8-2004

An Open Core System-on-chip Platform

Rishi R. Srivastava
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Rishi R. Srivastava entitled "An Open Core System-on-chip Platform." 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 requirements 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:

Gregory D. Peterson, Chandra Tan

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

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

To the Graduate Council:

I am submitting herewith a thesis written by Rishi R. Srivastava entitled “An Open Core System-on-chip Platform”. 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 requirements 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:

Gregory D. Peterson

Chandra Tan

Accepted for the Council:

Anne Mayhew

Vice Chancellor and

Dean of Graduate Studies

(Original signatures are on file with official student records)

An Open Core System-on-chip Platform

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Rishi R. Srivastava

August 2004

Acknowledgements

First of all, I would like to thank my academic and thesis advisor Dr. Donald W. Bouldin for his guidance and support throughout the course of my education. It was a rewarding experience especially when we worked together for the submission of Research paper related to my thesis. I would like to thank him for helping me in pursuing this research and providing me with every facility available through the Microelectronic Systems Research Lab of the University of Tennessee. I am also very grateful to Dr. Gregory D. Peterson and Dr. Chandra Tan for their interest in this work and for serving on the thesis committee. Dr. Chandra Tan deserves a special thanks for his encouragement and guidance at particularly difficult stages of my work.

I am thankful to all those who have supported me and discussed every new idea that encountered during my thesis period. I would like to thank all of the students in the Spring 2003 class of ECE 652 for helping build a library of Intellectual Property (IP) blocks used in this thesis.

I am particularly grateful to my boss, Ms. Sydney Post and the Business Office for providing financial support throughout my education. Last but not the least I would like to thank my parents, Shri. G. K. Srivastava and Smt. Ragini Srivastava and my brothers Ravi and Rudra for their unrelenting support and encouragement.

Abstract

The design cycle required to produce a System-on-Chip can be reduced by providing pre-designed built-in features and functions such as configurable I/O, power and ground grids, block RAMs, timing generators and other embedded intellectual property (IP) blocks. A basic combination of such built-in features is known as a **platform**.

The major objective of this thesis was to design and implement one such System-on-Chip platform using open IP cores targeting the TSMC-0.18 CMOS process.

The integrated System-on-Chip platform, which contains approximately four million transistors, was synthesized using Synopsys - *Design Compiler* and placed and routed using Cadence - *First Encounter, Silicon Ensemble*. Design verification was done at the pre-synthesis, post-synthesis and post-layout levels using Mentor Graphics - *ModelSim*. Final layout was imported into Cadence - *Virtuoso* to perform design rule check.

A tutorial was written to enable others to create derivative designs of this platform quickly.

Table of Contents

List of Tables

List of Figures

Chapter 1: Overview

1.1 Introduction

Moore's Law [1] predicted that the number of transistors on a chip will double every eighteen months and for more than three decades now the integrated circuit design industry has followed Moore's law. Various studies on similar topics also predicted a 20-fold increase in power and capabilities of integrated circuits over a period of a decade [2].

Conventionally, integrated circuit design involved circuits with medium complexity, around 200-500K gates, operating at 50-100 MHz speed, were designed using 0.35-micron silicon process technology and were made up of mostly core logic along with some hard macros like SRAMs. These designs would have a design cycle of 12-18 months [3].

Whereas modern designs involve circuits with superior complexity, around 10-25 million gates and are designed using 0.18 - 0.13-micron silicon process technology, and are able to sustain a clock speed in excess of 1 GHz. This explosive growth in gate count and speed as well as consumer requirements for bleeding edge technologies like modern telecommunication equipments, consumer goods like PDAs, 3rd generation mobile devices, has pressured the design technology community to harness its potential quickly. As a result we

have integrated circuits with much more complex capabilities. Today integrated circuits are not only faster and larger, they also include traditional microprocessor cores, Intellectual Property (IP) cores, and memory cores -- in other words, a System-on-Chip (SoC).

According to a report [4] on market growth for System-on-Chip, the volume will increase at a stupendous rate of 30-35% annually with many major companies investing two-thirds of their research and development in the System-on-Chip arena.

Emergence of System-on-Chip technology has brought with it a whole spectrum of opportunities and challenges. Opportunities are in the form of reduced cycle time, time-to-market considerations, bigger spectrum of customers, and superior performance. Whereas the challenges include deep sub micron design complexities, verification and integration.

Time-to-market may be optimized by reducing the design cycle and by reducing the manufacturing cycle. The design cycle can be reduced by providing pre-designed built-in features and functions such as configurable I/O, power and ground grids, block RAMs, timing generators and other embedded IPs. A basic combination of such built-in features is known as a **platform**. The platform used to implement a System-on-Chip greatly impacts all of the issues and is the

fundamental decision the hardware designers must make at the start of each new project.

Design reuse in the form of previously verified and used IP cores can greatly reduce time-to-market and increase quality for System-on-Chip designs. According to a report [5], by 2010 the percentage of IP contained in a System-on-Chip application is predicted to grow to 95%.

