed memory architecture where memory blocks are distributed across the chip rather then centralized in a large memory core. In fact, networking ICs often require large amount of memories [38][17][1]. For instance, [6] describes a networking SoC that contain 121 memory modules occupying 50% of the die area. In the case of the NPU SoC, there are also many memory elements in the proposed architectures (the FIFOs between the cores) as shown in Figure 10. Redesign of the interfaces to the FIFOs would allow them to be as buffers required by the NIMA interface modules.

5.3.3 TAM Design Automation

Synthesis automation is also an important consideration for TAM design, since building the TAM and the associated infrastructure manually is inefficient and error-prone. All three of the TAM designs compared can be built using standard cells and ASIC place-and-route tool. The P1500 wrappers consist of only standard logic elements such as D-

flip-flop, muxes, and AND gates, and can be automated generated by a script. The scan chains inside the cores require careful design to avoid timing violation during shift mode and capture mode, especially if there are scan chains that cross different clock domains [55]. Fortunately, scan-based methodology is widely supported by the industry and the CAD tools have the capability to construct race-free scan chains.

5.3.4 Summary

Table 4 summarizes the factors that SoC designers should consider when choosing a TAM architecture. Although a Serial TAM results in the longest test time, it is still included in the comparison because it offers the minimal area and routing overhead, which may be more important than other factors in certain applications. Judging from the merits that are presented, Test Rail is a better TAM if area overhead is important, while the NIMA is preferred if reducing I/O pin count is paramount. NIMA is able to save pin count because the control signals are embedded in the packets; once a packet arrived at the core, the interface module decodes the packet and generates appropriate control signals. The argument one could make is that the designer can serialize the control signals of the Test Rail by adding a shift register and hence reduce pin count [3]. The problem is that the test time becomes longer as the extra cycles are needed to setup the control signals and caution need to be taken to ensure that the cores' tests are not invalided while the control signals is being updated.

Table 4: TAM Comparison

	Serial		Test Rail		NIMA	
Test Time	-	Long test time	+	Can be scaled down	+	Can be scaled down
Logic Area	+	No overhead	+	Less than 2% of the SoC area	-	Overhead due to routers, interface logic, and funnels that act as buffers
Long Wire	-	Control lines scale with the number of cores	-	Control lines scale with the number of cores	+	No control lines
Total Test Pin	+	Constant number of dedicated test pins	-	3x width of Test Rail	+	2x width of NIMA network
Synthesis Automation	+	Can be automated	+	Can be automated	+	Can be automated
Pre- and Post-processing of test data	+	ATPG, pattern comparison	-	ATPG, distributing test data according to optimized test width and core assignment, and re-organizing test outputs for analysis	-	ATPG, test packet generation, packet scheduling, and extract test results from returned packets

+ = advantage
- = disadvantage

The control signals of the Test Rail are a hidden cost that is not emphasized in previous research papers. The number of I/O pins available for testing and the extra routing resource required by TAM control signals are important considerations. The proposed NIMA approach reduces the pin count and wires for control signals at the expense of more silicon area for additional logic. Many TAM designs also propose the use of the IEEE 1149.1 TAP controller as the central on-chip test control [9][64][54]. This can be done by adding custom TAP instructions and decoding logic to generate test control signals specific to the TAMs. In the case of NIMA, a centralized test control is not required since the schedule is re-computed and the test control signals are completely embedded inside the test packets. By incorporating the technique of embedding control signals into the test-rail architecture, a scalable TAM with low area overhead and low pin count could be devised. However, NIMA has other advantages that are not found in the other TAM architectures.

One additional advantage of NIMA is that it allows the test bandwidth to be re-allocated dynamically. For instance, during diagnostic test, engineers can use the full bandwidth of the network to test a single core if so desired. For the Test Rail, the bandwidth allocated to a particular core is fixed during implementation of the TAM. Furthermore, it appears that to achieve any significant improvement in performance, the TAM should be decoupled from the core DFT. Our current architectures of the NIMA TAM and Test Rail both use the same clock frequency as the core DFT. If the number of I/O pins available for testing is less than optimal, clocking the TAM at higher speed makes up for the lack of bit width and thus reduces test time. The Test Rail is an extension to the scan chain concept and is difficult to decouple from the core DFT. NIMA on the other hand, is conceived as a system completely independent of the core DFT, and thus allows such decoupling to be implemented naturally. A similar de-coupling scheme of separating communication from computation has already been proposed for the functional data transfer between the cores, although it is not yet clear that it is the solution for developing high-speed SoCs [49].

