To the Graduate Council:

I am submitting herewith a thesis written by Tushti Marwah entitled “System-on-Chip Design and Test with Embedded Debug Capabilities.” I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science, with a major in Electrical Engineering.

__Donald W. Bouldin_____
Major Professor

We have read this thesis
and recommend its acceptance:

___Syed Islam

__Mohammed Ferdjallah

Accepted for the Council:
_____Anne Mayhew_____

Vice Chancellor and
Dean of Graduate Studies

(Original signatures are on file with official student records.)

System-on-Chip Design and Test with Embedded Debug Capabilities

A Thesis
Presented for the
Master of Science Degree
The University of Tennessee, Knoxville

Tushti Marwah
August 2006

Acknowledgement

First of all, I would like to thank my thesis advisor, Dr. Don Bouldin for his guidance, advice and support throughout my research. It was he who suggested me to pursue my research on the concept of using reconfigurable logic for debugging an SoC and I have greatly enjoyed my work. He was a constant source of encouragement for me throughout my research. I am also very grateful to Dr.Syed K.Islam and Dr.M. Ferdjallah for serving on my thesis committee.

I would like to thank Wei Jiang, for his teamwork in this project and instrumentation of the LEON design with the pre-silicon tools that gave me a platform to start from.

I would also like to thank DAFCA Inc., where I interned from June 2005-August 2005 for providing me with the pre and post-silicon tools and teaching me their usage. I am thankful to Paul Bradley, Rich Lundblad, Tat Ng and Daniel Hoggar at DAFCA for guiding me during the timing analysis and layout generation of the design.

Last but not the least I would like to thank my parents, Dr. Narinder Singh and Mrs. Kamla Marwah and my brother Joban for their unrelenting support and encouragement throughout my education.

Abstract

In this project, I started with a System-on-Chip platform with embedded test structures. The baseline platform consisted of a Leon2 CPU, AMBA on-chip bus, and an Advanced Encryption Standard decryption module. The basic objective of this thesis was to use the embedded reconfigurable logic blocks for post-silicon debug and verification.

The System-on-Chip platform was designed at the register transistor level and implemented in a 180-nm IBM process. Test logic instrumentation was done with DAFCA (Design Automation for Flexible Chip Architecture) Inc. pre-silicon tools. The design was then synthesized using the Synopsys Design Compiler and placed and routed using Cadence SOC Encounter. Total transistor count is about 3 million, including 1400K transistors for the debug module serving as on chip logic analyzer. Core size of the design is about 4.8mm x 4.8mm and the system is working at 151MHz. Design verification was done with Cadence NCSim.

The controllability and observability of internal signals of the design is greatly increased with the help of pre-silicon tools which helps locate bugs and later fix them with the help of post-silicon tools. This helps prevent re-spins on several occasions thus saving millions of dollars. Post-silicon tools have been used to program assertions and triggers and inject numerous personalities into the reconfigurable fabric which has greatly increased the versatility of the circuit.

Table of Contents

Chapter 1 Overview 1

1.1 System-on-Chip (SoC) 1

1.2 Design Testing and Verification 1

1.3 Motivation 2

1.4 Thesis Goals 3

1.5 Thesis Outline 3

Chapter 2 Background 4

2.1 Challenges in System-on-Chip Design 4

2.2 Types of Testing and their Problems 4

2.3 The DAFCA Solution 5

2.4 Debug Methods Supported by DAFCA 6

Chapter 3 Volunteer SoC 9

3.1 Overview 9

3.2 LEON CPU 9

3.3 AMBA Buses 11

3.4 AES Module 13

3.5 Artisan RAM 13

Chapter 4 Volunteer SoC Debug using DAFCA Tools 16

4.1 Introduction to the DAFCA Tools 16

4.2 Pre - silicon Tools 16

4.3 Instrumentation of the Volunteer SoC Using Pre-Silicon Tools 19

4.4 Post-Silicon Tools 20

4.4.1 Personality Editor *22*

Chapter 5 Implementation 27

5.1 Customizing the LEON2 Processor 27

5.1.1 Configuration *27*

5.1.2 Artisan RAM *28*

5.2 The Integrated SoC 29

5.3 Insertion of rMatrix to the Design 30

5.4 Synthesis 31

5.5 Physical Place and Route 32

5.5.1 Floorplanning, Powerplanning and Placement 32
5.5.2 Clock Tree Synthesis 34
5.5.3 Routing 35
5.6 Simulation Result 35
5.7 Sending the Design to MOSIS 37
Chapter 6 Testing with Post-Silicon tools 45
6.1 Tests Performed 45
6.2 Discussion 55
6.2.1 Area Comparison 55
6.2.2 Timing Overhead 55
6.2.3 General Discussion 55
Chapter 7 Future Work and Conclusion 57
7.1 Future Work 57
7.2 Conclusion 57
References 58
Vita 61

List of Tables

Table 1.1 VLSI Technology Trends [5] **2**
Table 3.1 LEON2 Memory Address Space **14**
Table 4.1 Instrument Capability and Attributes **18**
Table 5.1 Block RAM Parameters **30**
Table 5.2 IP Block APB Memory Mapping **30**
Table 6.1 Transistor Count Comparison **56**

List of Figures

Figure 2.1 DAFCA Solution 6
Figure 2.2 rMatrix and rWrap 7
Figure 3.1 SoC Baseline Platform Block Diagram 10
Figure 3.2 LEON2 Block Diagram 10
Figure 3.3 AHB Master Interface 12
Figure 3.4 APB Slave Interface 13
Figure 3.5 Artisan RAM Read Cycle 14
Figure 3.6 LEON-2 Processor Read Cycle 15
Figure 4.1 Wrap versus Tap 17
Figure 4.2 Post-Silicon Tools Interface 20
Figure 4.3 Routing Signals 21
Figure 4.4 Personality Editor 23
Figure 4.5 Example Design Block with Associated Logic View 23
Figure 4.6 Example Logic View 24
Figure 4.7 LUT Function Entry Dialog 25
Figure 4.8 Example Routing View 25
Figure 4.9 Module Insertion Dialog 26
Figure 5.1 Instrumented SoC 31
Figure 5.2 Physical Design Flow [13] 33
Figure 5.3 Floorplan Guide 34
Figure 5.4 Clock Tree Phase Delay 35
Figure 5.5 SoC Design Routed 36
Figure 5.6 Post-Layout Simulation 38
Figure 5.7 SoC Layout in Cadence Virtuoso 40
Figure 5.8 Guard Ring and CHIPEDGE Polygon 41
Figure 5.9 Initial MT Density in the SoC 42
Figure 5.10 Initial ML Density in the SoC 42
Figure 5.11 SoC with MT Fill 43

Figure 5.12 SoC with ML Fill 44
Figure 6.1 Personality Editor Window Depicting ‘paddr’ Modification 46
Figure 6.2 SoC Post-Layout Simulation after ‘paddr’ Changes 47
Figure 6.3 Personality Editor Window Depicting ‘hgrant’ Modification 49
Figure 6.4 SoC Post-Layout Simulation after ‘hgrant’ Changes 50
Figure 6.5 Personality Editor Window Depicting ‘done’ Modification 51
Figure 6.6 SoC Post-Layout Simulation after ‘done’ Changes 52
Figure 6.7 Post-Layout Waveform after Injection of Counter Personality to paddr[2] and paddr[3] 53
Figure 6.8 ‘assert Always’ Assertion being Fired 54

Chapter 1 Overview

1.1 System-on-Chip (SoC)

System-on-a-Chip (SoC) design refers to implementing an entire electronics sub-system on a single IC. A typical SoC consists of:

1. one or more microcontroller, microprocessor or DSP core(s);
2. memory blocks including a selection of ROM, RAM, EEPROM and Flash;
3. timing sources including oscillators and phase-locked loops;
4. peripherals including counter-timers, real-time timers and power-on reset generators;
5. external interfaces including industry standards such as USB, FireWire, Ethernet, USART, SPI;
6. analog interfaces including ADCs and DACs;
7. voltage regulators and power management circuits.

These blocks are connected by an industry-standard bus such as the AMBA bus from ARM. DMA controllers route data directly between external interfaces and memory, by-passing the processor core and thereby increasing the data throughput of the SoC [1].Today designers of application-integrated circuits are faced with the challenge of creating and verifying the content of million-transistor chips as quickly as possible in order to reduce the time-to-market [2].It has been estimated that a one-month delay in bringing a product to market can result in a loss of ten percent of the potential revenue [3].Hence, not all of the transistors on these chips can be customized but instead must be ported from previous designs. These reusable cores or intellectual property (IP) blocks include CPUs (like ARM, PowerPC and LEON), MPEG decompression engines, PCI bus controllers, specialized DSPs, etc. Combining several complex cores using standard cells is much more manageable and quicker than designing millions of transistors one at a time. The myth that characterizes today's IP is that these components are blocks that have well-defined contents and interfaces. However, they are often fuzzy and hence appear more like patches in a quilt, which must be stitched together. The components cannot be assembled blindly and rapidly, but rather must be carefully pieced together to form a working system. Therefore, design for reuse does not come free. Rather it involves much more in-depth documentation and characterization.

1.2 Design Testing and Verification

Testing has two major aspects: control and observation. Control is the measure of the difficulty with which an internal net can be driven to a particular logic state. Observation is the ability with which an internal signal's current logic state can be driven to an output where it can be measured.

To test any system it is necessary to put the system into a known state, supply known input data (test data), and observe the system to see if it performs as designed and manufactured. If control or observation cannot be carried out, there is no way to know empirically if the system performs as it should. During the normal product development flow, testing (it may be known by different names) takes place at many points during the process.

Table 1.1 lists the trends that impact cost and difficulty of testing. In order to overcome the challenges posed by increasing complexity of designs (approaching one billion transistors) better testing algorithms and debug methods are needed to locate the design/physical errors and more efficient ways are required to verify the fixes. Currently, an increasing number of circuits are designed in accordance with Design-For-Testability (DFT) guidelines, which makes the design easier to be tested and debugged. The payoff is not only higher quality, but also shorter time-to-market [4].

1.3 Motivation

Silicon debug is a costly problem that delays volume production and market entry. Until recently, debugging pre-production silicon required one or two prototype re-spins, weeks of lab time to isolate problems, and cost hundreds of thousands of dollars. Today, four or five re-spins are not uncommon, and months can be spent stabilizing the design. Moreover, with the rapid shrinking of the feature size, more physical errors are bound to occur due to timing, crosstalk, noise, temperature, and process variation. At the same time, the designers are losing visibility into the design as the size of the design increases [6].

Scan chains are only accessible with very constrained test patterns, and not at speed. Simulators may take weeks to produce an incorrect result, or miss it if the problem is environmentally dependent or an intermittent manufacturing defect. The motivation of my research is to restore the designer's ability to debug silicon

Table 1.1 VLSI Technology Trends [5]

Year	97-01	03-06	09-12
Feature Size (um)	0.25-0.15	0.13-0.10	0.07-0.05
Millions of transistors/cm^2	4-10	18-39	84-180
Number of wiring layers	6-7	7-8	8-9
Die Size, mm^2	50-385	60-520	70-750
Pin Count	100-900	160-1475	260-2690
Clock Rate, MHz	200-730	530-1100	840-1830
Voltage, V	1.2-2.5	0.9-1.5	0.5-0.9
Power, W	1.2-61	2-96	2.8-109

rapidly. I have used tools from DAFCA Inc. to insert this at speed debug capability to our Volunteer SoC which is a LEON-based SoC platform built by previous students at the University of Tennessee. The baseline SoC was first instrumented with reconfigurable fabric (using DAFCA pre-silicon tools) that enables the user to run the design at speed and monitor selected signals using assertions and triggers. Then the post-silicon tools were used to isolate and repair bugs, as well as to accelerate fix verification.

1.4 Thesis Goals

1. To insert an rMATRIX (a two-dimensional reconfigurable logic block array used for wrapping ports/signals) into our SoC design.
2. To configure the LEON design according to the IBM 7RF 180-nm process.
3. To use DAFCA post-silicon tools to create debug assertions, signal generators and logic fixes or repairs.
4. To synthesize, place and route the design using the IBM 7RF process and submit to IBM via MOSIS for fabrication.
5. To test the fabricated chip and verify its self-repair capability using DAFCA post-silicon tools.

1.5 Thesis Outline

In this section I briefly introduced the challenges that the current design technology is facing and the need for new testing methodologies and tools. In Chapter Two I will discuss some background information such as the challenges in SoC design, problems associated with the presently prevalent testing methodologies and the solution offered by DAFCA to overcome these problems. Chapter Three explains the Volunteer SoC platform developed at the University of Tennessee and its components. Chapter Four discusses about the DAFCA tools and instrumentation of our SoC with these tools. Chapter Five presents the design implementation in detail and my results. Chapter Six presents the tests performed with the post-silicon tools. Chapter Seven concludes my work and presents a plan for future work.

Chapter 2 Background

2.1 Challenges in System-on-Chip Design

With rapid advances in semiconductor processing technologies, the density of gates on the die increased in line with what Moore's law predicted. This helped in the realization of more complicated designs on the same IC. Over the last few years, an increasingly evident need has been that of incorporating the traditional microprocessor, memories and peripherals -or in other words the whole system - on single silicon. This is what has marked the beginning of the SoC era.

The advantages of an SoC are drastic reduction in the overall design cycle time due to the use of existing IP (intellectual property) blocks ,superior performance levels, less area requirements and hence greater economic viability. On the flip side a greater concern of the designers today is the efficient testing of such a chip while meeting the time to market pressures. Bugs in the design can lead to several re-spins costing millions of dollars. Test costs in VDSM (very deep submicron) SoC designs is approaching 60% of the total product cost [7]. As the design complexity increases the designer loses visibility into the chip and testing becomes even more challenging. So what is needed to achieve at this point is:

1. Improvement in Test Coverage
2. Reduction in Test Time
3. Reduction in Tester Requirements
4. Self-repair

2.2 Types of Testing and their Problems

Today circuit density has increased dramatically, and the cost of devices has decreased as their performance has improved [8]. So reliability has become increasingly important. However with the advent of very-large-scale integration (VLSI), gate density is increasing much more rapidly than the number of access terminals. So the ability to generate test patterns and process fault simulation is deteriorating. This suggests that the circuits should be designed to be tested easily. Testability of a logic circuit can be defined as the ease of testing or as the ability to test easily or cost-effectively.

Early designers mainly used prototypes which were physical mockups of the circuit being designed for testing purposes. These prototypes were used to evaluate the logical correctness and timing characteristics of the design. The prototype is attractive because it can run at or near design speed, it can be evaluated near actual operating conditions, it does not require detailed simulation models, and it can be run with virtually unlimited amounts of stimuli [9]. But some of the drawbacks of a prototype are that many months of effort and expenditure

may be required to build it. Also, if the prototype goes down due to some reason the entire design team may be idled.

So eventually simulation started to play an important role in the testing process. Present day simulators can operate on models at levels of abstraction ranging from switch level to behavioral. Simulation at a high level of abstraction requires less detailed processing and hence is faster. But in the other cases the simulators may take weeks to reproduce an incorrect result, or miss it entirely if the problem is environmentally dependent, an intermittent manufacturing defect, or caused by inaccuracies in the pre-silicon models (e.g. an inadequately modeled hard IPcore [6]). So the designer is forced to guess the cause of the problem based on incomplete information and limited visibility. Consequently, the process of determining an appropriate repair strategy becomes incredibly difficult and risky.

Another approach to testing is generating test patterns and applying them to the circuit under test. These input patterns produce erroneous responses when faults are present. However as the complexity of circuits increases the generation of test patterns becomes a tedious process. Moreover scan chains are only accessible with very constrained test patterns, and not at speed. Also, storage of huge amount of test patterns and responses is required and testing is slow because of the shifting of patterns through the scan path.

2.3 The DAFCA Solution

One way to overcome the above problems is to add extra logic to be used for testing to the design. Formerly a designer used to concentrate on minimization of hardware cost and maximization of performance. However because of the increasing testing cost and decreasing hardware cost, design for testability is now becoming a cost effective approach.

DAFCA’s solution restores the designer’s ability to debug silicon rapidly [6]. It achieves this task by putting reconfigurable instrumentation onto the chip that enables the user to isolate and repair bugs. It provides at-speed access to internal signals on the chip, delivers instrumentation for trigger and capture events, and creates reconfigurable structures that can be used for in-situ repairs. Figure 2.1 depicts the pre-silicon instrumentation and post-silicon debug approach followed by DAFCA. The DAFCA solution has both software and in-silicon components. The software interface enables the user to specify where the instrumentation is placed on chip and provide debug and analysis capabilities (post-silicon).

DAFCA’s in-silicon instruments are known as ReDI fabric.

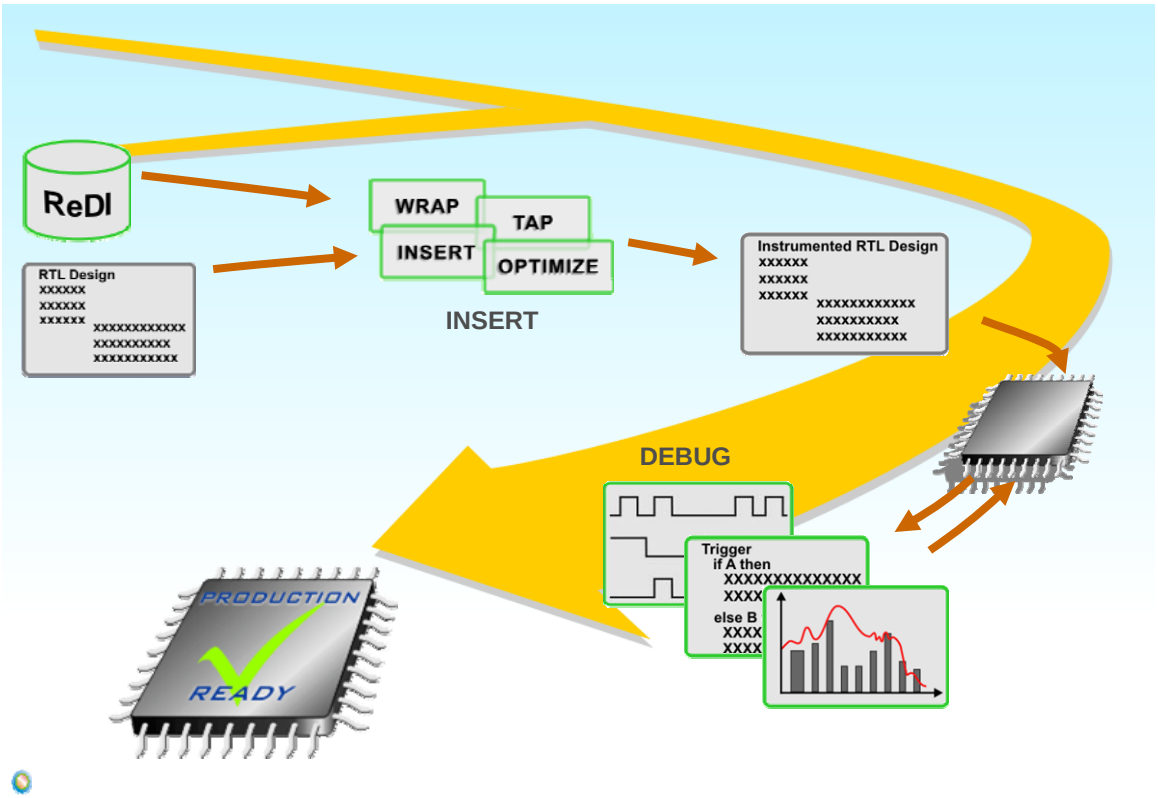


Figure 2.1 DAFCA Solution

Two of the ReDI logic elements, the rWrap and the rMatrix are shown in the Figure 2.2.The rWrap is a thin reconfigurable logic block that can be used to generate debug assertions, signal generators and even logic repairs. It wraps the desired user block and hence provides complete controllability and observability to the critical ports. An rMatrix is a two dimensional version of the wrapper and is used to wrap more complex user blocks.

With the DAFCA instrumentation in place the designer can perform at-speed in-system debug. The user can configure the instrumentation by identifying the signals to be monitored and programming the wrappers to realize assertions and logic fixes. The internal state of the system is hence available for examination.

A detailed discussion of the DAFCA tools is given in Chapter 4 of this thesis.

2.4 Debug Methods Supported by DAFCA

There are a number of industry-wide debug methods currently used for chip debug. Each of these methods is supported by DAFCA [10].