Now, million-gate integrated circuits are increasingly being designed as System-on-Chip platforms since platform design mitigates the risks involved with integrating a CPU core and other virtual components by a fixed deadline. Using this approach, designers can overcome uncertainties about the quality of the components and their interaction and can produce derivative designs rapidly.

The development of one such System-on-Chip platform is described in this thesis. In the process of implementing this project, emphasis was to learn not only to reuse existing cores but also the requirements to create high quality cores for reuse. This System-on-Chip platform, which uses only open cores that can be obtained by anyone at no charge, served as an “industrial strength” design for me to learn about optimizations at the logic and physical levels. Thus, synthesis and place/route tools were used to explore the power-delay-area solution space of a million-gate design. Having internal visibility of the

components at both the source code level and at the physical layout level greatly facilitates understanding of System-on-Chip development issues. The System-on-Chip platform is being placed in the public domain so that others may contribute to its enhancement.

1.2 Project Goals and Core Selection

The objective of this thesis was to design and implement a baseline System-on-Chip platform targeting the TSMC-0.18 CMOS process. To enhance the understanding of System-on-Chip issues, I have selected only open soft cores that could be obtained for free (e.g. AES) or have been generated internally at the University of Tennessee (e.g. FIR, FFT). For the CPU, I have selected the LEON-2 processor [6]. This processor is specifically designed for embedded applications. As shown in figure 1.2.1, LEON-2 core provides a direct memory-interfaced PROM, memory-mapped I/O, SRAM and SDRAM with variable memory width of 8, 16 or 32 bits. The LEON-2 processor can also include various other features such as two UARTs, an interrupt controller, a memory controller, and an interface for a coprocessor or floating-point unit.

A flexible configuration scheme makes it straightforward to add new cores as masters or slaves depending upon their functionality. The LEON-2 processor has implemented a Harvard Architecture for cache with separate data and instruction

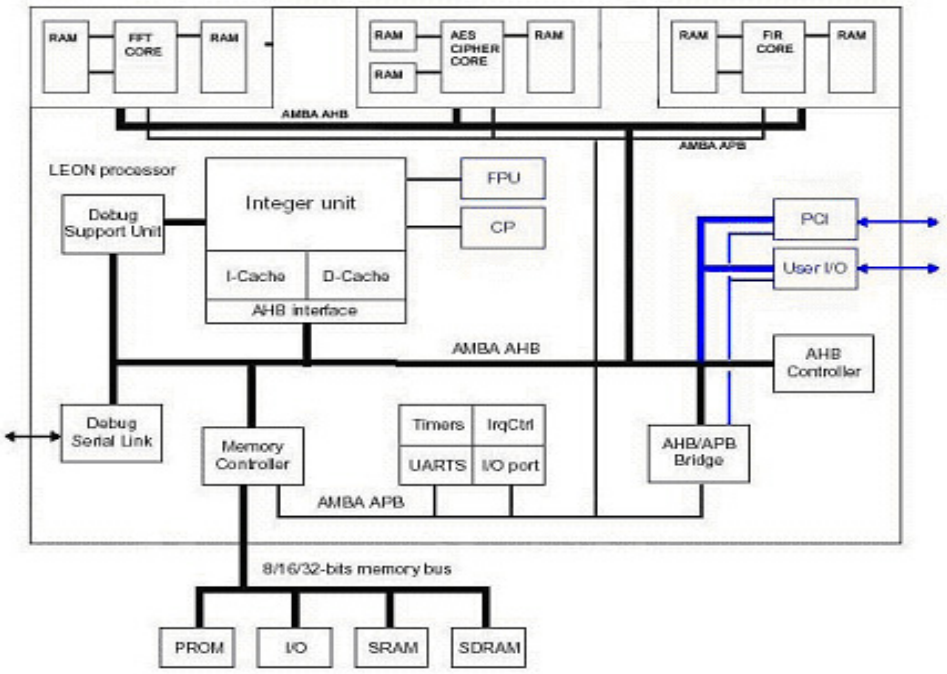


Figure 1.2.1 Block diagram of the open System-on-Chip platform

cache RAMs which can be generated in 1-4 sets each of 1-64Kb depending on the functionality desired.

The compiler for the LEON-2 is LECCS (LEON/Erc32 GNU Cross-Compiler System) [7] which is compatible with Sun Solaris / Linux / Windows operating systems. LECCS supports ordinary sequential C/C++ programming or multitasking using the RTEMS (Real Time Embedded Micro-controller Systems) kernel.

Chapter 2: Background

2.1 Design Process

As said earlier, it is not uncommon to find chip design that packs 20-50 million transistors on a single die. In a few years we hope to be able to pack hundreds of million transistors on a single die. That is assuming the tools are in place that will be able to manage the design of that complexity. The answer to this question lies in how industry develops the design process and methodology. Three major methods used to design integrated circuits are:

- Full-Custom design
- Standard-Cell design
- Gate-array design.

Full-Custom design is the lowest level, requiring the designers to specify the exact location of every wire and transistor. Standard-Cell designs are a bit simpler; the designer is given a library of fairly simple logic elements and allowed to assemble them in any way.