CHAPTER 6 CONCLUSIONS

The first part of this thesis work is a study of the reuse methodology based on the RMM. A reusable core (HC11) was developed by redesigning a legacy RTL block. The techniques described in RMM were applied to the core, but it was found that the time and effort spent in retrofitting the legacy block with reuse techniques were not justified. A better approach to convert a legacy block is to start the development from scratch and apply the reuse techniques coherently in the design flow. Another lesson learned from the exercise is that although the recommendations made by RMM are valid in general, there are situations where they should not be applied directly. It is important for the designers to fully understand the reasons behind these recommendations and analyze the effects before adopting them to the designs.

In addition, we find that the cost of reuse is high. Based on our experiment, the development time for a legacy core is estimated to be 3 times longer than a non-reusable one (the RMM estimate for a reusable design starting from scratch is 2 to 3x). In our case, learning the design and re-coding takes 1x, testbench development accounts for 1x, and documentation and prototyping accounts for another 1x. The testbenches are perhaps the most valuable part of the project because of the effort put into the development and its reuse potential. As suggested by RMM, the design should be reused 10x or more in order to recover the investment on reuse. For reuse methodology to be efficient, more support from the CAD tools is required. In addition to the testbench automation tools suggested by RMM, behavioral synthesis promises a faster and error-proof way of generating

reusable code, and formal verification and automated bus monitors should help simplify the verification task both at the core level and at the system level.

The second part of the thesis was the development of an SoC platform (NPU). The SoC was created using the HC11 and several other ASIC blocks. Since the design of a SoC is a major undertaking and could not be done within the scope of the thesis, the functionality of the SoC has been scaled back. Nonetheless, the project uncovered the major issues in SoC design, including bus interface design, inter-block timing, and SoC test, and provided solutions to these problems. The development of the SoC also reaches a point that the system allows various TAM architectures to be implemented and compared, which is the primary goal of this SoC.

The third part of the thesis involves integrating the NIMA TAM architecture into the NPU SoC and comparing NIMA with the serial TAM and the Test Rail TAM. We hoped to show that NIMA is a viable solution to the SoC test access problem. The results indicate that NIMA's performance is comparable to that of the Test Rail, but the area overhead is higher than the other architectures. The area overhead of NIMA can be reduced by optimizing the buffer memory or by making architectural changes to allow existing memory blocks to be reused. NIMA does have other benefits such as requiring fewer control signals and allowing the TAM to operate at a different frequency than the cores. These characteristics of NIMA make it an ideal candidate as a top-level TAM architecture for exchanging test data across an SoC, supported by other low-area-overhead TAM architectures that operate within a localized area of the chip.

6.1 Future Work

The future work for the HC11 core involves testing the fabricated chip. The future work for the NPU SoC includes performing system-level simulation using the actual cores and more test cases. Also an embedded software for the HC11 also needs to be developed in order to operate the NPU system. The future work for the NIMA TAM includes testing the fabricated NPU-NIMA chip and optimization of the NIMA design. The timing overhead of NIMA can be reduced by shortening the packet header, and the size of the router (considered as area overhead) can be minimized by more efficient coding. To reduce the area overhead due to the buffer in the NIMA interface modules, the logic surrounding the FIFOs needs to be modified so the FIFOs can be used by NIMA as well as the NPU. We would also like to extend the router design to support duplex packet transfers and explore asynchronous packet transfers using techniques such as self-timed clock encoding and handshaking [20]. Finally, the NIMA scheduler algorithm requires further investigation and a power constraint needs to be incorporated into the scheduler.

6.2 Contributions

This thesis work evaluates the reuse methodology using RMM as the guideline. The work provides insights into the recommendations made by RMM and argues their applicability using real design examples. A reusable core (HC11) was developed during the thesis work to demonstrate the processes converting a legacy IP to a reusable one. Using the HC11 and other ASIC blocks, a network processor (NPU) SoC was also developed. The NPU is intended to be a research platform and the cores are available as soft IPs. The NIMA TAM concept was implemented on this SoC platform and the process helped

refine the NIMA architecture. This thesis also discusses the trade-offs of the three TAM architectures and suggests some improvements for NIMA to reduce its area and delay overhead.

REFERENCES

[1] "7-Layer Packet Processing: A Performance Analysis." White paper, EZchip Technologies, 2000 <http://www.ezchip.com>.

[2] "AMBA Specification Rev 2.0." ARM Ltd., 1999 <http://www.arm.com>.

[3] Aerts, Joep and Erik Jan Marinissen. "Scan Chain Design for Test Time Reduction in Core-Based ICs." Proc. of IEEE International Test Conference, 1998, pp. 448-457.

[4] Benini, L. and G. De Micheli, "Networks on Chips: A New SoC Paradigm", IEEE Computer, January 2002, pp. 70-77.