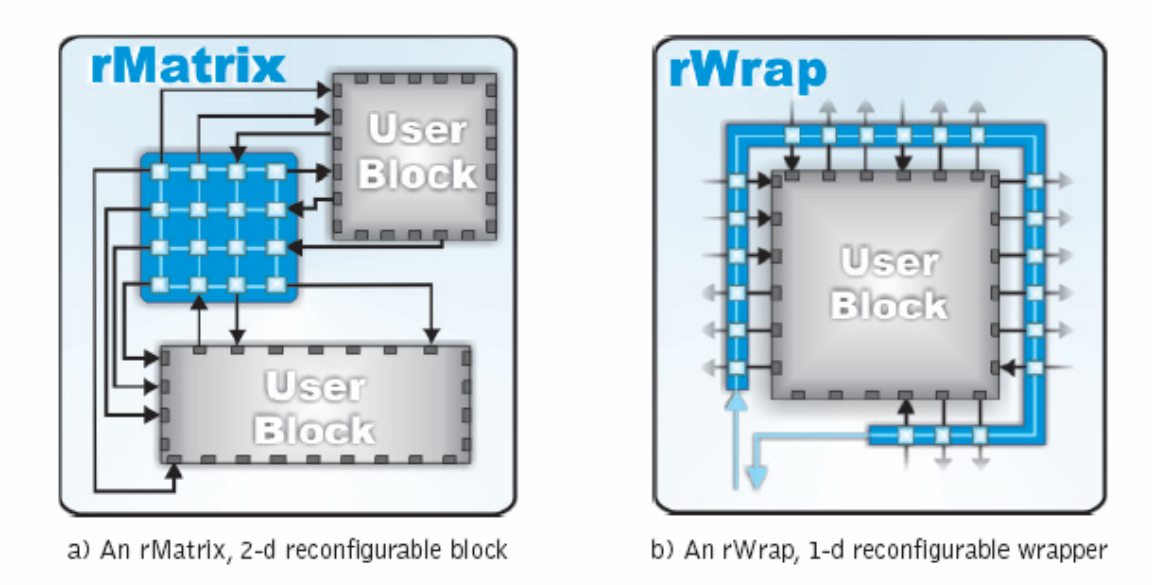


Figure 2.2 rMatrix and rWrap

1. **Full Scan/Partial Scan/Single-Step** - Full Scan or Partial Scan uses the scan registers to provide observation and control over the circuit under test. A circuit can be tested “at-speed”, but must be halted to retrieve data from the scan chain. Hence this debug method requires an iterative approach of starting, stopping, and restarting the circuit under test in order to isolate a problem. In many cases it is useful to advance a sequential circuit one clock at a time during debug. This is often the most effective method to isolate problems rapidly. Unfortunately many circuits can not be stopped and started, much less single-stepped.

2. **At-Speed Observation -** Whereas Full Scan provides a wide but shallow view of signal state, many chips are designed with debug muxes and memory that provide a means to see a narrow but deep view of selected signals. This is analogous to logic analyzer based debug techniques where a user builds triggers and captures data as a means to isolate a particular problem. While this method can be effective, it is often iterative as the user often must create many triggers and capture many signal states to reduce the scope of the problem continuously before it is fully isolated.

3. **Assertion-based Debug –** Assertion-based debug uses assertions to monitor the behavior of circuits. In some cases the assertion is hard-coded into the RTL (register transfer level) code, while in other cases the assertion can be defined post-silicon using the DAFCA post-silicon tools. While some assertions can pinpoint a problem, most only limit the scope

of a problem. Often additional methods (or assertions) are required to isolate a problem fully.

4. **At-Speed Control –** At-speed control refers to debug techniques that allow a user to dynamically modify the behavior of a circuit running at-speed. In some cases this is simply the control over a configuration register changed on-the-fly. In other cases, it may involve the dynamic reconfiguration of a programmable circuit. At-speed control does not include the ability to change scan registers and/or configuration registers while in “test mode”. Such “test mode” control is provided with full scan. At-speed control explicitly implies the ability to modify the behavior of a circuit without requiring the circuit to be slowed or stopped.

Chapter 3 Volunteer SoC

The Volunteer SoC that I have worked on was developed as part of a graduate course in the Electrical and Computer Engineering Department of the University of Tennessee over the last two years. The IP cores library were developed and verified by different student teams. In this section I am going to discuss about the basic components of this SoC and also the AMBA bus.

3.1 Overview

Our SoC is a LEON 2-1.0.12 based SoC and has two IP blocks attached to it. The communication between the processor and the IP blocks takes place through AMBA (AHB and APB) buses. The AHB is used for high-speed data transfer and APB for on-chip peripherals. A memory controller is used to communicate with external memories. The first IP block is an encryption module that performs 128-bit decryption according to the Advanced Encryption Standard (AES). The second block is a dummy block that is inserted to demonstrate the debug capabilities of DAFCA tools.

The SoC is designed at the register transfer level and can be fully synthesized for different processes. I have implemented it using the 180-nm IBM 7RF process. The block diagram of this SoC is shown in Figure 3.1.

The two user IP blocks are attached to both the APB and the AHB. They are defined as masters on the AHB and slaves on the APB. In our design, the AHB is used for data transfer between memory and IP blocks and the APB is used for control signals. APB bridge is the only master on the APB.

3.2 LEON CPU

The LEON2 CPU is a 32-bit processor conforming to IEEE-1754 (SPARC V8) architecture. It is designed for embedded applications with the following features on-chip: separate instruction and data caches, hardware multiplier and divider, interrupt controller, debug support unit with trace buffer, two 24-bit timers, two UARTs, power-down function, watchdog, 16-bit I/O port and a flexible memory controller. New modules can easily be added using the on-chip AMBA AHB/APB buses. The VHDL model is fully synthesizable with most synthesis tools [11].

A block diagram of LEON-2 can be seen in Figure 3.2.

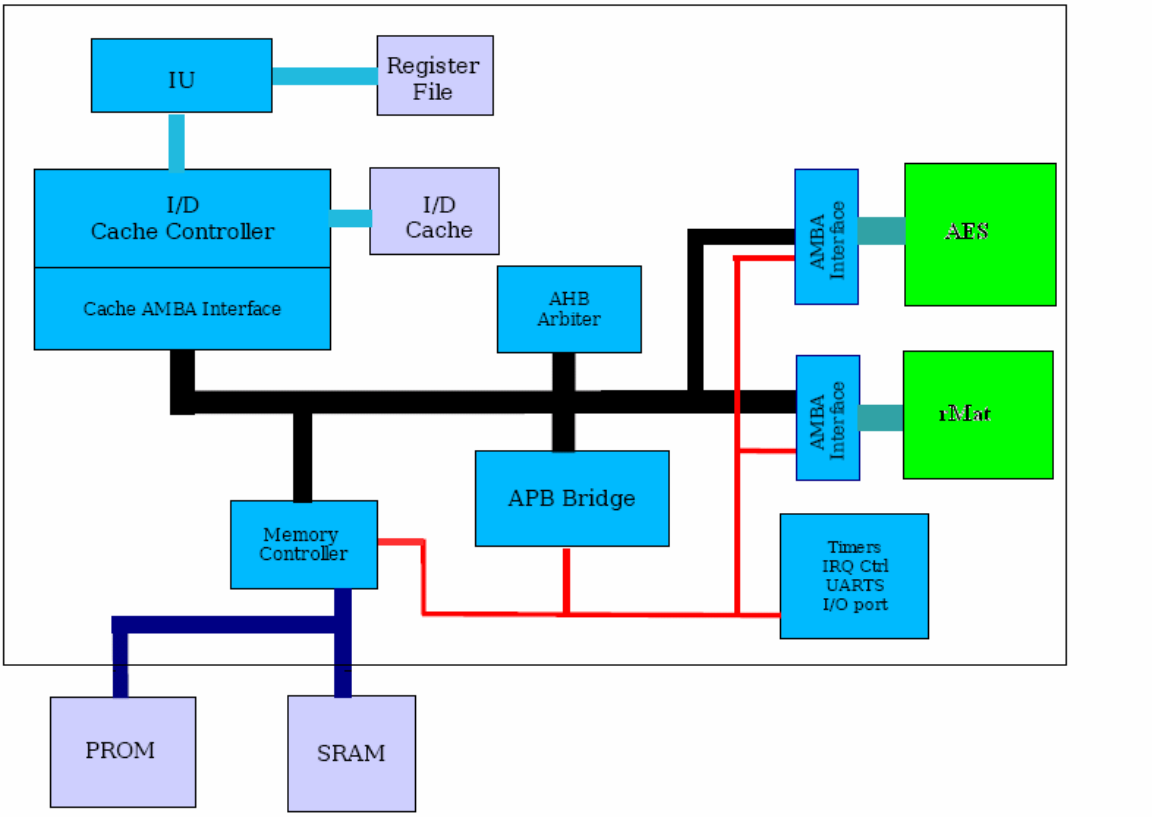


Figure 3.1 SoC Baseline Platform Block Diagram

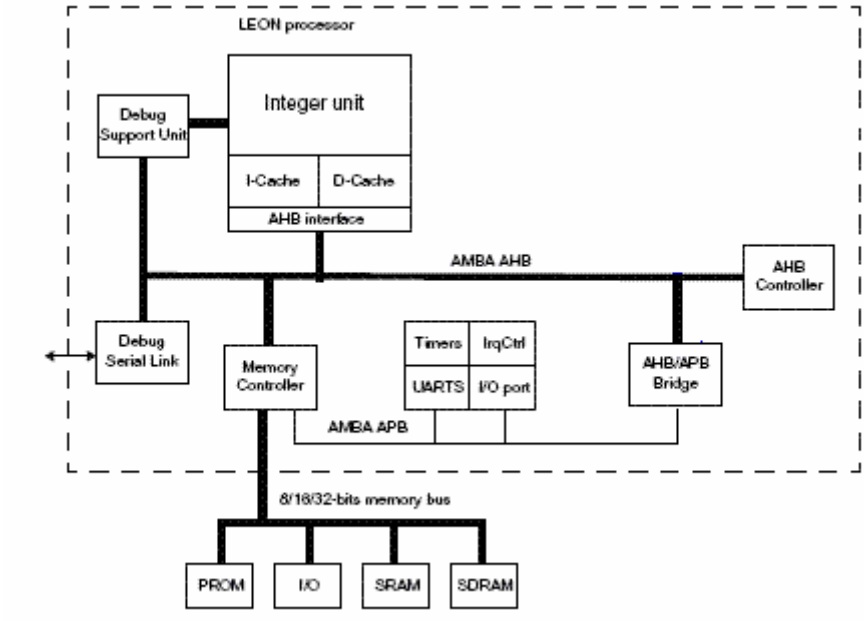


Figure 3.2 LEON2 Block Diagram

The Leon CPU is configured to minimum size in order to save silicon space. The registers in LEON2 are implemented using a dual port RAM (dpram136x32) while the instruction and data cache are implemented using single port RAMs (ram256x32 and ram32x30 respectively). I have used the IBM 7RF Artisan RAM generator to generate these RAM models for this design. Most of the optional components like the hardware multiplier/divider and the on-chip debug unit are disabled in the baseline design.

3.3 AMBA Buses

AMBA, which stands for an Advanced Microcontroller Bus Architecture, is an open standard which defines an on-chip bus specification for interconnection and management of various functional blocks that are a part of a System-on-Chip.

Two distinct buses defined within the AMBA specification are [12]:

Advanced High-performance Bus (AHB) - The AMBA AHB is for high-performance. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macrocell functions.

Advanced Peripheral Bus (APB) - The AMBA APB is for low-power peripherals. AMBA APB is optimized for minimal power consumption and reduced interface complexity to support peripheral functions. APB can be used in conjunction with the system bus.

In our design AHB and APB are used as the on-chip bus architecture for our platform. The APB is used to access on-chip registers in the peripheral functions while the AHB is used for high-speed data transfer.

The user IP blocks can be attached to AHB and/or APB, as a master or as a slave. In this design the IP blocks have been attached to both the AHB and APB in order to separate control signals and data transfer. They are defined as additional masters on the AHB and slaves on the APB. As a master on the bus, the IP blocks have the ability to initialize a data transfer with bus slaves without waiting for the CPU, which is essential for a high performance system [13].

The AHB master interface is shown in Figure 3.3. A description of some of the signals is given below:

1. HBUSREQx – A signal from the bus master x to the bus arbiter which indicates that the bus master requires the bus.
2. HWRITE – When high this signal indicates a write transfer and when low a read transfer.

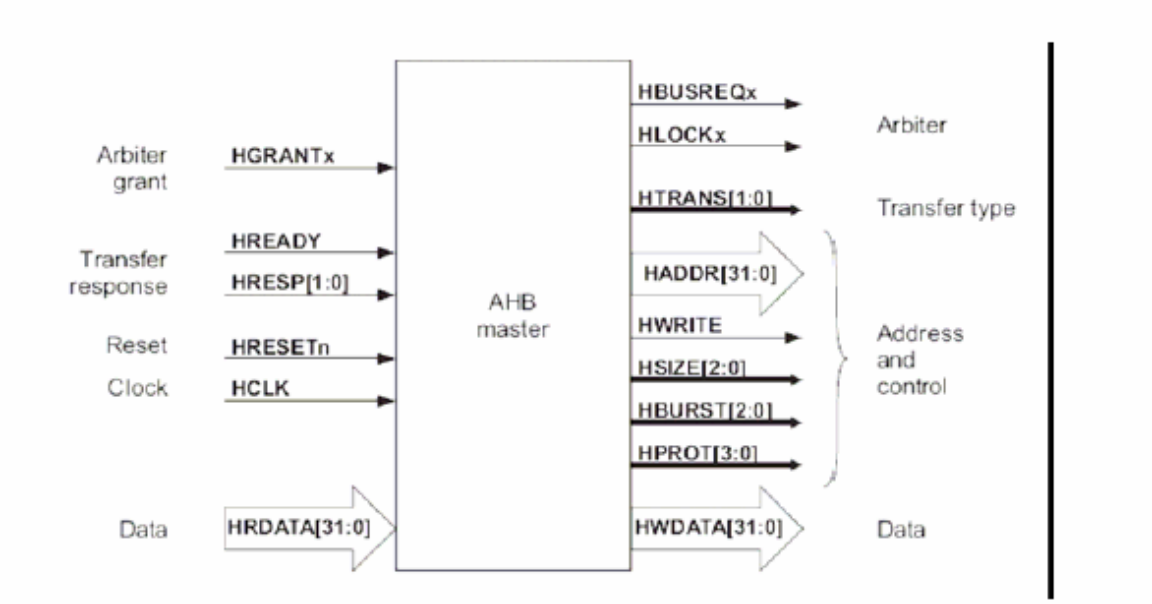


Figure 3.3 AHB Master Interface

3. HRDATA – The read data bus is used to transfer data from the bus slaves to the bus master during read operations.
4. HWDATA – The write data bus is used to transfer data from the master to the bus slaves during the write operations.

Before the data transfer, the master first requests the bus from the arbiter. Once the bus is granted, the master can initialize a data transfer.

The user IP block also serves as an APB slave, whose interface is shown in Figure 3.4.

A description of some signals of the APB is given below:

1. PSELx – This signal indicates that a slave device is selected and a data transfer is required.
2. PENABLE – This signal is used to time all accesses on the peripheral bus.
3. PWDATA - The write data bus is driven by the peripheral bus bridge unit during write cycles.
4. PRDATA – The read data bus is driven by the selected slave during the read cycles.

Being a slave, the IP block will wait for the selection signal from the bus master and read in the data/address bus, with control signals indicating the type of the data transfer. According to the command, it will perform a particular function.

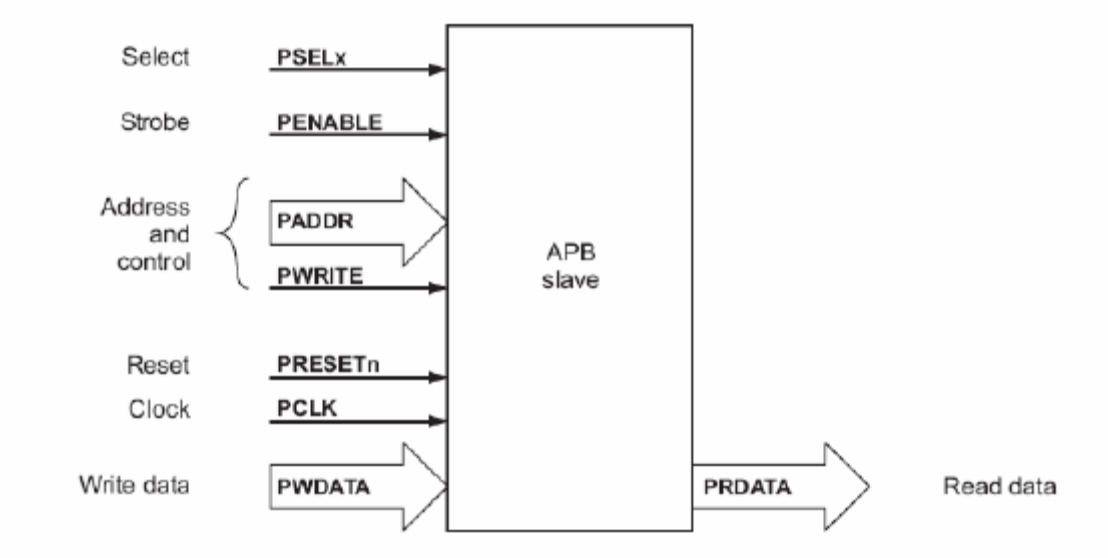


Figure 3.4 APB Slave Interface

3.4 AES Module

In this SoC design an Advanced Encryption Standard (AES) module is being used as a user IP block. AES is a block-cipher/decipher with block size of 128 bits. Keys for the cipher come in one of three lengths: 128, 192, or 256 bits. For the purpose of achieving smaller design size, only a 128-bit key is supported in this design. This particular AES module is a part of a cryptographic project developed at the University of Tennessee [13].

Table 3.1 shows the LEON2 memory address space. The address space for the APB Bridge is from 0x80000000-0x8FFFFFFF. Since the user IP blocks (AES and rMat) have been attached on the APB, they have to be assigned this part of the memory space.

3.5 Artisan RAM

The cache system and the register file are implemented by using technology-dependent RAM cells. The Artisan RAM generator for the 180-nm IBM 7RF process has been used for the generation of synthesizable RAM models.
To utilize this SRAM in our design it is required to understand its read and write operation cycle and then create a wrapper to enable the communication with the LEON-2 processor.

Figure 3.5 describes the read operation in Artisan SRAM. To perform a read operation an important thing to notice is that address of the memory location to be accessed should already be there when the rising edge of the clock appears.