The gate-array approach is not only the simplest but also provides an attractive alternative that offers shorter design cycle, quicker response on iterations and modifications, and lower non-recurring engineering (NRE) costs. A gate-array solution is frequently completed many months ahead of a full-custom or

standard-cell equivalent. I will focus on the gate-array design process because it is the simplest and because it is the method generally used to design the kind of chip discussed in this thesis.

Since designing a gate-array is simple, hardware description languages (HDL) and synthesis tools are very popular among gate-array designers. A hardware description language provides an easy way to specify the behavior of the chip and provides an environment for simulating the behavioral model. The synthesis tools can turn this model into a gate-level description, and often provide ways to simulate that description as well, thus completing most of the design work, although it is easier said than done.

As a design gets larger, timing closure at the chip level becomes much more complex. And as process geometries continue to shrink, signal integrity effects such as noise, increased interconnect crosstalk, lower power voltage, and other effects must be considered. The bottom line is that the design and implementation of sub-0.18 micron chips present significant challenges. If there is no change either in the methodology or the tools used today the cost and resource requirements that would be needed to design and implement deep sub-micron chips will be somewhat proportional to the number of transistors [8].

Hierarchical Design Process

An approach that has been taken while implementing the open core System-on-Chip platform is hierarchical design. It is a common approach to solving complex designs problem. The approach is to break down the design into manageable pieces and solve one piece at a time [8]. If the pieces are small enough, the problem can be manageable. However, success depends on bringing all of the pieces together again to provide an answer to the original problem. This approach has been applied to complex engineering projects and is now finding its way in the System-on-Chip design process. Hierarchical chip design can be roughly separated into three broad processes:

- Process of breaking the overall design into blocks that will be implemented individually. Planning in this process is critical, as project must yield a final design that meets the project goals for timing and other requirements.
- Process of implementing the detailed design of the individual blocks.
- Process of connecting all of the blocks in the design to result in the final chip.

We will be discussing more about these issues in the chapter discussing implementation of the System-on-Chip platform. There we will get a better understanding about how timing requirements are so essential at the block level and can have serious consequences on integration of the core.

2.2 Design Methodology

In the ASIC industry, switching from designs that were based on transistors to the designs that were based on gates proved to be a great boon for the industry [9]. It induced a huge growth in productivity and helped make concepts such as gate-arrays a reality. It provided the groundwork for new industries, restructuring the existing engineering organizations providing broader boundaries for the relationship between the designer and design by introducing a new level of abstraction.

A general pattern followed by most of the ASIC industry is that the silicon process technology changes which is then followed by making changes to the design technology. These changes are then adopted by the design methodology, which then implements these changes in the form of new processes. These processes further result in an increase in productivity [10]. However, over a period of time now, there has been major progress in silicon manufacturing technology leading to a situation where design technology is lagging far behind. Consequently, industries now need a fundamental reorganization so that designs are done not only faster but also in a different and more efficient way. Therefore, traditional design processes are now being replaced by the SoC designs.

We are now entering the era of block-based design (BBD); heading towards virtual component (VC) based System-on-Chip design, which is driven by our ability to harness reusable virtual components (VC), a form of IPs. Today design methods can be divided into four main segments [11]:

- Area Driven Design (ADD)
- Timing-driven design (TDD)
- Block-based design (BDD)
- Platform-based Design (PBD)

Timing-Driven Design (TDD)

TDD is the most efficient methodology used for designing a moderately sized ASIC, consisting mostly of new logic on deep sub-micron processes, without significant utilization of the hierarchical design process. When a team is working on some design that is required to meet certain performance constraints with respect to its speed and power consumption, they follow TDD. With the availability of modern tools to make delay calculations and timing analysis, accuracy has reached a new level and is able to provide an unbiased idea about the design capabilities. One major shortcoming on the part of TDD is that, at higher gate counts, usually in excess of the 150-K mark, it begins to fail as the complexity increases.

Block-Based Design (BDD)

Designers are now in a position to reuse system level functions, and the complexity in the design of a chip is also steadily increasing. Now, with a new relationship between system, RTL and physical design, designers are making the change from the Timing Driven Design (TDD) to a Block Based Design (BBD) methodology. Ideally, BBD is behaviorally modeled at the system level, where hardware/software trade-offs as well as hardware/software co-verification using software simulation or hardware emulation is performed. The new design components are then partitioned and mapped onto specific functional RTL blocks, which are then designed to budgeted timing, power and area constraints. This is in contrast to the TDD approach, where timings are captured along synthesis-restricted boundaries. The combination of system level simulation of designs and RTL simulation of individual blocks minimizes the requirement for a unique testbench. Reusable blocks are poorly characterized, subject to modification and require re-verification. This effects the time-to-market equation.

Block based design generally employ a bus architecture, either processor determined or custom. BBD needs effective block level floor planning to estimate effective block size quickly. This helps in creating a viable budget for all blocks and their interconnection, which is essential to the convergence.