[5] Bergamaschi, Reinaldo A. "Bridging the Domains of High-Level and Logic Synthesis." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 21, No. 5, May 2002, pp. 582-596.

[6] Bommireddy, A. et. al. "Test and Debug of Networking SoCs - A Case Study." Proc. of 18th IEEE VLSI Test Symposium, 2000, pp. 121-126.

[7] Camposano, Raul. "Behavioral Synthesis." Tutorial, Design Automation Conference, June 1996.

[8] Chakrabarty, Krishnendu. "Optimal Test Access Architecture for System-on-a-Chip." ACM Transactions on Design Automation of Electronic Systems, Vol. 6, No. 1, January 2001, pp. 26–49.

[9] Chandramouli, R. and Stephen Pateras. "Testing Systems on a Chip." IEEE Spectrum, November 1996, pp. 42-47.

[10] Crouch, Alfred L. Design-for-Test for Digital ICs and Embedded Core Systems. NJ:Prentice Hall, 1999.

[11] Cummings, Cliff. "Nonblocking Assignment in Verilog Synthesis." SNUG article, <http://www.deepchip.com>.

[12] Dally W. J. and B. Towles, "Route Packets, Not Wires: On-Chip Interconnect Network", Design Automation Conference, June 18-22, 2001, pp. 684-689.

[13] "Design Compiler User's Guide: Chapter 8 - Optimizing the Design." Synopsys Inc., 2000, pp. 1-48.

[14] "Designing with Reuse in Mind." Seminar presentation slides, Qualis Design Corporation, 2000 <http://www.qualis.com>.

[15] Edabi, Z. S. and A. Ivanov. "Design of an Optimal Test Access Architecture Using a Genetic Algorithm." Proc. of the Tenth Asian Test Symposium, 2001, pp. 205-210

[16] Edabi, Zahra Sabat and Julien Lamoureux. "TDM TAM Results." Report for EECE579, Department of Electrical and Computer Engineering, University of British Columbia, December 2001.

[17] Editors, "Next Generation Network Processor Technologies - Enabling Cost Effective Solutions for 2.5 Gbps to 40 Gbps Network Services." White paper, Intel, October 2001 <http://www.intel.com/design/network/papers>.

[18] Garey, Michael R. and David S. Johnson. Computer and Intractability - a Guide to the Theory of NP-Completeness. New York: Freeman and Company, 1979

[19] Ghosh, I., N.K. Jha, and S. Dey. "A Low Overhead Design for Testability & Test Generation Technique for Core-Based Systems." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 11, November 1999, pp. 1661-1676.

[20] Greenstreet, Mark. "Opportunities for Asynchronous Design." Presentation at the University of British Columbia, August 6, 2002.

[21] Gupta, Pankaj et al. "Packet Classification on Multiple Fields." Proceeding of ACM SIGCOMM, 1999, pp. 150-160.

[22] Guerrier, Pierre and Alan Greiner. "A Genetic Architecture for On-Chip Packet Switched Interconnects." Proc. of Design, Automation and Test in Europe Conference and Exhibition 2000, 2000, pp. 250 -256.

[23] Huang, C.H. "TSMC 0.18um Mixed Signal 1P6M Salicide 1.8V/3.3V SPICE Models." TSMC document, December 2000.

[24] Husak, David. "Network Processors: A Definition and Comparison." White Paper, C-PORT, 2000 <http://www.cportcorp.com>.

[25] Ichiriu, Mike. "High Performance Level 3 Forwarding." White paper, NetLogic Microsystems, 2000 <http://www.netlogicmicro.com>.

[26] Illman, Richard and Greg Aldrich. "On-time Finish Rests With Multiple Clocks." ISD magazine online archive, May 8, 2002 <http://www.isdmag.com>.

[27] Immaneni, Venkata and Srinivas Raman. "Direct Access Test Scheme – Design of Block and Core Cells for Embedded ASIC." Proc. of IEEE International Test Conference, September 1990, pp. 488-492.

[28] International Technology Roadmap for Semiconductors, Design Chapter, 1999 <http://public.itrs.net>.

[29] Ishkintana, Laura. "System-on-a-chip Reusability Study." Report for EECE496, Department of Electrical and Computer Engineering, University of British Columbia, December 2001.

[30] Janac, George et. al. "IP Supply Chain." ISD magazine online archive, March 1, 2001 <http://www.isdmag.com>.

[31] Jenkins, Charlie. "Speed and Throughput of Programmable State Machines for Classification of OC192 Data." White Paper, Solidum Systems, 2000.