Table 3.1 LEON2 Memory Address Space

Address Range	*Size*	*Mapping*	*Modules*
0x00000000-0x1FFFFFFF	512 M	PROM	Memory Controller
0x20000000-0x3FFFFFFF	512 M	Memory Bus I/O	Memory Controller
0x40000000-0x7FFFFFFF	1 G	SRAM/SDRAM	Memory Controller
0x80000000-0x8FFFFFFF	256 M	On-chip Registers	APB bridge
0x90000000-0x9FFFFFFF	256 M	Debug Support Unit	DSU
0xB0000000-0xB001FFFF	128 K	Ethernet MAC Registers	Ethernet

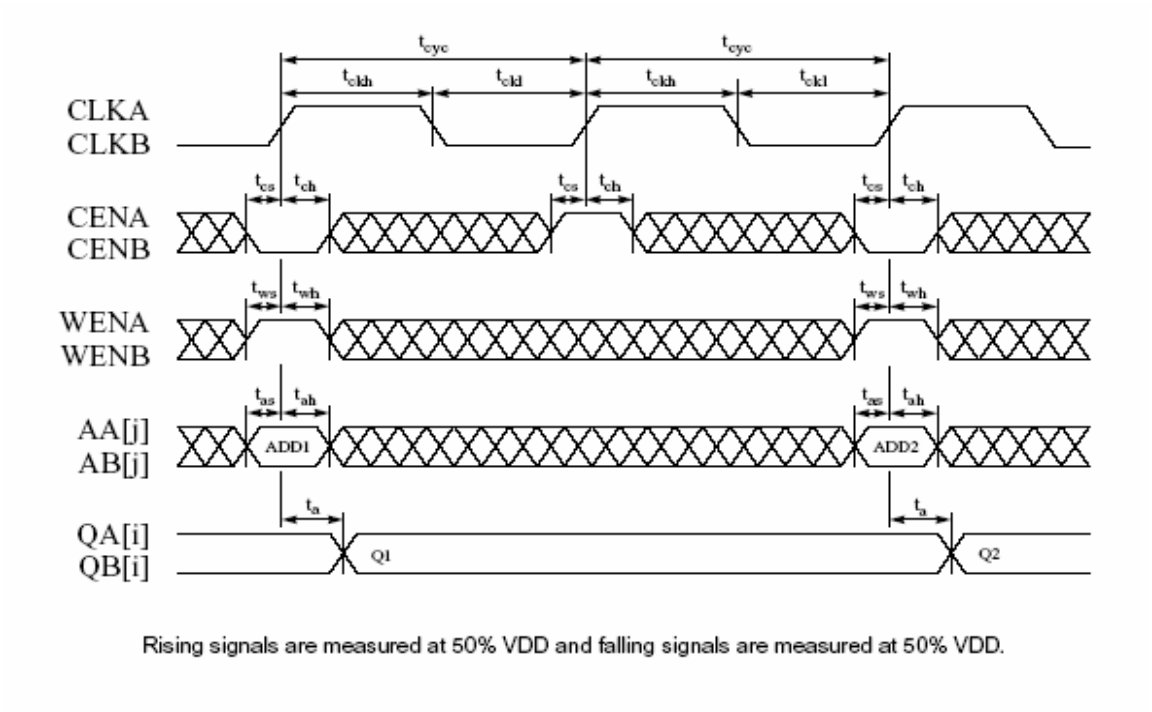


Figure 3.5 Artisan RAM Read Cycle

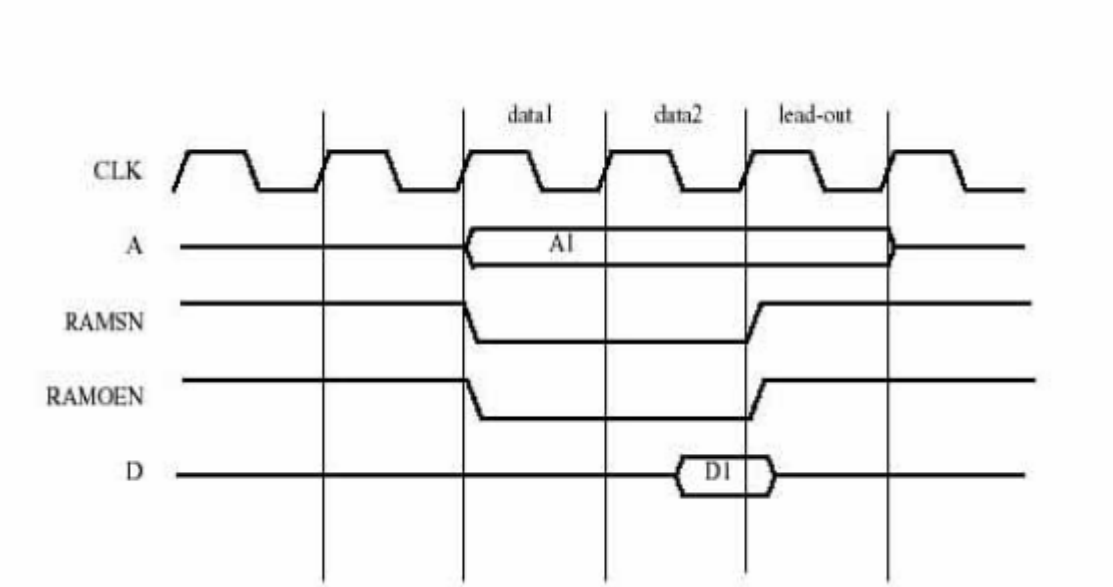


Figure 3.6 LEON-2 Processor Read Cycle

Similarly while writing to a memory location at the rising edge both data and address location should already be there at the data and address bus I/O ports. However the simulation of LEON-2 processor read cycle in Figure 3.6 shows that it loads the address and data I/Os at the rising edge of the clock. This caused a failure in the LEON-2 processor. Therefore a wrapper has been created, which acts as an interface between LEON-2 and Artisan RAM [14].

Chapter 4 Volunteer SoC Debug using DAFCA Tools

4.1 Introduction to the DAFCA Tools

DAFCA tools can be divided into two categories: pre-silicon tools and post - silicon tools.

Pre - silicon tools enable a designer to instrument a RTL design with reconfigurable debug infrastructure (ReDI) logic. This logic takes the following two forms:

• ReDI library blocks provided by DAFCA - fixed function blocks that are required for specific debug applications.
• Customizable logic blocks generated with the DAFCA pre-silicon tools which are highly configurable blocks that are tuned for a designer's special debugging requirements.

This instrumentation enables the post-silicon debugger/tester to detect, isolate, and repair bugs, as well as accelerate verification using Post-Silicon tools. It provides at-speed access to internal signals on the chip, delivers instrumentation to trigger and capture events, and creates reconfigurable structures that can be used for everything from assertions to on-the-fly repairs to signal generation [14].

The 'Personality Editor' package of the post -silicon tool enables a designer to program the reconfigurable logic blocks to realize assertions, logic modifications and fixes. At-speed patterns can be run through the system, by executing system software. The internal state, recorded by the Tracer in the debug module is then available for examination through the debug environment of NOVAS Debussy.

The user selects a net or set of nets to be to be observed, and the DAFCA tools automate routing the nets through interstitial PAN (parallel access network) network, programming triggers, and starting the tracer block. The result is that the SoC is no longer a black box. The user has the ability to debug the silicon at-speed, in the system, using the real logic and regains visibility to the signals that had become inaccessible.

4.2 Pre - silicon Tools

The pre-silicon tools are used for the insertion of ReDI logic to a SoC. User logic can be "wrapped" or "tapped" with DAFCA instruments. Wrapping refers to the insertion of an instrument on a signal/port that introduces a mux element in the path. Tapping refers to the insertion of an instrument that taps off a signal/port. No new elements are explicitly inserted within the existing path [9].

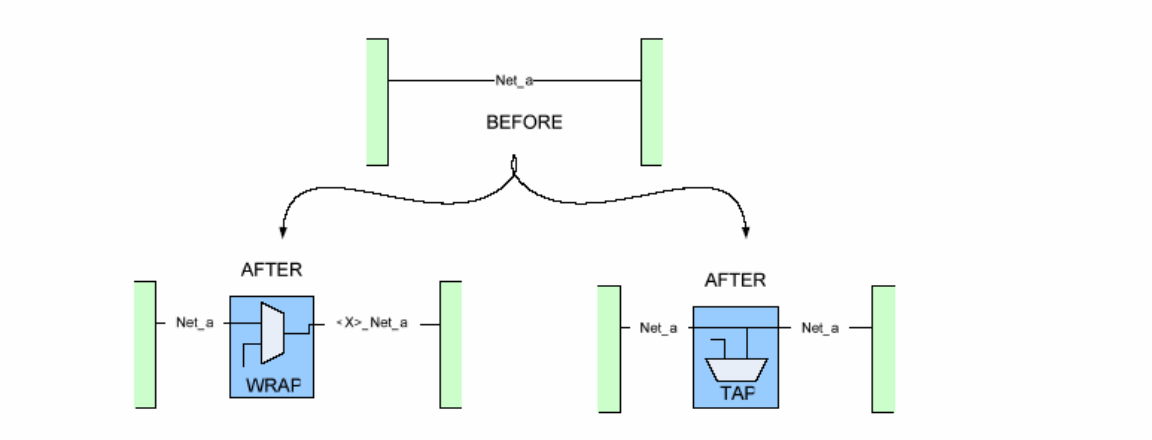


Figure 4.1 Wrap versus Tap

Figure 4.1 illustrates the difference between wrapping and tapping.

As stated in the previous section ReDI logic can be divided into DAFCA provided logic blocks and Customizable logic. DAFCA provided logic blocks include:

1. **Primary Controller (PCON)** – PCON is a part of the Access Mechanism which is communication channel instrumented on-chip and used to transfer control and data information between off-chip tools and on-chip instruments. The Access Mechanism is serial and utilizes a JTAG port for connectivity to the primary I/O. The PCON provides the interface and control between the JTAG TAP and the rest of the DAFCA infrastructure.

2. **Serial Access Node (SAN) -** The SAN provides the interface between end-point instrumentation and the serial access channel. All DAFCA instruments have serial chains for configuration and control.

3. **Monitor** - The Monitor is the programmable “controller” used to manage assertions, triggers and Tracer activity within the Debug Module (module used for on-chip logic analysis). Although the size of the rMonitor is about 600 K gates it will pay off to have an on-chip logic analyzer for a multi-million gate design.

4. **Tracer** - The Tracer is part of the Debug Module and used for the storage of state information during logic analysis. The Tracer uses embedded memory for storage.

Customizable logic blocks include:

1. **rWRAP** - rWRAP is a one-dimensional array of reconfigurable logic blocks. The number of blocks in the array is user-specified. Each reconfigurable

logic block includes configuration registers, logic, and routing resources that can be programmed to perform debug and validation functions during post-silicon verification.

2. **rMATRIX** - rMATRIX is a two-dimensional array of reconfigurable logic blocks. The number of blocks in the array (rows and columns) is user-specified. Whereas the rWRAP may be routing or logic resource limited due to the one-dimensional architecture, rMATRIX provides a richer set of logic and routing resources. As such, a wider range of functions (such as complex assertions) can be programmed into this resource.

3. **CMUX** - The CMUX is a collection of 2:1 muxes, pipeline registers and configuration registers. It is used to observe the critical ports/signals in a circuit. A CMUX cannot be used for controlling ports/signals like an rWRAP.

4. **r1500** - r1500 instruments provide resources for basic observation and control. An r1500 cell can be composed of one or more scan chain registers, an update register and control muxes.

As Table 4.1 suggests the r1500, rWRAP and rMatrix introduce an extra mux delay to the design, while rMux does not. This is because of the difference between the nature of tapping and wrapping a signal.

Table 4.1 Instrument Capability and Attributes

		Instrument Capability			Instrument Attributes			
		Observe User Logic	Control User logic	Modify User logic	Programmable	Mux Delay	Load	Size (gates)
ReDI™ Instruments	rMUX	✓			✓		✓	10 - 100 / signal
	r1500	✓	✓			✓		20 - 100 / signal
	rWRAP	✓	✓	✓	✓	✓		100 - 500 / signal
	rMATRIX[1]	✓	✓	✓	✓	✓		500 - 2000 / signal
	rMonitor[2]	✓	✓		✓			198k / chip

4.3 Instrumentation of the Volunteer SoC Using Pre-Silicon Tools

The SoC I started working with had already been instrumented with some ReDI blocks by Wei Jiang as a part of his thesis work. The design had two rWraps (XRW_Leon1 and XRW_Leon2) and 3 CMUXs (XCMUX_Leon1,XCMUX_Leon2 and XCMUX_Leon3).77 ports had been wrapped and 672 ports had been tapped using these wrappers and MUXs respectively.

The instrumentation of the ReDI block results in a larger design size. The number of extra gates depends on the choice of ReDI blocks and instrumentation decision. Designers should have a basic guideline in the design process to decide what signals or ports need to be instrumented and what kind of wrappers should be used. As a principle consideration, it makes perfect sense that the debug logic should be inserted in relative new and more complex logic, which has a high risk of failure and may be harder to be verified [13]. In this system design, since the IP blocks were pre-verified, they were less likely to have problems after silicon manufacture. So critical control logic (e.g. AMBA interface, APB Bridge, AHB arbiter) was instead wrapped since it was more problem prone. Similarly as the data transfer is generally straight forward and does not have much scope for logic error, the AMBA data bus was tapped (and not wrapped) to monitor the data transfer. This was also done to save silicon space as a CMUX occupies much less area (100 u^2/port) than an rWrap (4000 u^2/port).

Another consideration is that although it is required to test the DAFCA debug approach, it cannot be predicted whether the error will occur in-silicon or not. In the case that no design or fabrication error occurs in the silicon, a mechanism is needed to test the ability of the DAFCA debug method. To solve this problem, I decided to introduce an rMatrix into the design. This rMatrix wraps the second IP block (rMat) which is the dummy IP block. The rMatrix can be used to deliberately insert errors into the rMat block and hence demonstrate the error locating and debug capabilities of the DAFCA tools. The DAFCA Post-Silicon tools have a package called the ‘Personality Editor’ which can also be used to inject any desired personalities (adder, counter etc.) into this rMatrix and hence enable it to work as an eFPGA (embedded FPGA) thus enhancing the versatility of the SoC.

In addition to the wrappers, MUXes and the rMatrix, the design also has a debug module comprised of the Monitor and the Tracer and a PCON and SAN for off-chip communication.

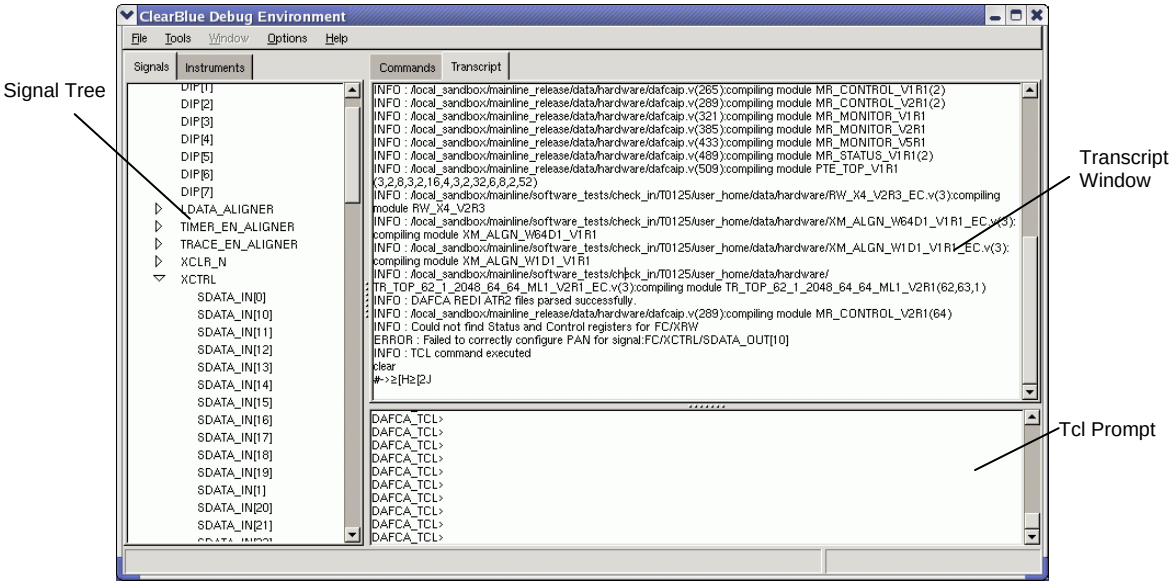


Figure 4.2 Post-Silicon Tools Interface

4.4 Post-Silicon Tools

Post-silicon tools utilize the ReDI blocks inserted into the design using pre-silicon tools to isolate and repair bugs. They have a GUI with an interface as shown in Figure 4.2.

The Debug Environment GUI has three main areas [16]:

- Signal Tree – Located on the left side of the main window, the signal tree shows all wrapped and tapped signals in the current design hierarchically. The signals that have been routed through a wrapper are shown in green.

  The Instruments tab displays a list of all ReDI instruments that were inserted into the design.

- Command Log – The top-right area of the main window shows the Debug Environment commands exactly as they are executed.

The Transcript tab provides the details of the debug session—not only the commands that are issued, but also the specifics of each route that is made (that is, signal sources and destinations). If there are any problems during the session, the errors are displayed in the Transcript.

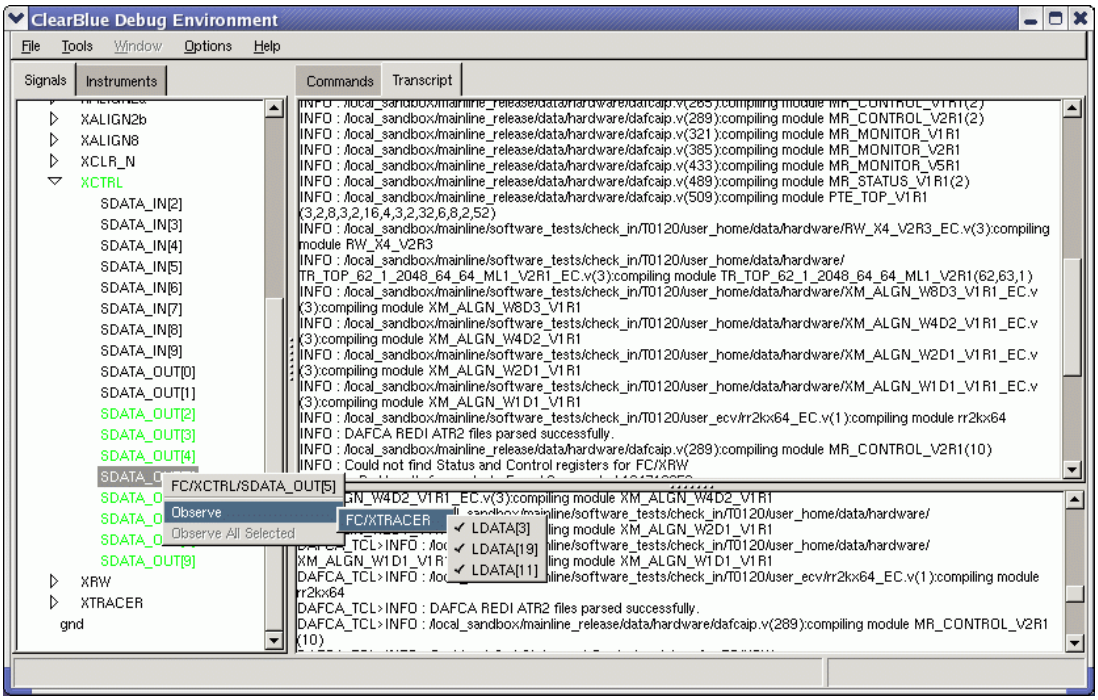


Figure 4.3 Routing Signals

Tcl Prompt – The bottom-right area of the main window provides a Tcl prompt that can be used to issue Debug Environment commands. The Tcl prompt has a multi-line mode, which allows access by placing the cursor in the window and clicking the right mouse button. Once in multi-line mode, multiple commands can be issued without them being executed. To execute the commands, one can press Shift-Enter while in multi-line mode or can switch back to single-line mode and then press Enter.

To route signals:

1. Expand the signal tree hierarchy to locate the signals that have to be routed.

2. Right-click the selected signal to display a drop-down menu. Select observe, and the instrument and port that has to be routed to. The full hierarchy of a routed signal displays in green in the Signals tree as shown in Figure 4.3.

After selecting all the signals, the routes can be saved to a .pan file which can be pulled up if needed in a later debug session.

Certain ReDI instruments have an associated graphical user interface (GUI) that simplifies the debug process. The instruments with an available GUI can be controlled either by using the GUI or the Tcl commands accessible from the CLI. An instrument's GUI can be accessed by clicking the instrument's name in the instrument browser. If the instrument is colored blue, it can be double-clicked to control it through the GUI.

4.4.1 Personality Editor

The Personality Editor is a GUI-based tool that can be used to create assertions and checkers to isolate design defects. A personality is essentially a specific configuration of a ReDI instrument. Currently, personalities can only be created for the rWrap instrument. The Personality Editor may be invoked from the instrument's application within the Debug Environment GUI.

To access the Personality Editor from the Debug Environment GUI:

1. Select the Instruments tab from the main window.

2. Double-click on the instrument (rWrap) with the left mouse button to invoke the wrapper's GUI.

3. Select Configure/Launch Personality Editor from the menu bar of the rWrap GUI.

The Personality Editor interface is shown in Figure 4.4. There are three main views of the Personality Editor, which work together to provide a complete picture of the instrument and its functions as they are configured.

The three views are:

a) Block View

An rWrap instrument is composed of a set of blocks, which display along the bottom of the Personality Editor. All input and output signals are noted on each block; lines that connect to status and control bits appear in green. This area of the Personality Editor provides an overview of the instrument, that is, what functions are configured for the instrument and in which blocks.