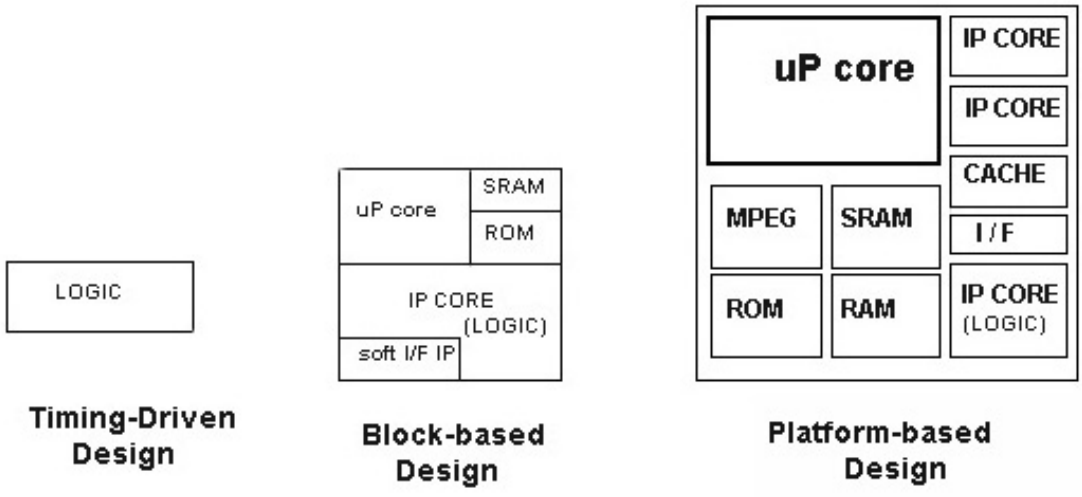


Figure 2.2.1 Design methodologies

Platform Based Design (PBD)

As shown in figure 2.2.1, PBD constitutes the next step in the evolution of design technologies. It attempts to comprise the cumulative capabilities of both TDD and BDD technologies. One quality that separates PBD from BBD is extensive planned design reuse and difference achieved in time-to-market for even the first products. It has also expanded the opportunities and speed of delivering derivative products.

Like BBD, PBD too is a hierarchical design methodology that starts at the system level. Using predictable, preverified reusable IP blocks that have standardized interfaces increases productivity and greatly effects time-to-market equations [11]. PBD methodology separates design into two areas of focus:

- Block Authoring
- System-Chip Integration.

Block authoring primarily uses a methodology suited to the block type (TDD, ADD), but the block is created so that it interfaces easily with multiple target designs. To be effective two new design concepts must be established: interface standardization and virtual system design.

In interface standardization many different design teams, both internal and external to the company can do block authoring, as long as they are all using the same interface specification and design methodology guidelines. Virtual system design answers the question related to power consumption and distribution, test options for different blocks, aspect ratio and clock distribution.

System integration focuses on designing and verifying the system architecture and the interface between the blocks. Contrary to its name, system integration starts with partitioning the system around the pre-existing block level functions taking into consideration performance analysis, hardware software design tradeoffs.

The basic idea behind the platform-based design approach is to avoid designing a chip from scratch. Some portion of the chip's architecture is predefined for a

specific type of application. Usually there is a processor, a real-time operating system (RTOS), peripheral intellectual property (IP) blocks, some memory and a bus structure. Depending on the platform type, users might customize by adding hardware IP, programming FPGA logic or writing embedded software.

2.3 Challenges in System-on-Chip Design

With the unprecedented level of integration in integrated circuit design, designers can pack a variety of functionalities on one chip. But to be able to take real advantage of such opportunities, System-on-Chip designers have to grapple with an exponential increase in design complexity. Also exploding transistor counts and skyrocketing clock rates coupled with changes in design methodologies have unleashed an entirely new set of design challenges.

The impact of exploding transistor counts on design methodologies has been profound. A few years ago the majority of silicon respins were due to simple functional design errors [12]. Furthermore designers could make simple assumptions to predict and compensate for the impact of physical effects such as signal integrity and crosstalk.

Today this is no longer the case. Designers can no longer manage those physical effects with simple models and assumptions regarding the design. As System-on-Chip designers venture into nanometer processes they are finding that an

increasing proportion of failures are a result of physical effects that are not reflected in the simple models used to represent transistors and wires. As a result traditional approaches to design no longer apply and new verification techniques have to be incorporated.

Today designers of ASICs 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 [13]. 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 [14]. Hence, not all of the transistors on these chips can be customized but instead must be ported from previous designs. These reusable cores or IP blocks include CPUs (like ARM, PowerPC and LEON-2), 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 than for a design that is not intended to be reused. Based on the experiences of software engineers, it is estimated that preparing a component for reuse will require about 50% additional effort [15]. Once this has been done, the designer who is reusing the component may naively think that his design time for that component will be reduced to zero. But alas, he must take care to understand fully how the component works and how it should be integrated with other components. Again from the experiences of software engineers, the second design generally requires about 30% of that required to produce the component originally. Thus, the reuse is not for free but does make a significant (70% reduction) impact on the next design.