[32] Karnane, Kishore and Leonard Drucker. "How Do You Know If Your Design Has Been Fully Verified?" Online presentation by Cadence, February 7, 2002 <http://www.netseminar.com>.

[33] Keatling, Michael and Pierre Bricaud. Reuse Methodology Manual. Second edition, Boston: Kluwer Academic Publishers, 1999.

[34] Levin, Peter L., and Reimhold Ludwig. "Crossroads for Mixed-Signal Chips." IEEE Spectrum, March 2002, pp.38-43.

[35] "Longest Prefix Match using the LN17010 Search Engine." Application Note 003, Lara Networks Inc., 1999 <http://www.laranetworks.com>.

[36] Marinissen, E. J. and Yervant Zorian. "Challenges in Test Core-Based System ICs." IEEE Communication Magazine, Vol. 37, Issue 6, June 1999, pp. 104-109.

[37] Marinissen, E. J. et al. "A structured & Scalable Mechanism for Test Access to Embedded Reusable Cores." Proc. of IEEE International Test Conference, 1998, pp. 284-293.

[38] Min, John. "Accelerating Network Applications with a User-Configurable Processor." Online presentation by ARC Cores, <http://www.techonline.com>.

[39] Mohor, Igor. "Boundary Scan Implementation." <http://www.opencores.org>.

[40] Mosenoson, Guy. "Practical Approaches to SoC Verification." White paper, Verisity <http://www.verisity.com>.

[41] "M68HC11 Reference Manual." Rev 5, Motorola Inc., February 2002 <http://www.motorola.com/semiconductors>.

[42] Nahvi, Mohsen and Andre Ivanov. "A Packet-Switching Communication-Based Test Access Mechanism for System Chips." IEEE European Test Workshop 2001.

[43] Partridge, Craig et al. "A 50-Gb/s IP Router." IEEE/ACM Transactions on Networking, Vol. 6. No. 3 (June 1998): 237-248.

[44] Petropoulus, Leo and Jeff McVay "Verification and Debug of Xtensa Configurable Processors." Online presentation by Mentor and Tensilica, February 2002, <http://www.techonline.com>.

[45] Richhetti, Mike. "Overview of Proposed IEEE P1500 Scalable Architecture for Testing Embedded Cores." Slides of the presentation at Design Automation Conference, June 20, 2001, pp. 1-26.

[46] Rothfus, Eric J. "The Case for a Classification Language." White paper, Agere Inc. <http://www.agere.com>.

[47] Saleh, Res. Notes, ms. University of British Columbia, August 2002.

[48] Santarini, Michael. "Standards group VSIA focuses on adoption challenges." EEDesign online archive, August 16, 2000 <http://www.eedesign.com>.

[49] Sgroi, M. et al., "Addressing the System-on-a-Chip interconnect Woes Through Communication-Based Design", Design Automation Conference, 2001, pp. 667-672.

[50] Sheafor, Stephen J. "Network Processors: Ushering in a New Era of Performance and Flexibility." White Paper, Sitera <http://www.sitera.com>.

[51] Shung, Bernard. "Network Processing ICs." ISSCC 2001 Tutorial, February 2001.

[52] "Static Crosstalk Analysis." White paper, Synopsys Inc., 2001 <http://www.synopsys.com>.

[53] Tumpach, Chris. "A Study of Reuse Methodology Through the Modification of a Microprocessor Core." Report for EECE496, Department of Electrical and Computer Engineering, University of British Columbia, December 2000.

[54] Varma, P. and S. Bhatia, " A Structured Test Re-Use Methodology for Core-Based System Chips." Proc. of IEEE International Test Conference, 1998, pp. 294-302.

[55] Wagner, Kenneth D. "Robust Scan-Based Logic Test in VDSM Technologies." IEEE Computer, November 1999, pp. 66-74.

[56] Website, <http://www.design-reuse.com>.

[57] Website, <http://www.gmvhdl.com>.

[58] Website, <http://grouper.ieee.org/groups/1500>.

[59] Website, <http://www.intel.com/research/silicon/mooreslaw.htm>.

[60] Website, <http://www.superlog.org>.

[61] Website, <http://www.systemc.org>.

[62] Website, <http://www.vsia.org>.

[63] Whetsel, Lee. "Addressable Test Ports, an Approach to Testing Embedded Cores." Proc. of IEEE International Test Conference, 1999, pp. 1055-1064.

[64] Whetsel, Lee. "An IEEE 1149.1 Based Test Access Architecture for ICs with Embedded Cores." Proc. of IEEE International Test Conference, 1997, pp. 69-78.

[65] Zorian, Yervant. "System Chip Test Strategy." Design Automation Conference, 1998, pp.752-757.