Figure 4.5 shows an example block that has been configured and its accompanying logic (from the Logic View).Cells in the block correspond to the cells in the Logic View of that block. The up arrow icon appears in the left side of a cell if a carry chain is in use. The flip-flop icon appears in the right side of a cell if a flop is in use.

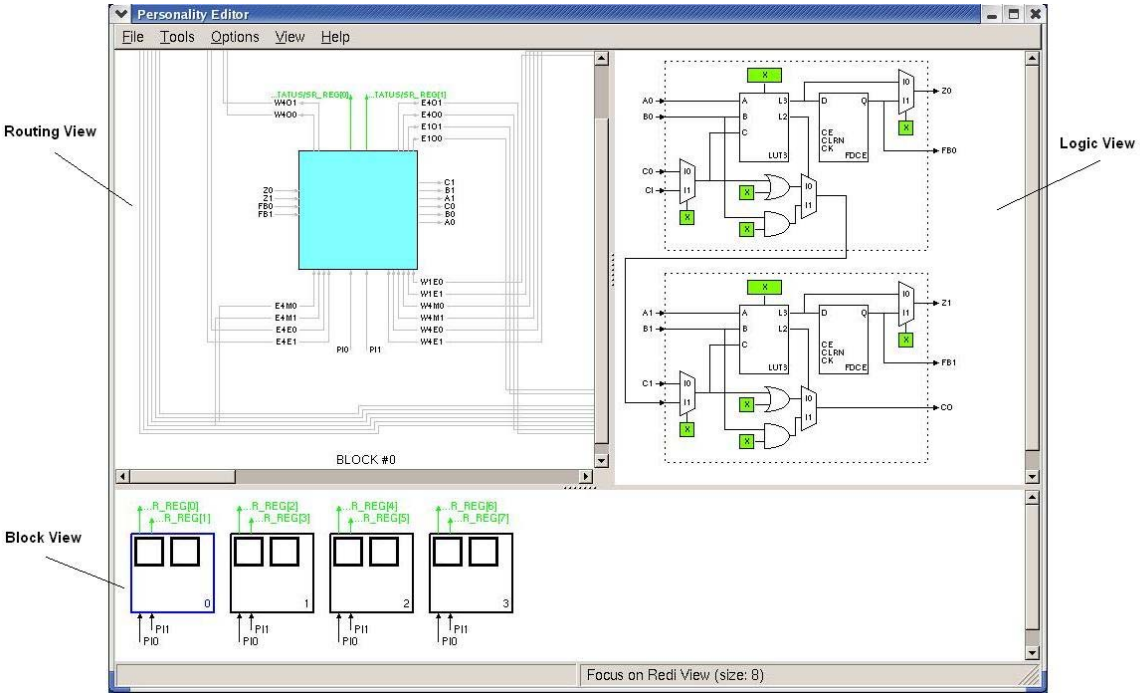


Figure 4.4 Personality Editor

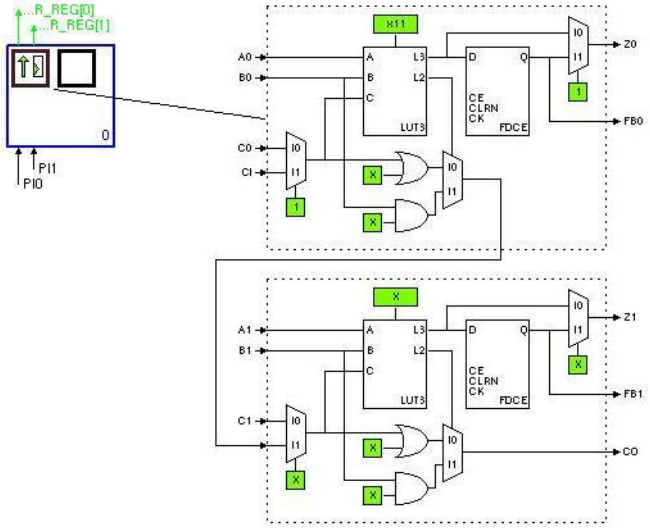


Figure 4.5 Example Design Block with Associated Logic View

b) Logic View

The Logic View, located in the top right of the Personality Editor, displays the logic for the selected block as shown in the following screen shot. Figure 4.6 shows an example logic view. Cells displayed in green may be set to a value by left clicking on them.

A logical function of a, b, and c can be entered for the look-up table using standard logical operators: & = and, | = or, + = xor, ~ = not. Right-click on the look-up table to display the LUT function entry dialog which appears as shown in Figure 4.7. Once we have entered the logical function for a cell, it is displayed in actual bit values in hexadecimal format.

c) Routing View

The Routing View, located in the top-left of the Personality Editor, displays all routes between blocks within the instrument as shown in Figure 4.8.The routing done in the Personality Editor is different from that in the Debug Environment. In the Debug Environment, signals are routed up to the perimeter of an instrument. In the Personality Editor, ports are routed within an instrument (in effect, interconnecting logic within the instrument).

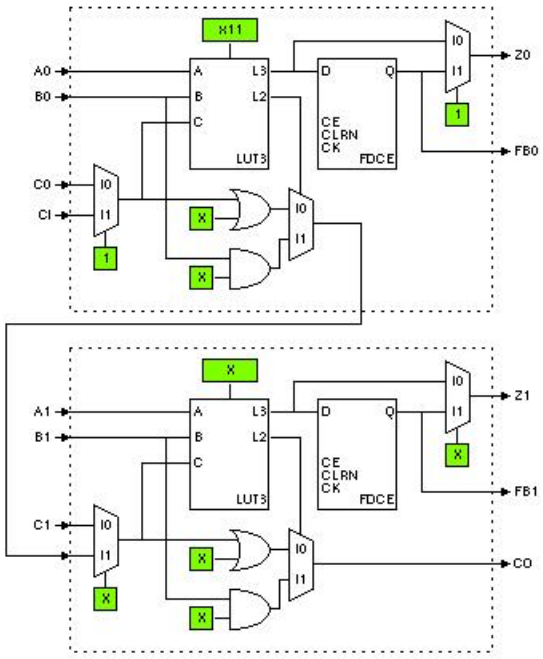


Figure 4.6 Example Logic View

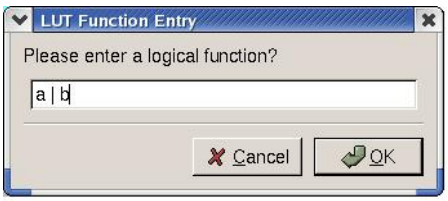


Figure 4.7 LUT Function Entry Dialog

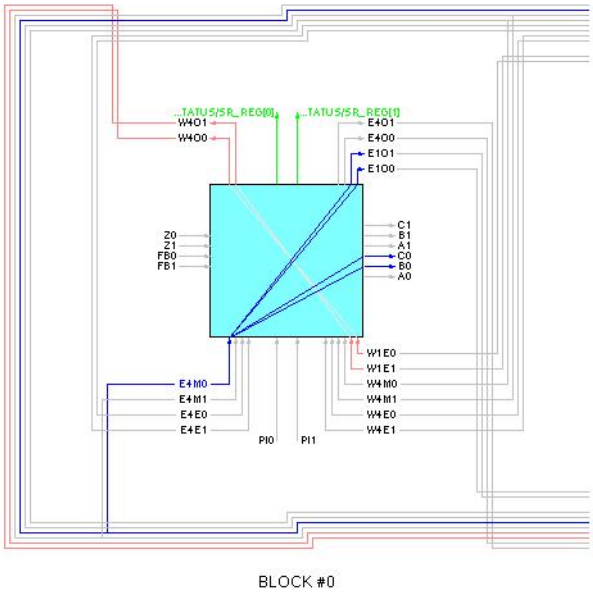


Figure 4.8 Example Routing View

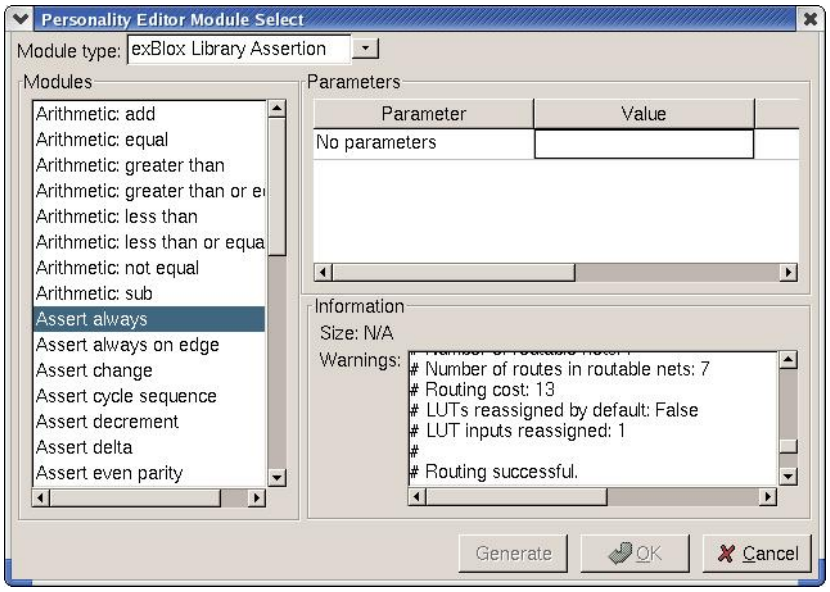


Figure 4.9 Module Insertion Dialog

To view possible routes for a port, select the port name with the left mouse button. All legal routes are highlighted in blue. To route one port to another, left click on the start port and then left click on the end port. The routes are highlighted in red.

Apart from designing personalities manually using the Logic and Routing View as discussed previously, pre-defined personalities (programmed using Python) can also be inserted into the wrappers. There are 32 such ready-to-use personalities (assertions) available with the Personality Editor. The ‘Insert Module’ option on the ‘Tools’ menu bar in the Personality Editor allows one to insert these pre-defined modules into the instrument. Figure 4.9 shows this module insertion dialog box. Once the desired personality is injected into the Wrapper, its MODE bit can be set to 1 to activate the built-in personality.

The post-silicon tools also provide integration into third party tools like NOVAS Debussy for debug purposes. Waveforms for the signals added to the tracer can be observed using Debussy, which makes debugging easier as it provides active annotation and active tracing of the signals. It is possible to click on the signal in the waveform and go back to the line of the HDL (hardware description language) code that is driving the signal at that time.

Chapter 5 Implementation

In this section I describe the detailed implementation steps of the design. At each step of the implementation, verification is done with simulation to ensure the correctness of the final design. The design is implemented targeting the IBM 7RF 180-nm process. The target process can be changed easily since the design is technology-independent.

As stated in the previous sections I started my work using the Volunteer SoC which has two IP blocks (AES and rMat) attached to it. The design also had some DAFCA ReDI fabric inserted into it. VHDL and Verilog are used in the design. I have used the Synopsys Design Compiler for synthesis and Cadence SoC Encounter for place and route of the design. Simulation has been done using Cadence NCSim. DAFCA Post - Silicon tools use several C++ function calls and NCSim supports the Programming Language Interface (PLI) of the Post - Silicon tools. This is the primary reason for using NCSim.

5.1 Customizing the LEON2 Processor

5.1.1 Configuration

The LEON model is highly configurable and hence can be customized for a certain application or target technology. A graphical configuration tool based on the linux kernel *tkconfig* scripts is used to configure the model. After a configuration has been saved, the corresponding VHDL configuration file (device.vhd) is generated.

The graphical configuration tool can only be used to configure LEON for the Xilinx VIRTEX, Atmel ATC, UMC FS90A/B, UMC 0.18 um CMOS, TSMC 0.25 um, Actel Proasic FPGA or the Actel AX anti-fuse FPGA technology. But the technology I was looking at was the IBM 7RF 180-nm process and so I could not use the tkconfig scripts to configure LEON but had to do so manually.

LEON uses three types of technology-dependant cells; RAMs for the cache memories, 3-port register files for the IU registers, and pads. These cells can either be inferred by the synthesis tool or directly instantiated from a target specific library. The selection of instantiation method and target library is done through the configuration record in the TARGET package.

To port to a new technology:

1. A technology-dependent package should be added, exporting the proper cell generators. I achieved this by creating a file tech_ibm.vhd containing the IBM 7RF specific RAMs, regfile, and pads. In my course work I had customized LEON

for the TSMC 0.25 um process where the IE and OE pins of the pads were active low. But the pad documentation for the IBM process suggested that these pins were active high and hence I wrote the behavioral models of the pads taking into account the new polarity. The technology-dependant packages can be seen as wrappers around the mega cells provided by the target technology or synthesis tool. The wrappers are then called from *tech_map.vhd*, where the selection is done depending on the configured synthesis method and target technology.

2. The *target.vhd* should be edited to include the new technology or synthesis tool in *targettechs*. Shown below are a few lines from my *target.vhd*. The new technology 'ibm' has been added to *targettechs*.

```
library IEEE;
use IEEE.std_logic_1164.all;

package target is
type targettechs is
  (gen, virtex, virtex2, atc35, atc25, atc18, fs90, umc18, tsmc25, proasic,
axcel,ibm);
```

3. The *tech_map.vhd* should be edited to instantiate the cells when the technology is selected. Shown below are a few lines from my *tech_map.vhd* which indicate dpram instantiaton.

```
ibm7rf : if TARGET_TECH = ibm generate
     u0 : ibm_dpram generic map (abits => abits, dbits => dbits)
         port map (address1, clk1, zero, dataout1, enable1, zero(0),
                  address2, clk2, datain2, open, enable2, write2);
   end generate;
```

4. The *device.vhd* should be edited to define and select the configuration using the new technology.

In our LEON design, optional components such as FPU, Hardware Multiplier, and Debug Units were not selected during configuration. Instruction and data cache were set to be 1K, single set and 8 words per cache line.

5.1.2 Artisan RAM

The instruction/data caches are implemented as Single Port RAM (SPRAM) blocks and the register file is designed as two Dual-Port RAM (DPRAM) blocks. As the previous section states, the RAMs are technology-dependent and have to be instantiated by the user. The RAM blocks that come with the LEON2 package are behavioral models that can be used only for simulation. In order to synthesize the design these RAM blocks have been replaced with the ones generated by the

Artisan RAM generator for the IBM 7RF process. The size of the RAM block needed depends on the size of the I/D cache and register file. We have used two 136x32 DPRAMs, two 256x32 SRAMs and two 32x30 SRAMs as hard macros in our design.

Five views of the RAM blocks were generated using the Artisan RAM generator. They were: Verilog Model for simulation, Synopsys Model for logic synthesis, TLF Model for timing analysis, VCLEF footprint for Physical Design tools and GDSII layout for final tape out. The specifications used are shown in Table 5.1.

As discussed in section 3.5, the RAM blocks cannot be used directly because of the timing specifications difference between the Artisan RAM and the Leon. Hence, they have been wrapped with a RAM block wrapper so that they can communicate with the LEON2 in the same fashion as the behavioral model. Then the Synopsys model is converted into the Synopsys database format to be used for logic synthesis.

5.2 The Integrated SoC

As mentioned in Chapter 3, our design has a 128-bit AES decryption module and a dummy rMat block as the two user IP blocks and these are defined as additional masters on the AHB bus and slaves on the APB bus. The user IP blocks have been instantiated in the top level VHDL file (*mcore.vhd*), in order for Leon to recognize the extra master on AHB. For the APB connection, the APB/AHB Bridge has been modified to define the memory mapping for the APB signals of AES block [13].

The AMBA interface of our SoC is designed to take care of AMBA signals and pass the input/output data between the AMBA bus and the IP blocks. It is a state machine which performs several sequential steps. For AES, the steps are [13]:

1) Initialize, Enable AES module and wait for start request signal on APB.
2) Read in control data through APB (input data memory address).
3) Request AHB bus and transfer data upon bus granted.
4) Load key/data into AES and signal AES to start.
5) Read in AES output when AES finishes, signal system that output ready.
6) Write back output data to the memory address that system specified.
7) Go back to initial state and wait for next call.

In order to have the system recognize the additional bus masters and slaves, the IP blocks had to be connected to the bus arbiter and the APB Bridge had to be modified to re-define the peripheral memory mapping. The IP blocks were instantiated and connected to the AHB arbiter with a unique index. The arbiter's arbitration scheme is fixed and a higher index has higher priority. The CPU is

Table 5.1 Block RAM Parameters

Components	Register File	Cache Data	Cache Tag
Instance Name	dpram136x32	ram256x32	ram32x30
Depth	136	256	32
Width	32	32	30
Frequency (MHz)	50	50	50
Multiplexer Width	4	4	8

Table 5.2 IP Block APB Memory Mapping

IP blocks	Memory Range	IRQ	Index
rMAT	0x80000200-0x800002FF	15	2
AES	0x80000300-0x800003FF	13	1

the default master on the bus and has an index of zero, which means that the IP block has higher priority than the CPU when both of them are requesting the bus at the same time. As we see in Table 3.1, the memory range of the APB Bridge is between 0x80000000 and 0x8FFFFFFF, so the IP blocks have been assigned addresses within this range. The memory mapping and AHB master indexes of both the IP blocks are shown in Table 5.2.

5.3 Insertion of rMatrix to the Design

As mentioned in Section 4.3, the SoC I started working with had already been instrumented with two rWraps and 3 CMUXs. This ReDI fabric was inserted into the most critical and error prone portion of the design. The wrappers were inserted around the AMBA bus interface, AHB bus arbiter and APB bridge while the MUXs were inserted around the data bus.

The instrumented design is shown in Figure 5.1. In order to display the debug ability of DAFCA tools in a better way I have introduced an rMatrix into the design. The rMatrix can be seen in Figure 5.1 as the grey block next to the dummy IP block (rMat) and can be used to deliberately insert some errors or additional features into rMat. The rMatrix wraps the input and output ports of the rMat block and can also be used as an embedded FPGA block.

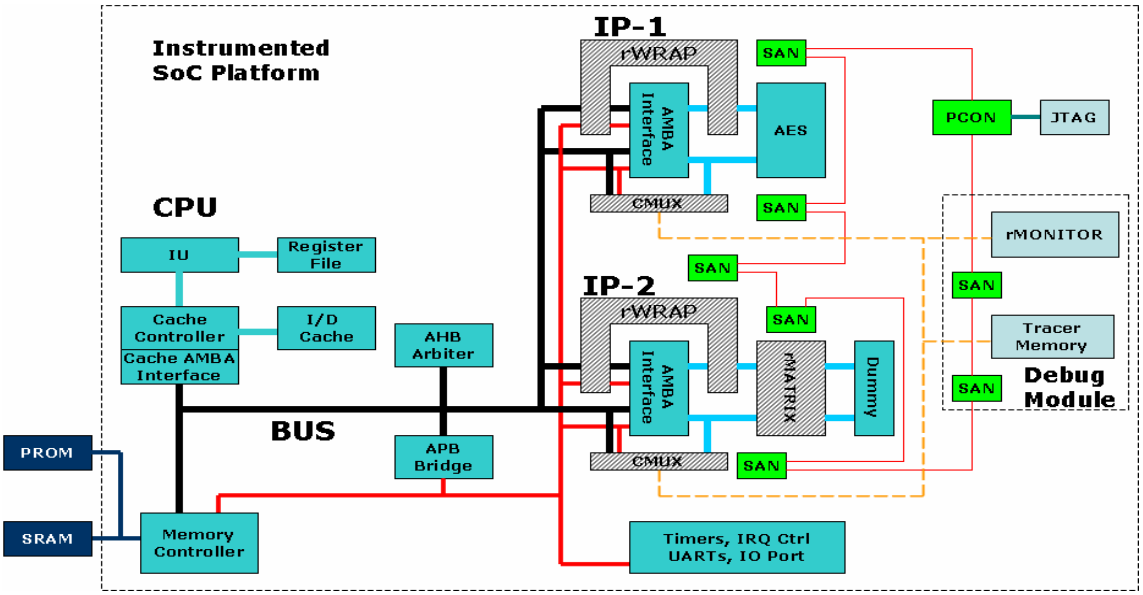


Figure 5.1 Instrumented SoC

After the design was instrumented, simulation was also done to make sure that the insertion of the DAFCA ReDI fabric did not affect the logic of the design.

5.4 Synthesis

Logic synthesis was done using Synopsys Design Compiler after the DAFCA IP blocks were inserted into the design. The synthesis script was written with tighter timing constraints to accommodate the delay that would be introduced due to the addition of the DAFCA logic. I have used the 'set_case_analysis' command to assign a constant value to some ports and pins so that the false paths in the design are ignored during timing analysis and optimization.