Chapter 3: Component Background

3.1 AMBA Overview

Design reuse in a System-on-Chip is a critical feature and it can be successfully achieved through proper investment in standards. AMBA, which stands for an Advanced Microcontroller Bus Architecture, is an open standard [16], which defines an on-chip bus specification for interconnection and management of various functional blocks that are a part of System-on-Chip. Using the AMBA specification enhances the reusable platform based design methodology by defining a common standard for data transfer in a System-on-Chip module. AMBA has been widely adopted throughout the industry and, as a consequence, there is support for the development of AMBA bus-based systems from a growing number of companies. The AMBA specification has been derived to satisfy four key requirements:

• To facilitate the right-first-time development of embedded microcontroller products with one or more CPUs or bus masters.

• To be technology-independent and ensure that highly reusable peripheral can be migrated across a diverse range of IC processes.

• To encourage modular system design to improve processor independence, providing a development road map for advanced cached CPU cores and the development of peripheral libraries.

• To minimize the silicon infrastructure required to support efficient on-chip and off-chip communication for both operation and manufacturing test.

Three distinct buses are defined within the AMBA specification:

Advanced High-performance Bus (AHB)

The AMBA AHB is for high-performance, high clock frequency system modules. The AHB acts as the high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macrocell functions.

Advanced System Bus (ASB)

The AMBA ASB is for high-performance system modules. AMBA ASB is an alternative system bus suitable for use where the high-performance features of AHB are not required.

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 either version of the system bus. The AMBA APB should be used to interface to any peripherals which are low bandwidth and do not require the high performance of a pipelined bus interface.

System-on-Chip platform: Bus Architecture

For the open core System-on-Chip platform discussed in this thesis there were various options to attach IP blocks as AHB bus-masters or AHB bus-slaves. Although attaching them as AHB slaves would be easier and less complicated the overall architecture would have become restrictive and also very much dependent upon the availability of LEON-2 processor to carry out a process. On the other hand, implementing the IP block as AHB bus-master would be more complex but at the same time provide appropriate flexibility for future modification or performance improvement tasks.

The AMBA AHB bus protocol is designed to be used with a central multiplexer interconnection scheme. Using this scheme, all bus masters drive out the address and control signals indicating the transfer they wish to perform and the arbiter determines which master has its address and control signals routed to all of the slaves. A central decoder is also required to control the read data and response signal multiplexer, which selects the appropriate signals from the slave that is involved in the transfer.

As shown in figure 3.1.1 an AHB bus master has the most complex bus interface in an AMBA system. Typically an AMBA system designer would use pre-designed bus masters and therefore would not need to be concerned with the detail of the bus master interface. But for our project we had to develop a model using one of

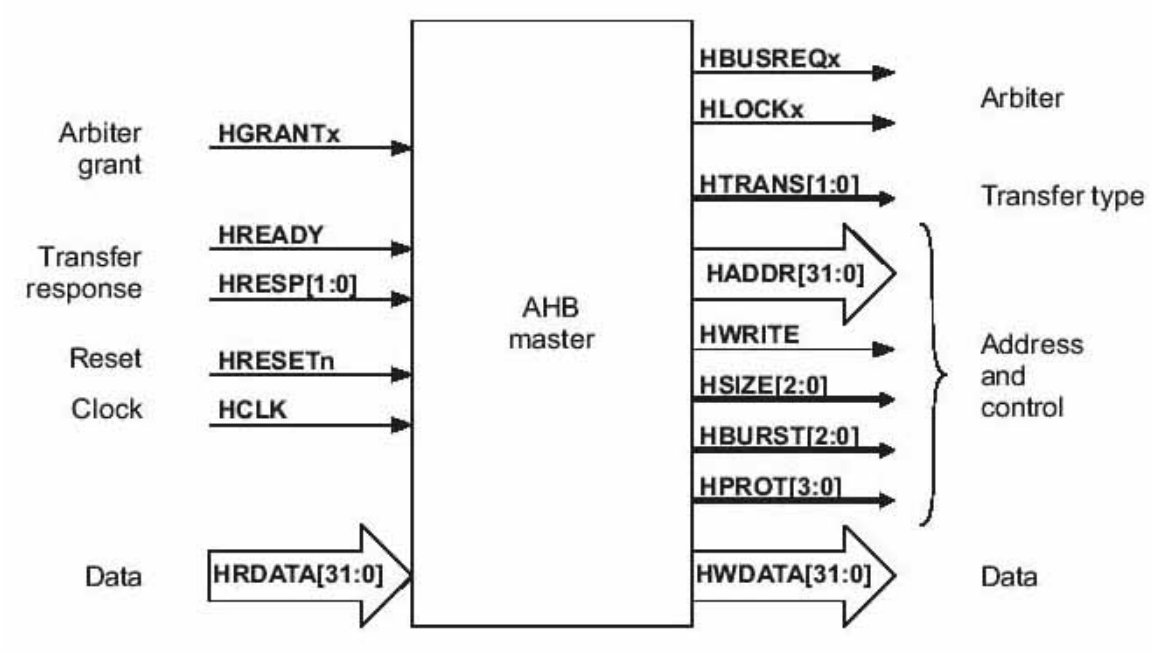


Figure 3.1.1 AHB bus master interface diagram

the IP blocks as an example, which we refer to as AMBA wrapper, could then be reused for other IP blocks with minor modifications.

Before an AMBA AHB transfer can commence, the bus master must be granted access to the bus. The master asserting a request signal to the arbiter starts this process. Then the arbiter indicates when the master will be granted use of the bus. A granted bus master starts an AMBA AHB transfer by driving the address and control signals. These signals provide information on the address, direction and width of the transfer, as well as an indication if the transfer forms part of a burst.

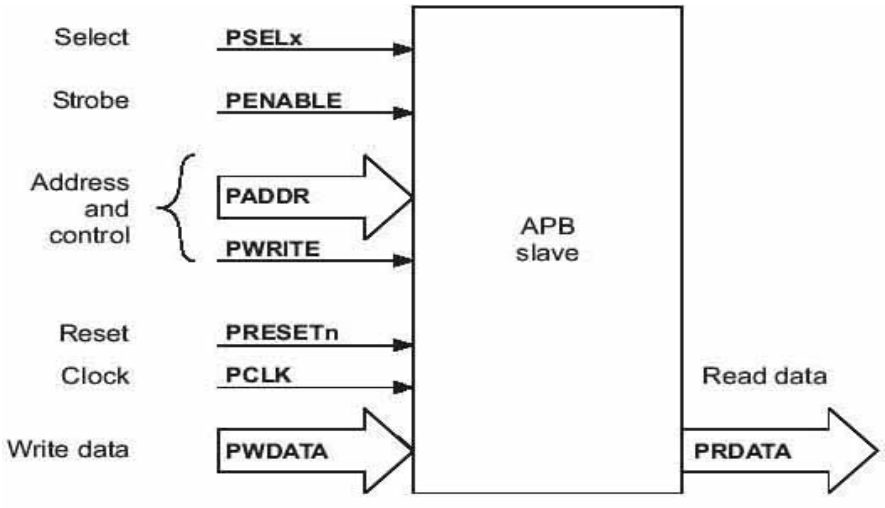


Figure 3.1.2 APB slave interface description

An IP block to be attached to the open core System-on-Chip platform can be attached as APB slaves since the LEON-2 processor itself is an APB master. This way all the control signals to individual IP blocks can be sent through the APB bus. Figure 3.1.2 describes the interface for a block acting as an APB slave.

3.2 LEON-2 Architecture

LEON-2 is a 32-bit processor conforming to the IEEE-1754 (SPARC V8) standard. The VHDL model of the processor, which is available free, and is highly flexible can be configured and made suitable for embedded applications and System-on-Chip designs [17]. Figure 3.2.1 describes the block diagram for LEON-2 processor.

For my System-on-Chip platform I used the latest version of the LEON-2

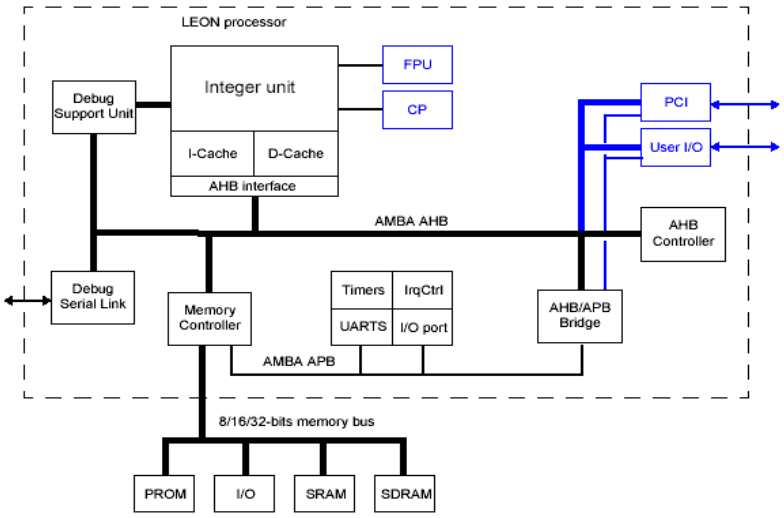


Figure 3.2.1 LEON-2 processor block diagram

processor (LEON2-1.0.12) available at the time. Updating to newer versions of LEON-2 is not a very difficult task provided there aren't many changes in the upcoming versions.

LEON-2 Architecture Overview

The LEON-2 processor is designed for embedded applications containing the following on-chip features:

- Separate instruction and data cache (Harvard Architecture)
- Hardware Multiplier and Divider
- Interrupt controller
- Debug Support Unit with trace buffer
- Two 24-bit timers
- Two UARTs

- 16-bit I/O port and a flexible memory controller.
- APB is used to access on-chip registers in the peripheral functions.
- AHB is used for high-speed data transfers.