The bottom-up synthesis approach has been used for the on-chip debug circuitry. Because the muxes and D flip-flops are instantiated in the CMUX design and a large number of signals are being tapped, a large number of muxes and D flip-flops are instantiated in the design. Design Compiler automatically compiles hierarchical circuits without collapsing the hierarchy. After each module in the design is compiled, Design Compiler continues to optimize the circuit until the constraints are met. This process sometimes requires recompiling sub-designs on a critical path [17].To prevent this recompiling and hence save time in the synthesis process 'set_dont_touch' command has been used in the synthesis script. Hence each sub-design is synthesized and compiled only once and there is only one copy loaded in the memory for each sub-module. After the debug module is synthesized, top-down approach can be followed for the rest of the design in order to have better optimization. After the design is synthesized, a

gate-level verilog netlist and a timing constraint file (SDC) are written out to be used in the physical design.

5.5 Physical Place and Route

After synthesis, the physical place and route of the design was done using Cadence SOC Encounter. This included floorplanning, clock tree synthesis and timing analysis. The physical design flow is shown in Figure 5.2.

I started by importing the gate level net-list into Encounter. The timing files (.tlf) and lef files for the IBM 7RF standard cells, pads and RAM blocks were also imported. I have used the slow timing libraries for better timing closure on the design. In order to use the 132-pin package PGA132M from MOSIS, an I/O file was created for I/O pad assignments with I/Os evenly spread out on each side of the chip. The I/Os included 8 VDD and 8 VSS pins too. This I/O file was also imported into Encounter. Then floorplanning was initiated where the hard macro blocks (i.e. memory) were pre-placed. Then I proceeded with standard cell placement. After all the cells were placed, three clock trees were synthesized to minimize the clock skew. At this point trial route and timing analysis was used to check if the timing requirement was met. Very often there are timing violations so several iterations of in place optimization (IPO) are needed. The whole process is an iterative procedure and it is possible to go back to modify the initial floorplan, until the timing requirement is met. Then the final layout and timing delay file for post-layout simulation was generated.

5.5.1 Floorplanning, Powerplanning and Placement

The goal of floorplanning is to place the logic in ways that make the routing easier, less congested and shorter. The RAM blocks (register files and cache for LEON) have been placed and aligned in a corner. The tracer memory has been placed away from rest of the RAM blocks to separate the original baseline design from the DAFCA instrumentation. The basic principle used in the floorplan was to arrange the modules such that blocks with connections are placed closer to each other, especially for critical components such as the CPU, cache controller, memory controller, etc. At the same time, the original un-instrumented baseline design was separated from the DAFCA instruments, to minimize the impact of introducing extra logic into our design. Cache blocks for LEON were placed in the bottom left corner, while register file RAM blocks were placed in the middle left and the tracer memory has been placed in the top left corner. Block halos were added around these blocks to reduce congestion around the block. We also set the die size (4.784mm x 4.796mm) and the core to I/O boundary distance (100 um) in the ‘Specify Floorplan’ form. The die size was decided taking into account the size of the I/O Pad cells and the Pad filler cells

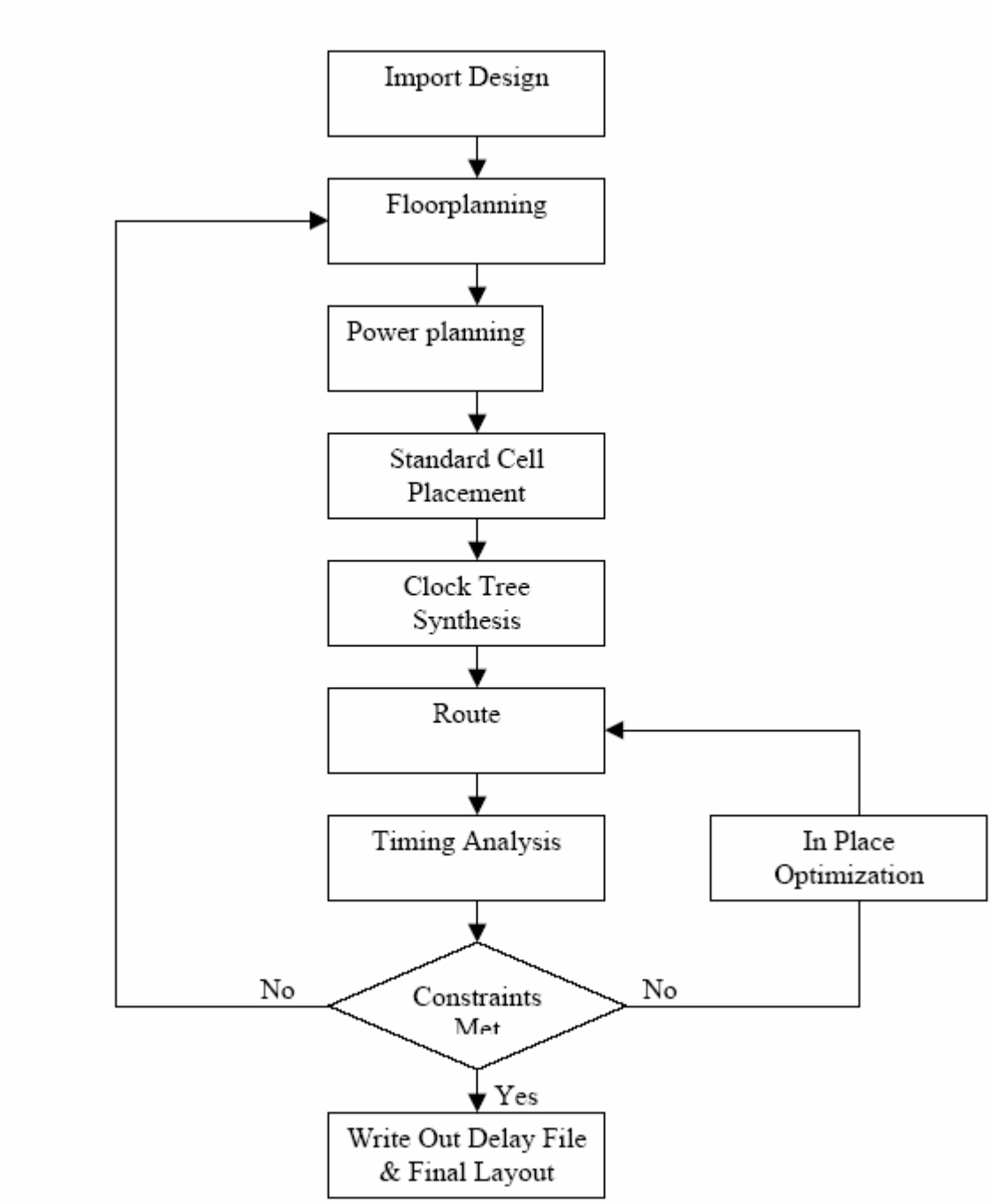


Figure 5.2 Physical Design Flow [13]

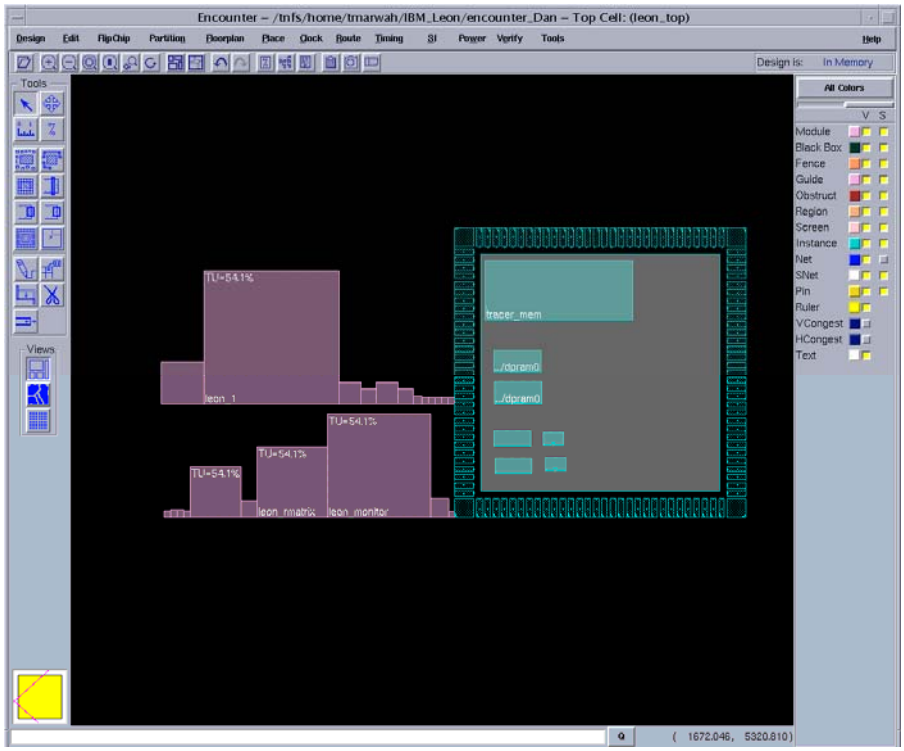


Figure 5.3 Floorplan Guide

Figure 5.3 shows the floorplan guides for the layout. Power rings were added around the core boundary. Power stripes were also added for easy access to VDD and GND but stripes were omitted inside the block rings.

Then amoeba placement was carried out on the design. It was a medium effort, timing-driven placement.

5.5.2 Clock Tree Synthesis

The clock tree was synthesized after the placement was done. I have generated an individual clock tree separately each for CLK (clock for LEON), PTCK (clock for JTAG controller) and PTRST_N (reset for the JTAG controller).The clock tree specification file for the CLK signal is listed below:

AutoCTSRootPin clk
NoGating NO
MaxDelay 2ns
MinDelay 0.5ps
MaxSkew 0.5ns
SinkMaxTran 3ns

BufMaxTran *3ns*
Buffer *BUFX1 BUFX2 BUFX3 BUFX4 BUFX8 BUFX12 BUFX16*
End

After the clock tree was synthesized, the modified net-list was saved and the clock phase delay is shown in Figure 5.4.

5.5.3 Routing

The design was routed using 6 layers of metal. I used NanoRoute for the process and performed global and detail routing. Global Route plans the global interconnect and detailed routing for the design using NanoRoute while Detail Route performs detailed routing on the design following the global routing plan. Timing analysis and in placement optimization were done to solve the timing violation after the routing. The final layout of the design is shown in Figure 5.5.

5.6 Simulation Result

The simulation of the design was done at the pre-synthesis, post-synthesis and post-layout steps to verify the functionality at each step. After the design was

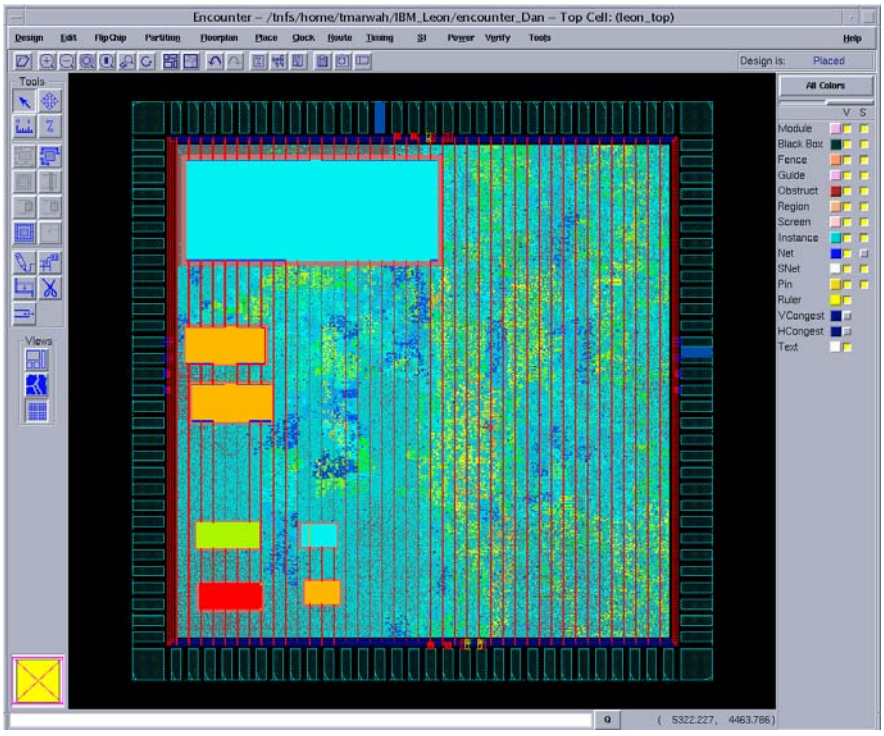


Figure 5.4 Clock Tree Phase Delay

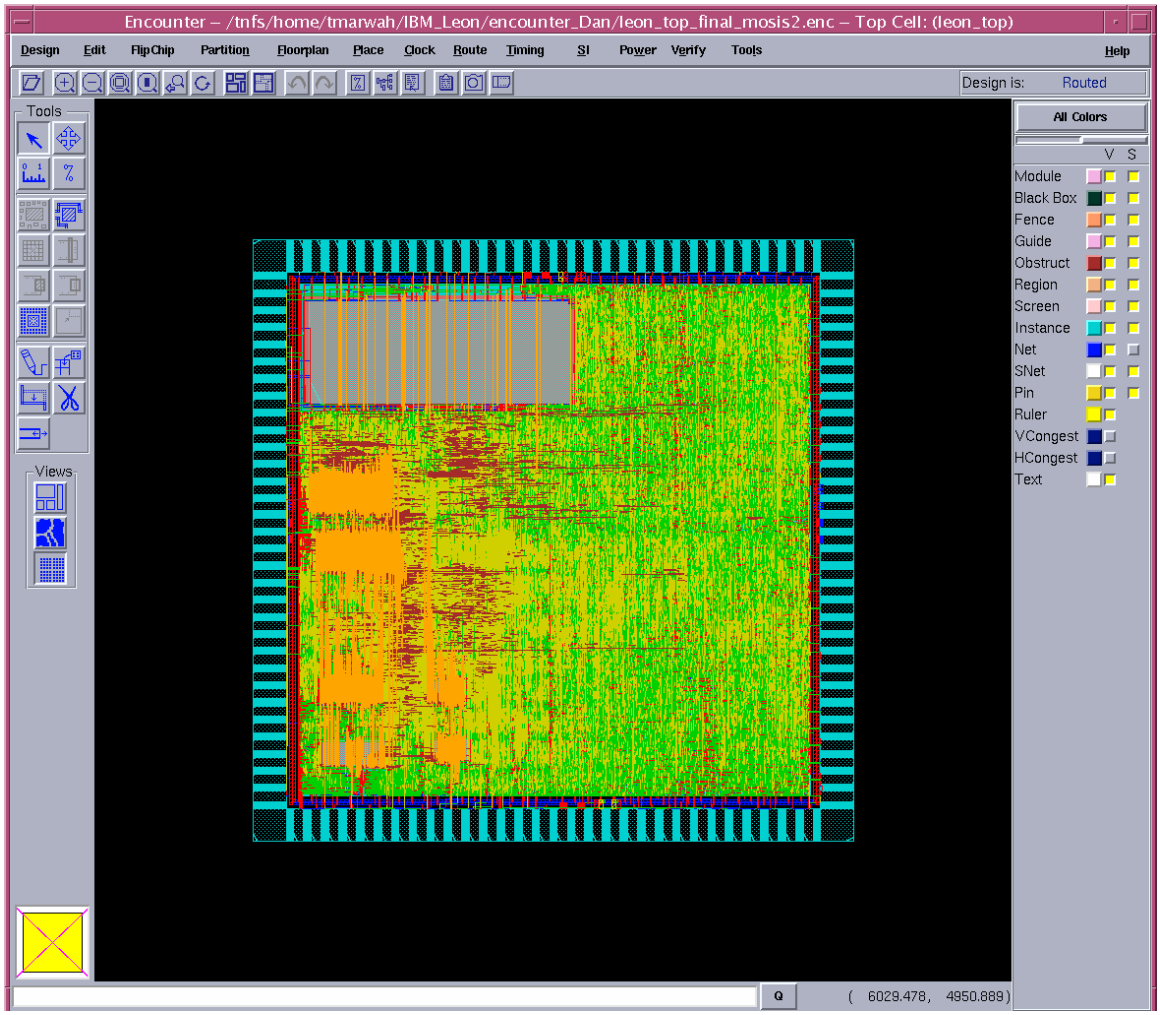


Figure 5.5 SoC Design Routed

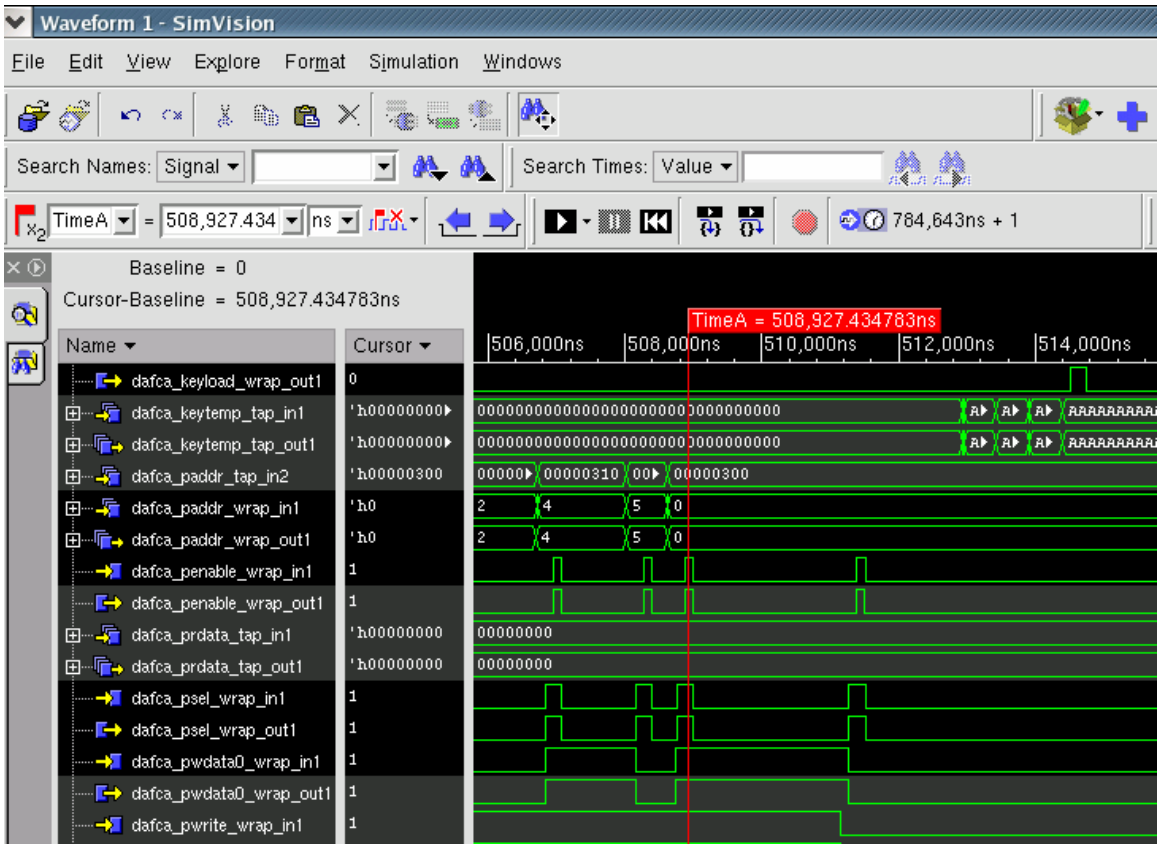
routed to the final layout, the timing information was extracted and written to a file in the Standard Delay Format (SDF). The SDF file was imported back to the simulator for back annotation of the simulation.