The full AHB/APB standard is implemented and the AHB/APB bus controllers can be customized through the TARGET package and DEVICE.VHD, which is a configuration file. Additional (user-defined) AHB/APB peripherals should be added in the MCORE module. For the bus controller to recognize a new IP module, the following changes are needed in the DEVICE.VHD file. Figure 3.2.2 shows the addition of AES block in the list of APB slaves.

Important thing to note in the figure 3.2.2 is that the memory range allotted to AES is 0x800000300 H to 0x8000003FF H. When the LEON-2 processor accesses

```
constant apbslvcfg_tkconfig : apb_slv_config_vector(0 to APB_SLV_MAX-1) := (
--   first        last      index  enable     function           PADDR[9:0]
( "0000000000", "0000001000",  0,   true), -- memory controller, 0x00 - 0x08
( "0000001100", "0000010000",  1,   false), -- AHB status reg.,   0x0C - 0x10
( "0000010100", "0000011000",  2,   true), -- cache controller,  0x14 - 0x18
( "0000011100", "0000100000",  3,   false), -- write protection,  0x1C - 0x20
( "0000100100", "0000100100",  4,   true), -- config register,   0x24 - 0x24
( "0001000000", "0001101100",  5,   true), -- timers,            0x40 - 0x6C
( "0001110000", "0001111100",  6,   true), -- uart1,             0x70 - 0x7C
( "0010000000", "0010001100",  7,   true), -- uart2,             0x80 - 0x8C
( "0010010000", "0010011100",  8,   true), -- interrupt ctrl     0x90 - 0x9C
( "0010100000", "0010101100",  9,   true), -- I/O port           0xA0 - 0xAC
( "0010110000", "0010111100", 10,   false),-- 2nd interrupt ctrl 0xB0 - 0xBC
( "0011000000", "0011001100", 11,   false), -- DSU uart          0xC0 - 0xCC
( "0100000000", "0111111100", 12,   false), -- PCI configuration  0x100- 0x1FC
( "1000000000", "1011111100", 13,   false), -- PCI arbiter        0x200- 0x2FC
( "1100000000", "1111111111", 14,   true), -- AES Module         0x300- 0x3FF
   others => apb_slv_config_void);

constant apb_tkconfig  : apb_config_type := (table => apbslvcfg_tkconfig);
```

Figure 3.2.2 Configuring apb_slv_config_vector

any registers in this memory range then only the AES module is triggered for a response. Similarly we need to specify in AHB master's vector, number of bus masters attached. This notifying the bus-controller which in turn arbitrates which master has control of the bus. The number of bus masters will change depending upon the number of IP blocks added at a time. Priority can be assigned in the MCORE module. In our case following priority were assigned:

- LEON-2 Processor (0), AES block (1), FIR block (2)

Where higher number has higher priority.

LEON-2 itself uses AHB bus to connect the processor cache controllers to the memory controller and other (optional) high-speed units. In the default configuration, the processor is the only master on the bus, while two slaves are provided: the memory controller and the APB bridge. Figure 3.2.3 shows the default address allocation.

Address range	Size	Mapping	Module
0x00000000 - 0x1FFFFFFF 0x20000000 - 0x3FFFFFFF 0x40000000 - 0x7FFFFFFF	512 M 512 M 1 G	Prom Memory bus I/O SRAM and/or SDRAM	Memory controller
0x80000000 - 0x8FFFFFFF	256 M	On-chip registers	APB bridge
0x90000000 - 0x9FFFFFFF	256 M	Debug support unit	DSU

Figure 3.2.3 Default address allocation

From the above address space it is evident that we can read and write to APB devices in the range 0x80000000 – 0x8FFFFFFF and this is why we added the AES block in this range. The APB bridge is connected to the AHB as a slave and acts as the (only) master on the APB. Most on-chip peripherals are accessed through the APB.

3.3 RTEMS, LECCS

LECCS is an acronym for LEON/ERC-32 Cross Compiler System. Today almost all real-time embedded software systems are developed in a cross development environment using cross development tools. In a cross development environment, software development activities are typically performed on one computer system, the build host system (in this case LECCS), while the result of the development effort (produced by the cross tools) is software executable to be used on the target platform. Figure 3.3.1 explains shows the analogy between gcc and LECCS.

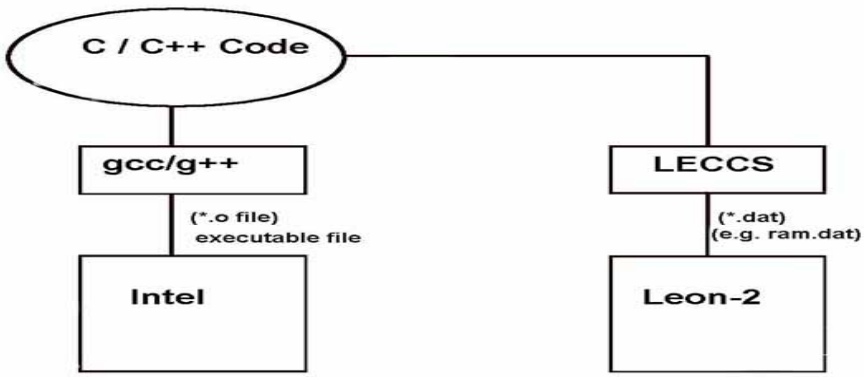


Figure 3.3.1 Analogy between gcc and LECCS compiler systems

The cross development toolset must allow the developer to customize the tools to address the target specific run-time issues. The toolset must have provisions for board dependent features like initialization code, real-time operation etc. LECCS is one such cross development tool. It is a multi-platform development system based on the GNU family of freely available tools with additional point tools developed by Cygnus, OAR and Gaisler Research [7]. The most important property of LECCS is its ability to incorporate multi-tasking and real-time operations using RTEMS kernel.