In order to test the design, the C testbench from LEON has been modified and the RAM image has been recompiled for simulation. After LEON boots up, it reads the data from memory, sends it to the AES block through the AHB bus and enables the GO signal through the APB bus. AES reads in the key and encrypted text, performs the decryption and sends the data back to RAM and flags the DONE signal. The post-layout simulation is shown in Figure 5.6a and 5.6b. The instrumentation of the DAFCA ReDI blocks has been done in the top level of the design. Hence we have routed all the signals that had to be tapped / wrapped to the top level of the design and then finally sent them back to their respective modules. All wrapped signals begin with the word ‘dafca’ and are named in the format ‘dafca_xxx_wrap_out’ or dafca_xxx_wrap_in’ where ‘xxx’ denotes the port/signal name. The ‘dafca_xxx_wrap_out’ are the signals that have been routed to the top level and appear at the input of the debug circuitry at the top level. The ‘dafca_xxx_wrap_in’ are the signals that are routed back to the inner modules after wrapping. A similar naming convention has been followed for the tapped signals. It can be seen in Figure 5.6a that at around 508.9 µs, ‘psel’, ‘penable’, ‘pwrite’ are 1, ‘apb_paddr (4 downto 2)’ = ‘000’ and ‘apb_pwddata (0)’ = 1, hence AES is enabled. It can be seen in Figure 5.6b that at around 509 µs, the AES requested the AHB bus. It is granted around 510 µs and data are transferred through the AHB bus. After the data are read, ‘data ready’ and ‘go’ become high. Then ‘kld’ (key load) becomes high which signals the AES block to load the encryption key. The data is converted into 128-bit encrypted cipher text and 128- bit key and sent to the AES module. At 521 µs, the AES module finishes decryption and the plain text is written out as ‘dataout’ and ‘done’ is high to signal the AMBA interface that the data is decrypted and ready to be sent back to memory.

5.7 Sending the Design to MOSIS

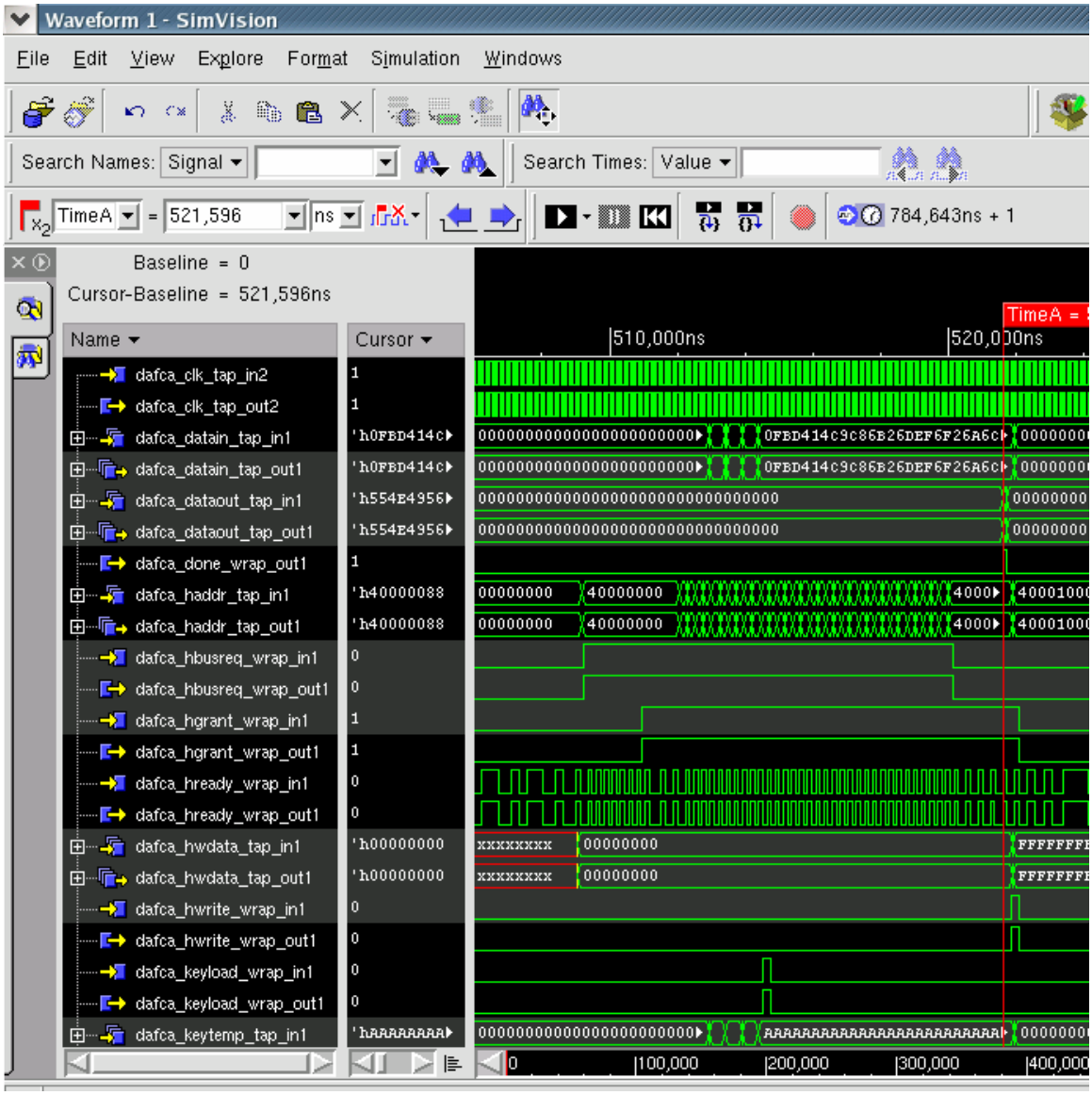
The SoC layout imported in Cadence Virtuoso is shown in Figure 5.7. A few modifications and pattern density checks had to be performed on the design before submitting it to MOSIS for fabrication. These were performed using Cadence Virtuoso. The GDS file for the layout that had been exported from First Encounter was imported into Cadence Virtuoso using the *‘gdstocds.map’* file provided with the IBM PDK. The modifications and checks performed included –

1. *Placement of a CHIPEDGE polygon around the Design* – MOSIS requires that the design should contain a CHIPEDGE shape which defines the correct size of the chip matrix. This shape is required for pattern density checking of the matrix. The CHIPEDGE requirement was met by creating a guardring (which contains its own CHIPEDGE polygon) around the chip.



(a)

Figure 5.6 Post-Layout Simulation



(b)

Figure 5.6 Continued

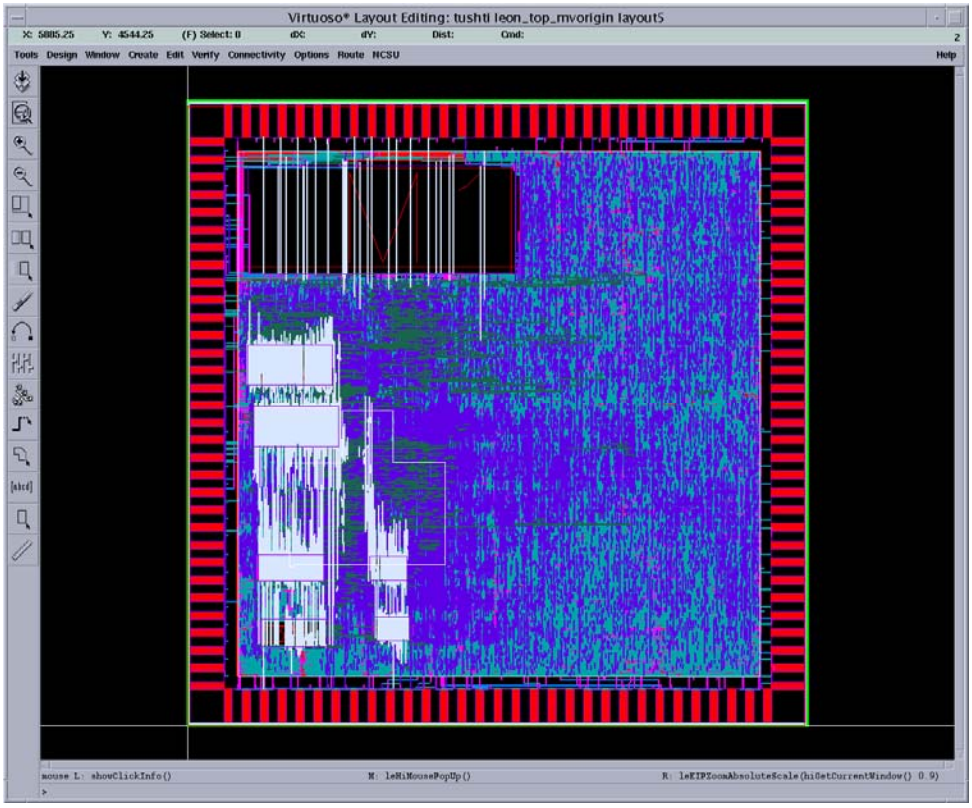


Figure 5.7 SoC Layout in Cadence Virtuoso

The guardring was created using the rectangular 'Image' Pcell provided in the IBM PDK. Finally, the chip origin (x=0, y=0) was placed at the lower left corner of the CHIPEDGE polygon. The layout with the CHIPEDGE polygon is shown in Figure 5.8.

2. *Pattern Density Rules* - Pattern density rules have been developed as a manufacturing requirement for production of all semiconductor products in order to better control the manufacturing process and assure control of manufacturing parameters (film thicknesses, etch control, linewidths etc.) and product yield [18].

Pattern density rules can be divided into-

a) Global Pattern Density Rules –

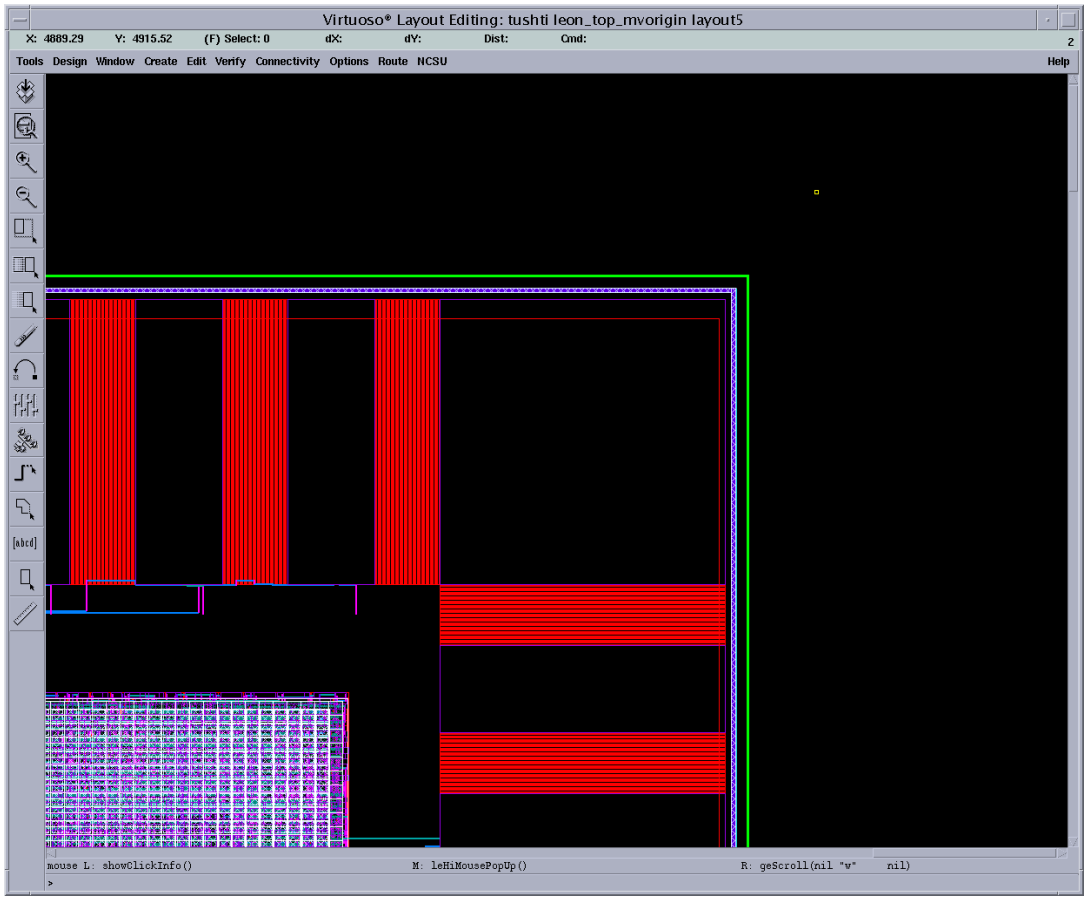


Figure 5.8 Guard Ring and CHIPEDGE Polygon

These rules require the designer to make sure that the density of MT and ML metals in the chip is >= THRES-GLOBAL-MIN (IBM 7RF specific minimum global density threshold value) and <= THRES-GLOBAL-MAX (IBM 7RF specific maximum global density threshold value) [18].The metals M1, M2, M3 and M4 are auto-filled by MOSIS. Figures 5.9 and 5.10 show the initial density of MT and ML metals respectively in our layout.

I met the density requirements for MT and ML by creating small dummy shapes using the MT and ML dg layers. The pattern density fill guidelines for the IBM 7RF process required that these dummy shapes should not be more than DA (area specified by the IBM 7RF process) um^2 in area [18]. Dummy shapes should be relatively small, especially on metal levels, to reduce cross-talk due to the additional level-to-level capacitive coupling.

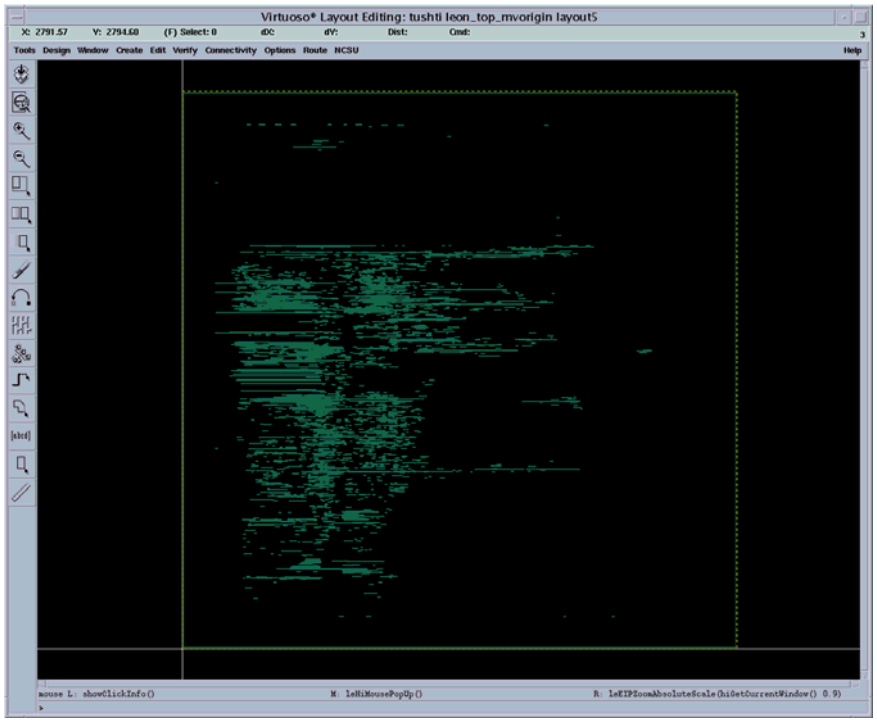


Figure 5.9 Initial MT Density in the SoC

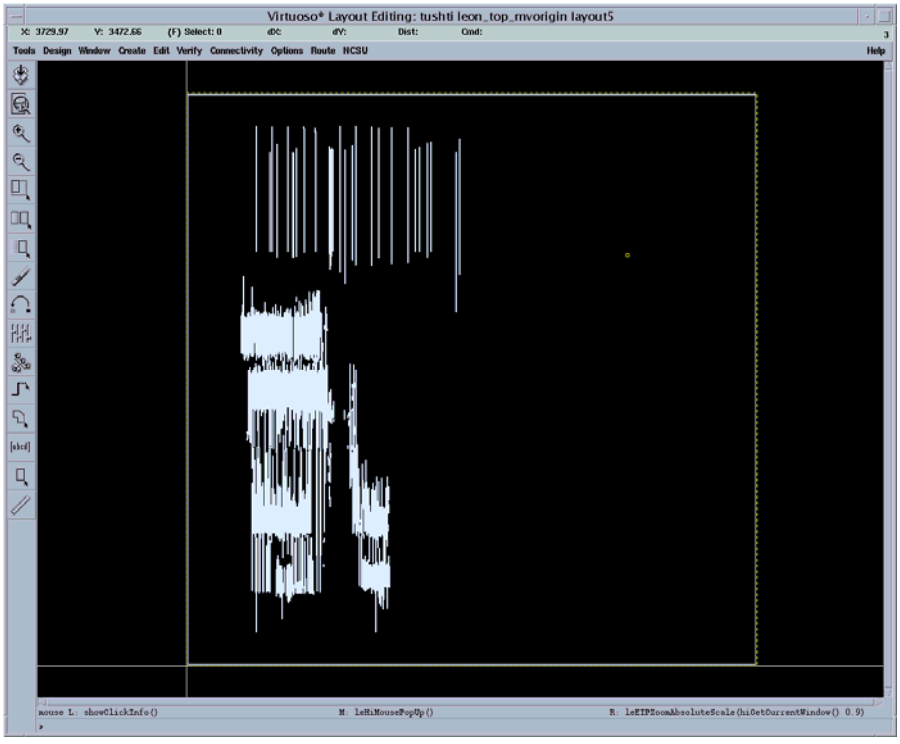


Figure 5.10 Initial ML Density in the SoC

Also they should be well-dispersed, if possible, to maximize process uniformity. I placed the dummy shapes as instances all over the layout till the global density requirements were met. The key point to be careful about was not to overlap these dummy shapes with the already present respective metal layers. Figures 5.11 and 5.12 show the layout with MT and ML fill respectively. The global pattern density test was performed by running the file ‘divaDensity.rul’ and the density requirements for ML and MT were met.

b) Local Pattern Density Rules –

These rules require that local density (minimum) for M2, M3, M4, M5 and MT metals checked in 400μm x 400μm checking boxes stepped in 400μm increments across the chip be greater than THRES-LOCAL (IBM 7RF specific minimum local density threshold value) [18].Our design met the M2, M3, M4 and M5 local densities by itself but the local density of MT had to be raised by insertion of MT dummy shapes uniformly throughout the chip. The local pattern density test was performed by running the file ‘divaLocalAlum.rul’.

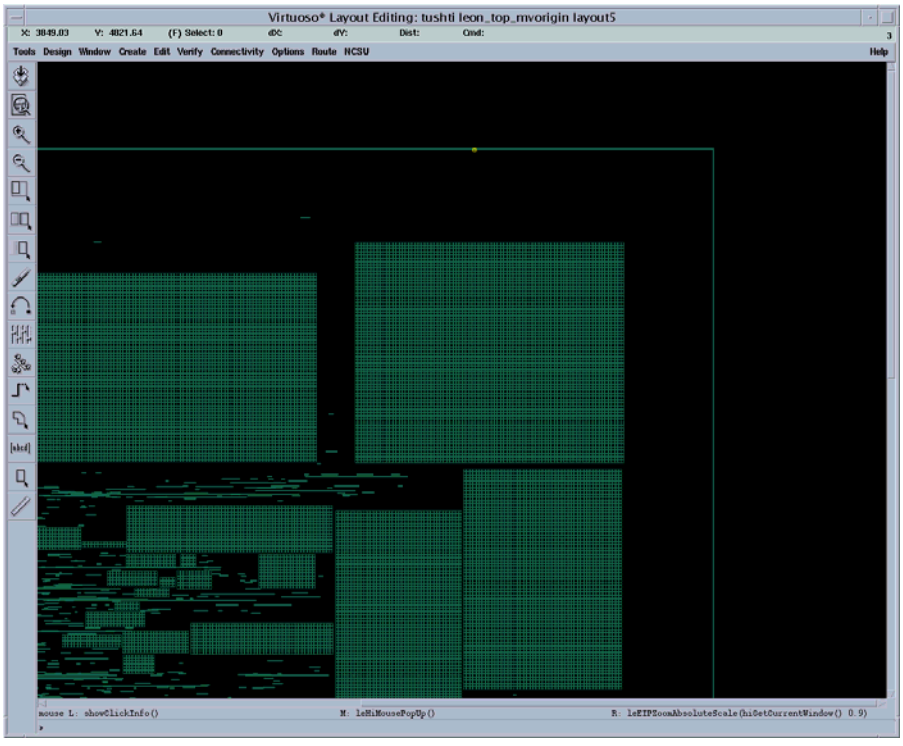


Figure 5.11 SoC with MT Fill

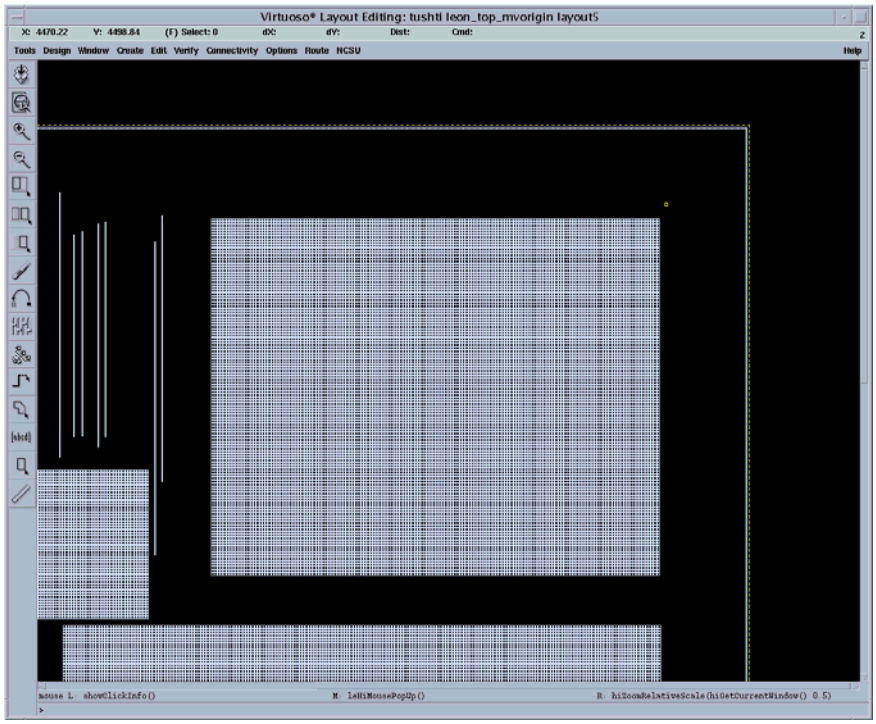


Figure 5.12 SoC with ML Fill

Chapter 6 Testing with Post-Silicon tools

The experiments with the post-silicon tool have been performed on the post-layout netlist due to unavailability of the fabricated chip at this time. The Personality Editor from the post-silicon tools suite has been used to program certain assertions and checkers by injecting different personalities into the rWraps. Post-silicon tools communicate with the Cadence simulator as if it is a real chip. I have modified the testbench to loop infinitely. Initially, I chose to bypass the testing structure and hence the circuit was functioning normally. Then I injected several personalities into the wrappers and could see the effect of that change in the functionality of the of the IP blocks. I also implemented several assertions with the help of wrappers to verify the functionality of critical signals. The Personality Editor for rMatrix is not available at the moment so I could not inject personalities into the rMatrix. Some of the tests I performed are:

6.1 Tests Performed

Forcing the bits [2], [3], [4] of the APB Address bus to '1'-

For AES to be enabled it is required that apb_paddr [2 - 4] be '0'. But with the help of the personality editor, I programmed the wrapper XRW_Leon1 such that these bits be stuck at 1 and as expected AES stopped working. Figure 6.1 shows the changes made in the wrapper using the Personality Editor. PO0 and PO1 are output pins of the wrapper block 3 that in turn connect to the bits[2] and [3] of the APB Address bus. Pin PO0 of block 4 connects to bit [4] of the address bus. As seen in the logic view on the right the cells have been set to the logic function 'FF' which implements the stuck at '1' function for the pins Z0 and Z1 for block 3 and Z0 for block 4. As these pins are connected to the PO's of the wrapper they aid in setting APB address bus [2], [3] and [4] bits to 1.

Figure 6.2 shows the waveform for the SoC after this change. As expected the AES has stopped working and no data input or output can be seen in the 'datain' and 'dataout' pins respectively. Also paddr[2],[3] and [4] are stuck at '1'.

Forcing the 'hgrant' signal of the AHB bus to '0'-

When AES is enabled, it waits for the 'hgrant' signal for the AHB bus which indicates that the bus has been granted. The AHB address bus carries the address of the data to be decrypted whereas the AHB data bus carries the data to be decrypted. With the help of the Personality Editor, I stuck the 'hgrant' signal to '0' and hence the AES stopped working.

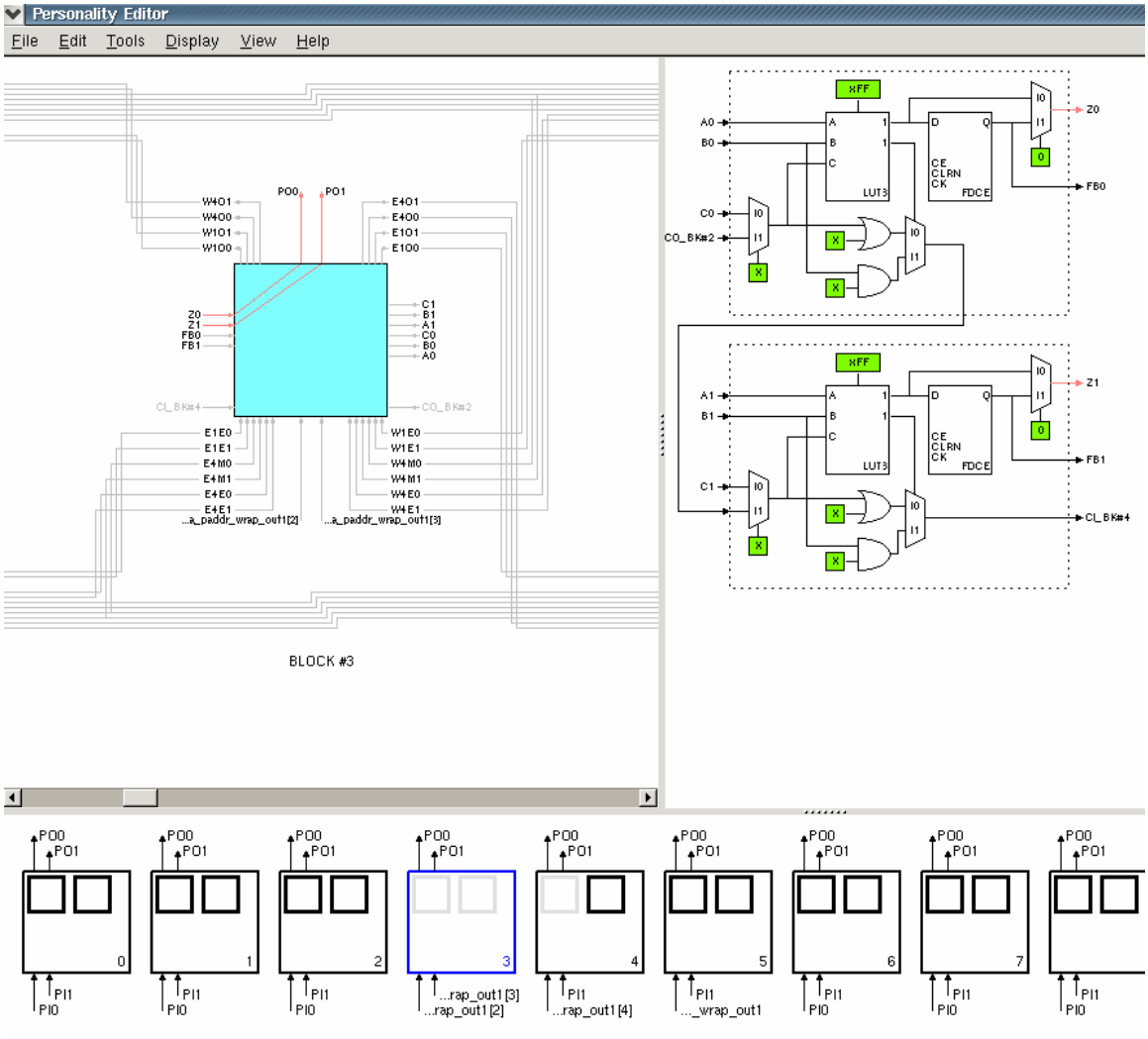


Figure 6.1 Personality Editor Window Depicting ‘paddr’ Modification

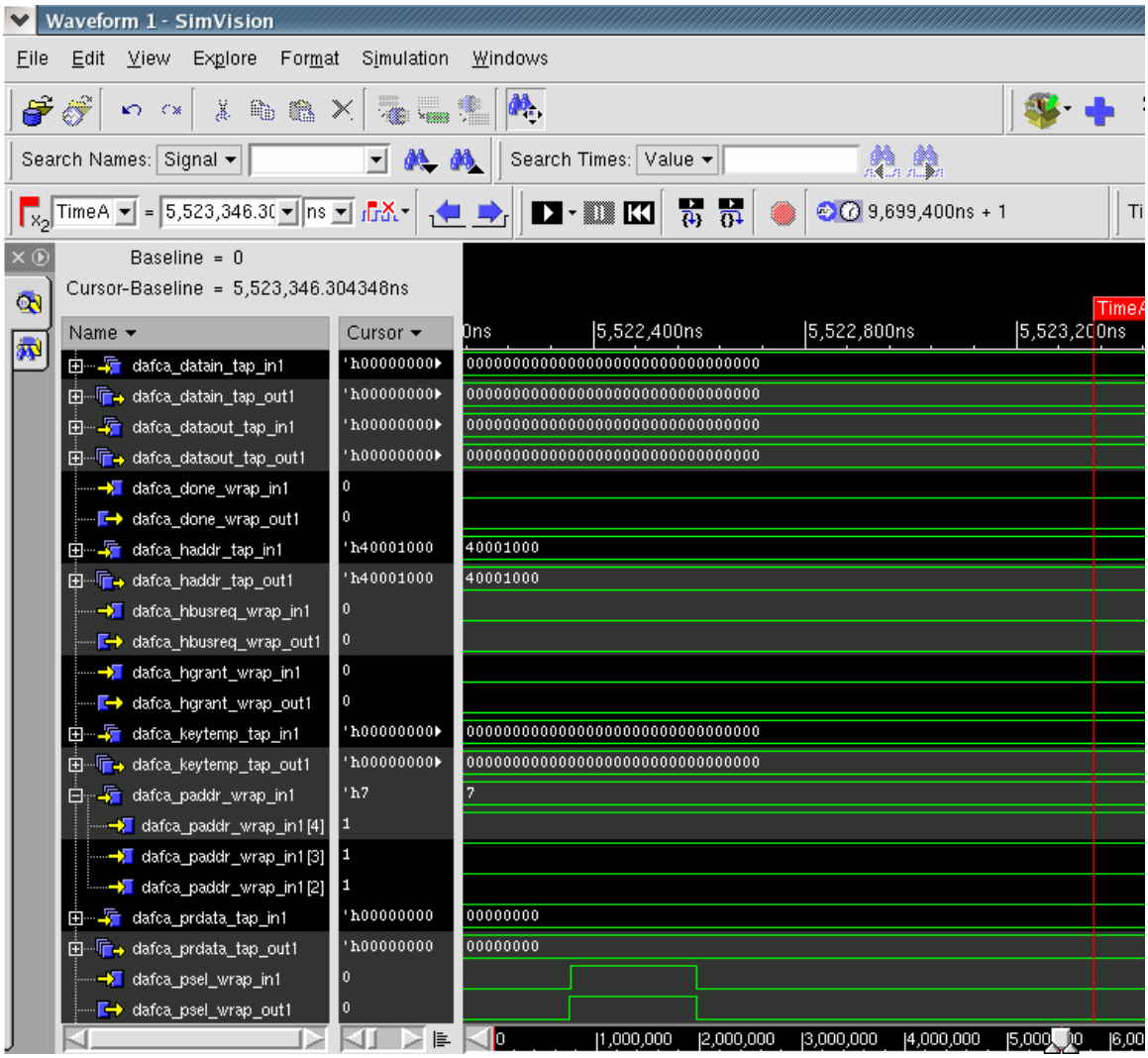


Figure 6.2 SoC Post-Layout Simulation after ‘paddr’ Changes

Figure 6.3 shows the changes made in the wrapper using the Personality Editor and Figure 6.4 shows the waveform for the SoC after this change. As seen in Figure 6.4, ‘dafca_hgrant_wrap_in1’ is stuck at ‘0’. This is the signal that is at the output of the wrapper XRW_Leon1 and goes all the way into the hierarchy to the AES module from the top level. As expected ‘datain’ and ‘dataout’ pins show no data values.

Forcing the ‘done’ signal from the AES module to ‘0’-

Once the AES is done with the decryption, it flags the ‘done’ signal to a high value indicating the same. With the help of the Personality Editor, I stuck the ‘done’ signal to ‘0’ and Figure 6.5 indicates this change. The effect of this personality can be seen in the waveform in Figure 6.6. As seen in Figure 6.6 as soon as the ‘dafca_done_wrap_in1’ becomes ‘0’, the ‘dataout’ signal changes to all zeroes.

Injecting the Counter Personality into the ‘paddr’ signals-

The Personality Editor can also be used to inject a Counter behavior to the wrappers. The Counters to choose from include the Gray counter, Carry-Look Ahead counter and the Ripple counter. I have inserted the 2-bit ripple counter personality into paddr[2] and paddr[3] and the waveform reflecting this change can be seen in Figure 6.7.

Programming the ‘assert always’ assertion for the ‘start’ signal-

The Personality Editor provides a library of 32 built-in assertions to load into the wrappers. Some of these assertions are asset_always,assert_never, assert_change,assert_decrement,assert_handshake etc. In addition to these built-in assertions the designer also has the option of writing his own assertions using the Python programming language. The assertions can be accessed from the Tools -> Insert Module menu option from the Personality Editor. They are an excellent way of verifying the functionality of critical signals in the SoC.

Using the Personality Editor, I programmed the ‘assert always’ assertion for the ‘start’ signal of XRW_Leon1. Once the key is loaded into AES, the ‘start’ signal becomes high for a short duration to initiate the AES to start the decryption. Figure 6.8 shows the assertion being fired as indicated by the ‘status’ bit on XRW_Leon1. The assertion is fired because the ‘start’ signal is not stuck at ‘1’ but toggles between the ‘0’ and ‘1’ values.

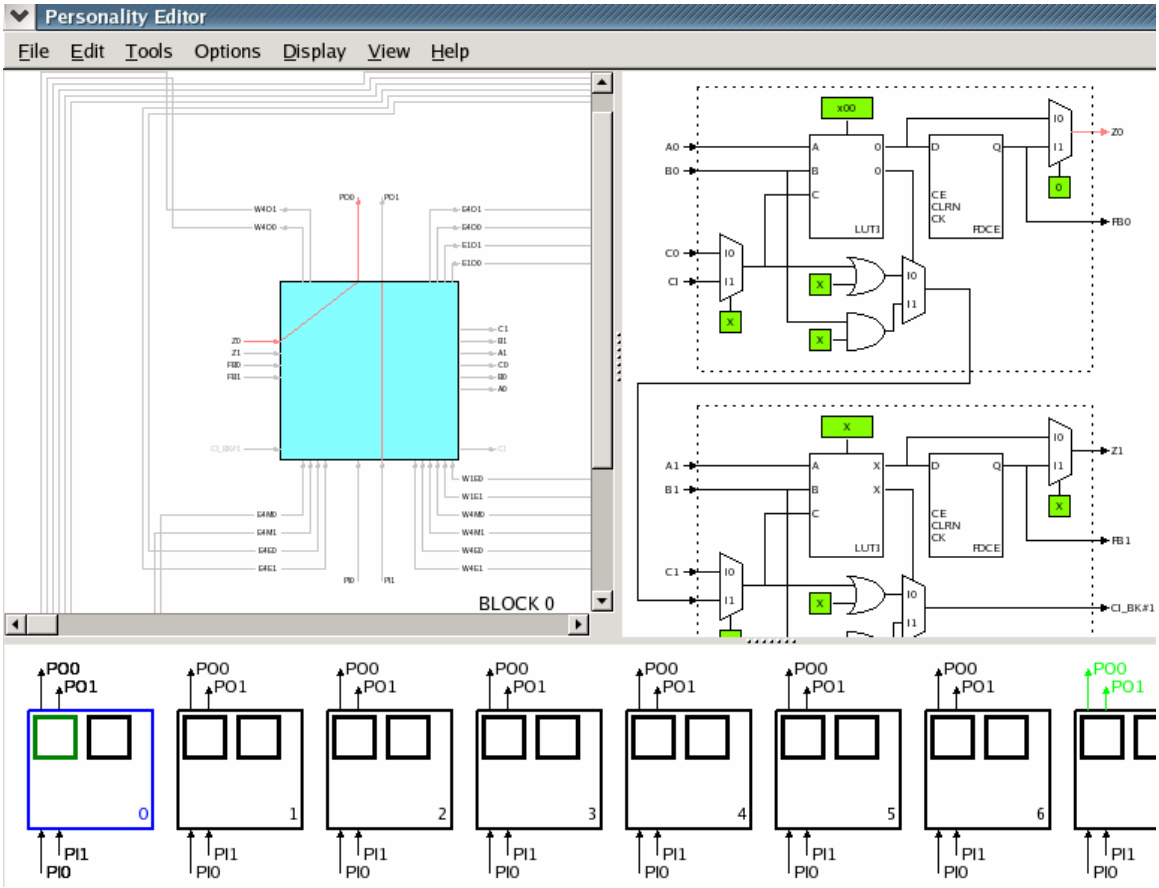


Figure 6.3 Personality Editor Window Depicting ‘hgrant’ Modification

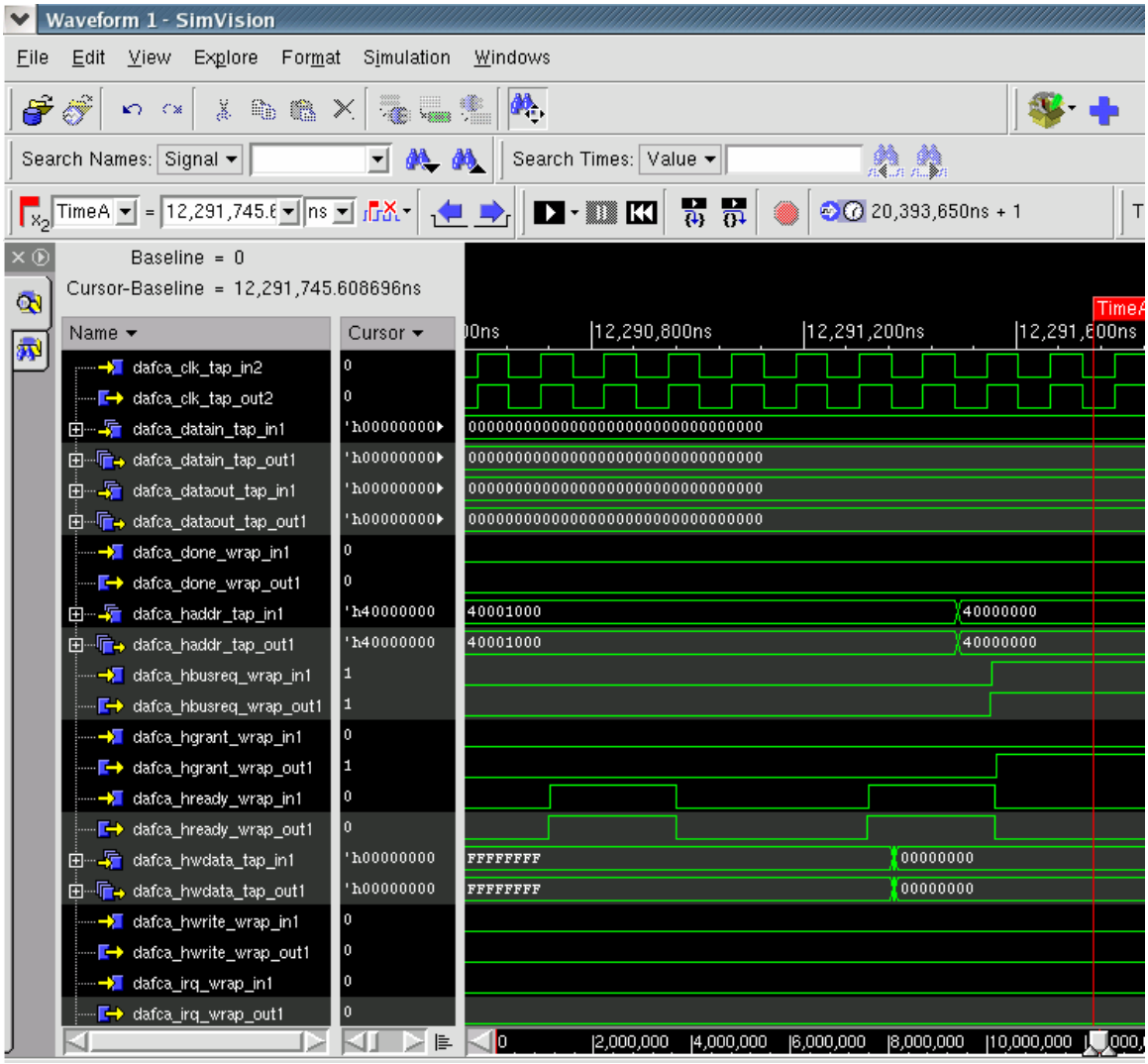


Figure 6.4 SoC Post-Layout Simulation after ‘hgrant’ Changes

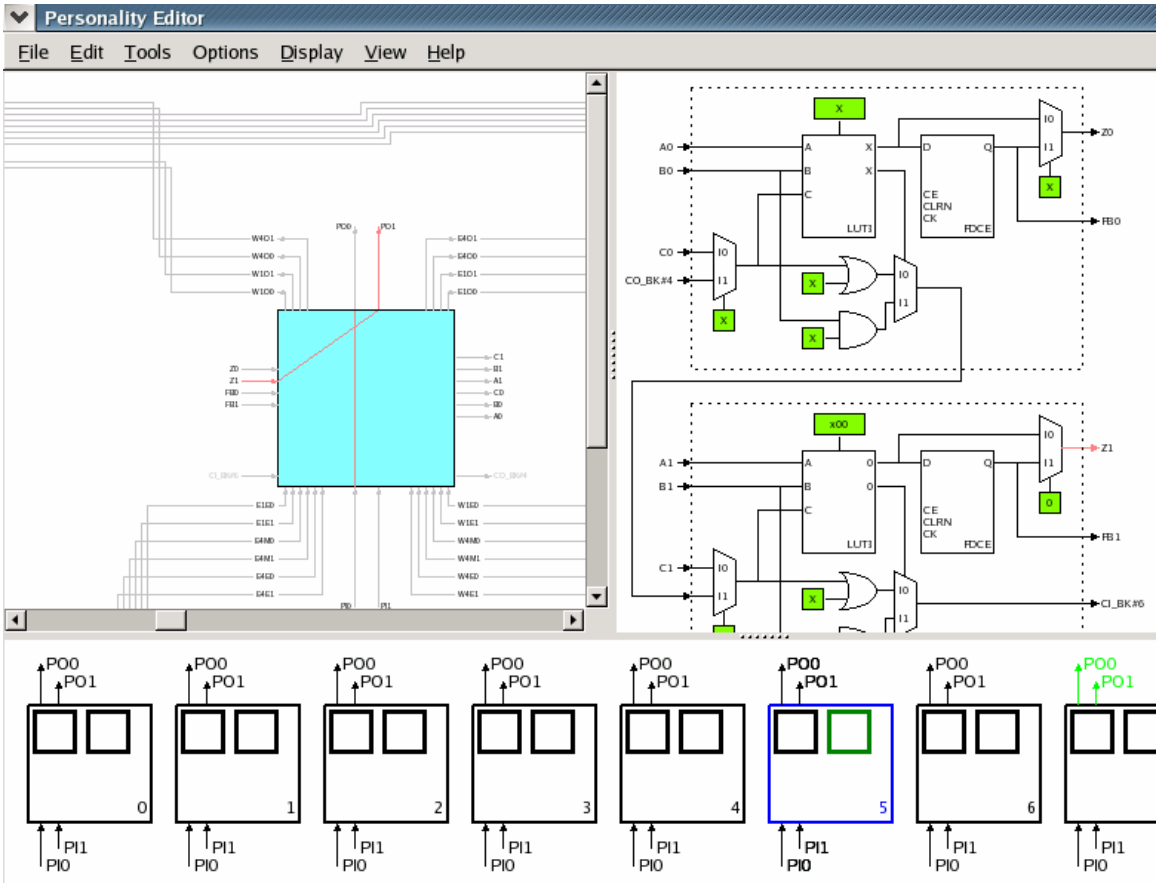


Figure 6.5 Personality Editor Window Depicting ‘done’ Modification

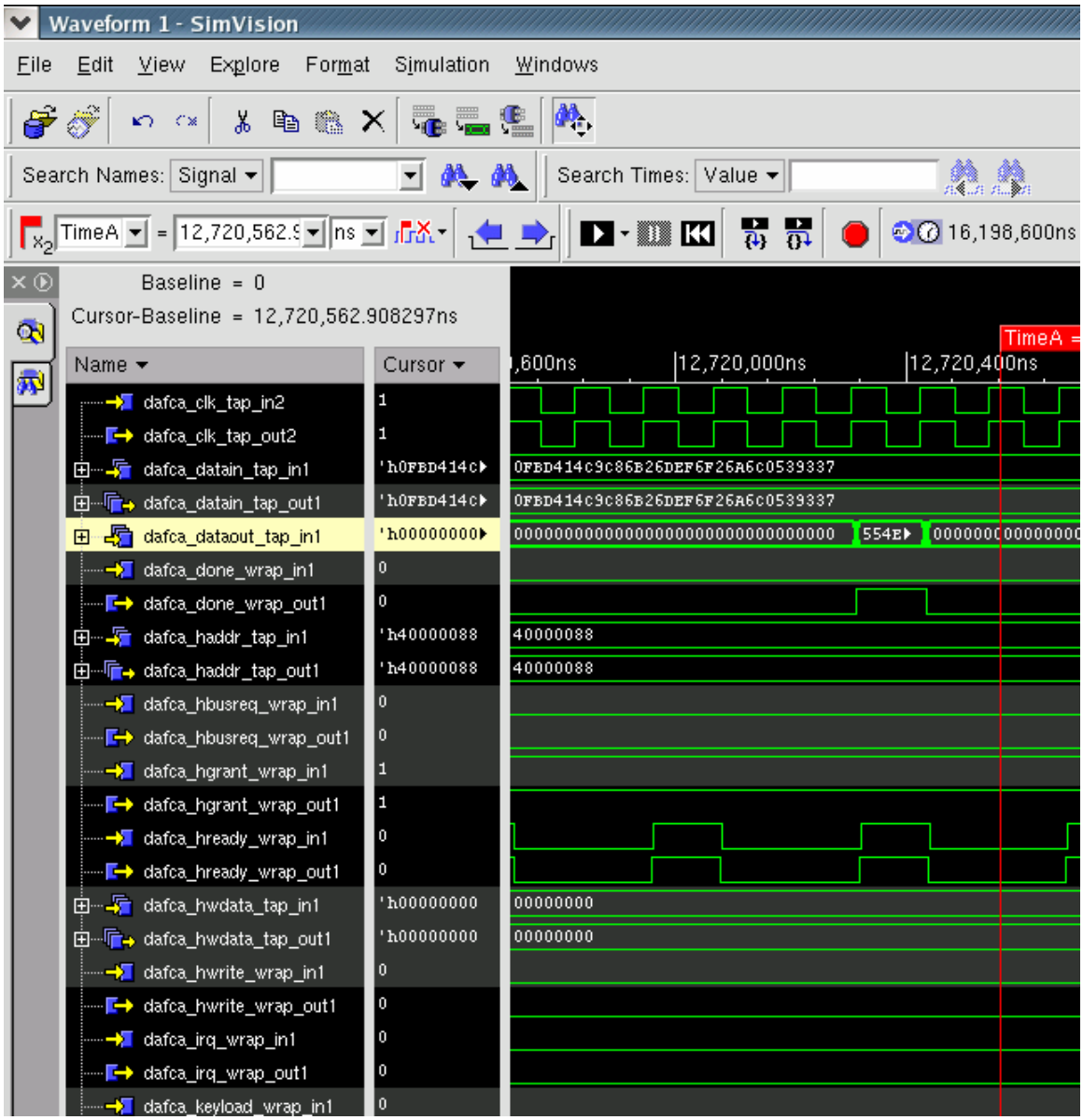


Figure 6.6 SoC Post-Layout Simulation after ‘done’ Changes

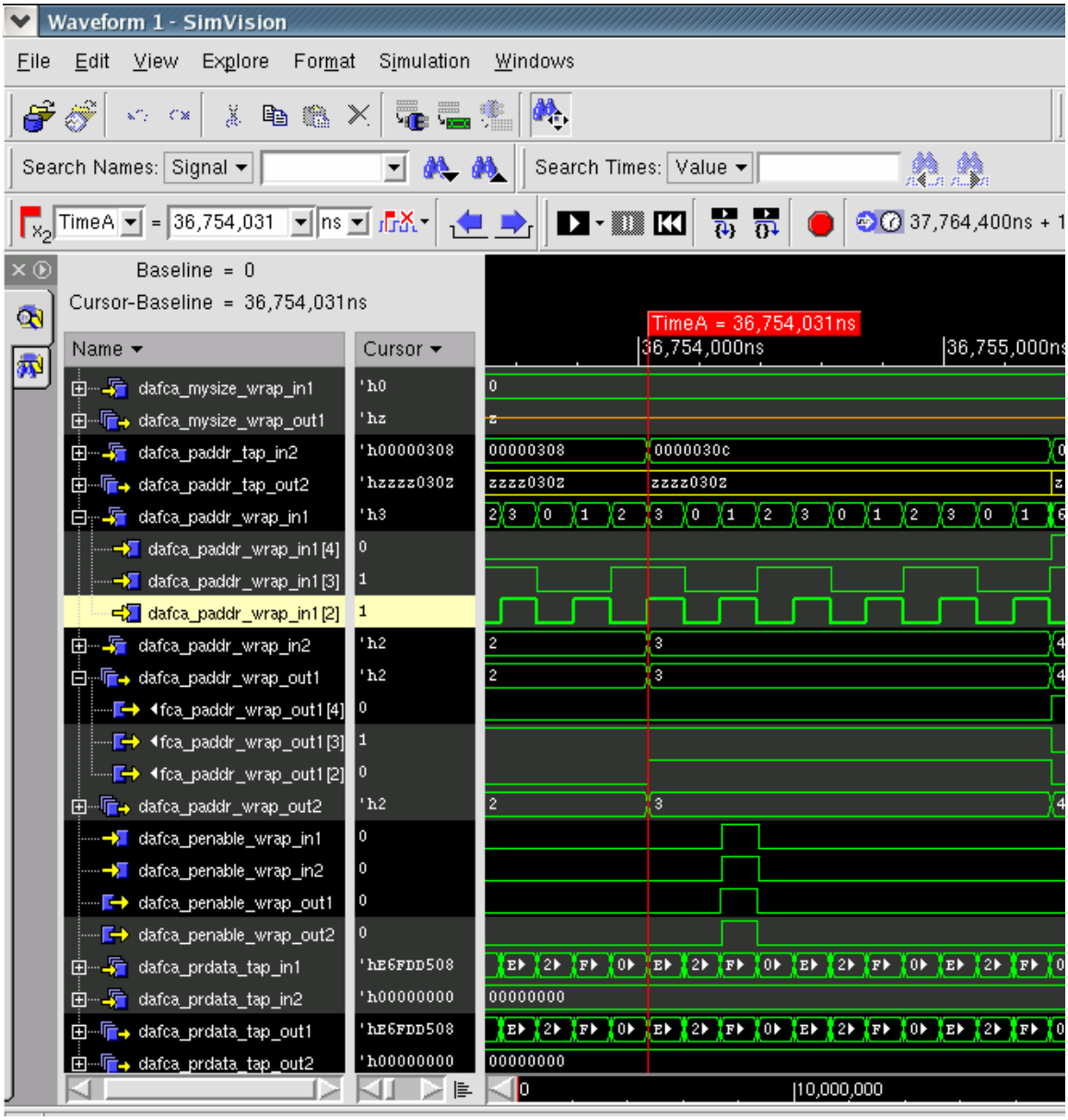


Figure 6.7 Post-Layout Waveform after Injection of Counter Personality to paddr[2] and paddr[3]

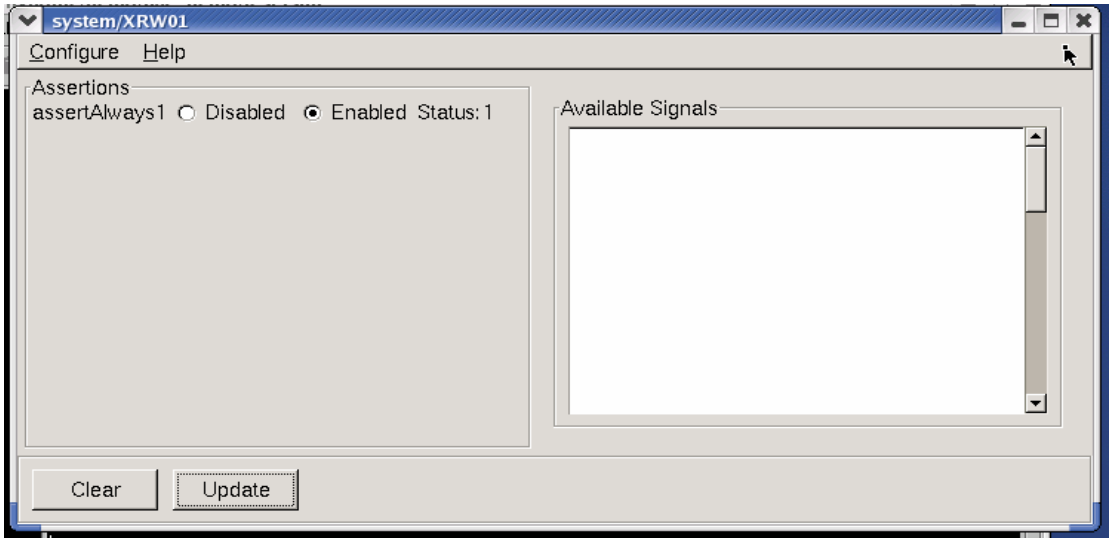


Figure 6.8 ‘assert Always’ Assertion being Fired

6.2 Discussion

The extra flexibility and debugging ability comes with the price of performance and area. The amount of reconfigurable logic introduced in a SoC design is always about the tradeoff between the flexibility and the area/delay of the design.

6.2.1 Area Comparison

The un-instrumented design has 812K transistors and the design core fit into a 2mm x 2mm area. In this instrumentation, 672 signals have been tapped and 77 signals have been wrapped and the tracer has a 2048x64 bit memory. The total transistor count of the final design is 3135K; the whole design area takes about 4.8mm x 4.8mm. The transistor count comparison for different modules is listed in Table 6.1.

6.2.2 Timing Overhead

Wrapping a signal introduces an extra MUX delay. Although tapping a signal does not explicitly introduce any MUX delay in the signal path, the extra load will slow down the transition time and the extra logic will introduce extra wiring delay as well. In the physical layout an attempt has been made to separate the original design and the embedded DAFCA blocks to minimize the impact of the design timing due to the extra wiring. The impact due to the extra load was also reduced since First Encounter performed transistor resizing during timing in-place-optimization. Furthermore, during the synthesis step a tighter timing constraint was used to leave some margin for the extra logic in the ReDI blocks. As a result, when the circuit is running in mission mode (by-passing the DAFCA logic), there is no degradation in design performance.

6.2.3 General Discussion

From the results above, it can be seen that the insertion of reconfigurable fabric into the chip greatly enhanced its observability and controllability and made error location and debugging easier. It gave the designer the internal access to the chip. Assertions, checkers and logic fixes could be performed on signals and the internal state of the circuit could be recorded. In addition to programming the wrappers with assertions and counters, the Personality Editor also provides the option of inserting arithmetic modules for implementing functions like add, equal, greater_than, less_than etc. to the wrappers. This adds to the versatility of the SoC.

It is true that about 60% of this design is embedded test logic, which might not seem very attractive in this particular case. But it should be noted that the final

Table 6.1 Transistor Count Comparison

	Original Baseline	With Test Logic
LEON CPU	408 K	467 K
AES	356 K	464 K
Cache	140 K	160 K
Register File	188 K	195 K
RAM Total	328 K	355 K
Wrapper 1		100 K
Wrapper 2		137 K
CMUX		74 K
rMatrix		280 K
rMonitor		603 K
Tracer Memory		819 K
SoC Total	812 K	3135 K

design has only about 3 million transistors, including 1.4 million transistors that are used for on-chip debug module (rMonitor and tracer memory). In a multi-million gate design, the size of the on chip debug module will remain the same while only the size of the wrappers and MUXs will be scaled. In general, this approach will be more suitable for a large SoC design where the extra cost can be justified. But an important consideration will always be to make an intelligent decision as to where to insert the ReDI fabric.

Chapter 7 Future Work and Conclusion

7.1 Future Work

As mentioned in the previous sections, I have inserted an rMatrix in our SoC to wrap the I/O ports of the dummy IP block. But due to unavailability of a Personality Editor for the rMatrix at this point, no tests could be performed on it. So injecting personalities into the rMatrix and testing it with the Post Silicon tool needs to be done.

Also, the Python programming language can be used to program assertions and personalities to be used in the Personality Editor in addition to the assertions already available.

The chip has been submitted to MOSIS for fabrication and is expected back on May 31, 2006. So a testing board needs to be designed before that time. Also, the chip needs to be tested in the same way as the post-layout netlist.

7.2 Conclusion

- Inserted the rMatrix to the original instrumented design.
- Configured the design for the 180-nm IBM 7RF process
- Synthesized, Placed and Routed the design using the 180-nm IBM 7RF process and submitted to IBM via MOSIS for fabrication.
- Tested the design through post-layout simulation and verified its self repair capability using DAFCA post-silicon tools.
- Reconfigurable instrumentation can greatly help reduce testing time and costs by making fault location and fixing easier.
- The approach is highly cost effective for multi million gate designs as product delays and re-spins may be avoided.

References

[1] Online Wikipedia. Available: http://en.wikipedia.org/wiki/System-on-a-chip

[2] International Technology Roadmap for Semiconductors, [Online].Available: http://public.itrs.net/

[3] Smith, M. J. S., "Application-Specific Integrated Circuits", Addison-Wesley, Boston, MA, 1997

[4] Miczo, Alexander, "Digital Logic Testing and Simulation", John Wiley & Sons, Inc. 2003

[5] Plusquellic, J. CMPE 646 Class Note, [Online]. Available: http://www.csee.umbc.edu/~plusquel/vlsi_test/

[6] DAFCA Inc., "In-Silicon Solution for Silicon Debug", DAFCA whitepaper.

[7] Lin, J. Challenges for SoC Design in Very Deep Submicron Technologies, [Online].Available: http://www.ece.uci.edu/codes+isss/Invited/JamesLin.pdf

[8] Fujiwara, Hideo,"Logic Testing and Design for Testability", MIT Press, 1985.

[9] Miczo, Alexander, "Digital Logic Testing and Simulation", John Wiley & Sons, Inc. 2003

[10] DAFCA Inc., "DAFCA ReDI HW Data book"

[11] Jiri Gaisler, Gaisler Research. The LEON-2 Processor: User's Manual, [Online]. Available: http://www.gaisler.com/.

[12] AMBA Specifications Rev 2.0, ARM IHI 0011A, ARM Limited, (1999), [Online]. Available: http://www.arm.com

[13] Jiang, W.," Enhancing System-on-Chip Verification using Embedded Test Structures", M.S. Thesis, University of Tennessee, December 2005.

[14] Srivastava, R.,"Development of an Open Core System-on-Chip Platform", M.S.Thesis, University of Tennessee, August 2004.

[15] DAFCA Inc., "DAFCA pre-silicon Reference Guide"

[16] DAFCA Inc., "ClearBlue Debug Environment User's Guide"

[17] Ecole Polytechnique notes, [Online]. Available: http://www.grm.polymtl.ca/~mahoney/dcrmo/dcrmo_2.pdf

[18] IBM Microelectronics Division, “CMOS7RF Design Manual”

Vita

Tushti Marwah was born in Delhi, India on August 29th, 1981. She received her Bachelor of Technology degree in Electronics and Communication from Guru Gobind Singh Indraprastha University in 2003. She attended The University of Tennessee, Knoxville in 2004 and received her Master of Science degree in Electrical Engineering in August 2006 under the guidance of Dr. Don Bouldin. Her research interests include Digital VLSI design and debug and Embedded Systems.