RTEMS [18] is an acronym for Real – Time Executive for Multiprocessor Systems. It provides a high performance environment for embedded applications including many features such as: -

- TCP/IP Stack, UDP DHCP
- POSIX including API threads
- Debugging – GNU debugger, thread aware
- Multitasking capabilities
- Event-driven, priority based scheduling
- High level of user configurability.

3.4 Development of IP Library

To build an IP Library an entire graduate level class with sixteen students was divided into small groups working independently on cores [19]. It was essential

to define some specifications and guidelines to enable the integration of these into a complete System-on-Chip at a later stage. Each core was verified individually via pre-layout simulation, synthesis, place/route and post-layout simulation prior to attempting integration with the LEON-2 or other cores. Thus, we could be assured that adding a new core to our System-on-Chip design would not introduce any errors within the system and we need to test only for its interaction with the rest of the System-on-Chip platform. The task of integrating these cores into a System-on-Chip platform is greatly facilitated by using a common bus protocol to interconnect them. For this purpose, an AMBA–wrapper was created for each core such that it would enable the cores to act as AHB bus masters and APB bus slaves. Specification guidelines as defined in class were:

- Address width is 32 bit.
- Data width is 32 bit.
- **RESET** signal to initialize all the registers and rams.
- Data has to be loaded into the RAM.
- **GO** signal for IP blocks to start functioning.
- **Done** Signal to indicate output data is ready.

The Advanced Encryption Standard (AES) cipher core was obtained from opencores.com and is available for free [20]. Similarly other cores used in the thesis as bus-masters are FIR and FFT cores, which were generated internally at the University of Tennessee.

AES is the latest Federal Information Processing Standard (FIPS) [21]. AES is implemented using the Rjindael algorithm. This is a block cipher that takes in a key and input text in variable-bit block lengths. The current version can have 128, 196, 256-bit key to cipher data with block length of 128,196,256 with all the nine combinations possible. The AES core is basically two parts. The AES Cipher top and the AES Inverse Cipher top. The core comes along with a verilog test-bench. The test bench supplies the Key, Plain Text and Ciphered data (to cross-check simulation results) in blocks of 128 bits to test the functionality.

AES Cipher

The AES cipher core consists of a key expansion module, an initial permutation module, a round permutation module and a final permutation module. Figure 3.4.1 explains the block diagram for AES Cipher module. The round permutation module will loop internally to perform 10 iterations (for 128 bit keys).

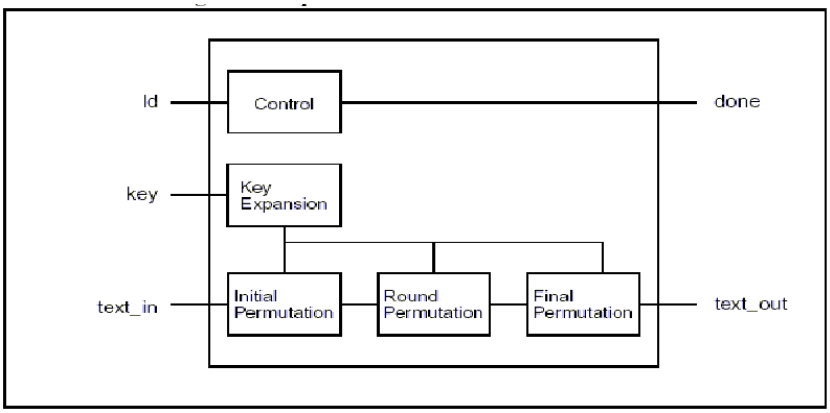


Figure 3.4.1 AES cipher block diagram

Task Requirements

Figure 3.4.2 explains the design flow for designing IP block and its verification.

- Simulate the AES Cipher (Open IP) core before synthesis.

- Synthesize the core targeting FPGA Xilinx Virtex 1000e and ASIC TSMC 0.18 technology using FPGA Compiler and Design Compiler.

- Place and Route the Synthesized design using *XVMake* (Xilinx Virtex) and *Silicon Ensemble* (ASIC) to get the SDF files for the design.

- Perform Post-Layout Back Annotated Simulation using SDF File for both technologies.

- Add DesignWare RAM to the front and back of the design to read the Key and Data required by the AES Cipher and write back the Ciphered text into the RAM.

- Perform Pre-Synthesis Simulations on the RAM-IP Core-RAM System.

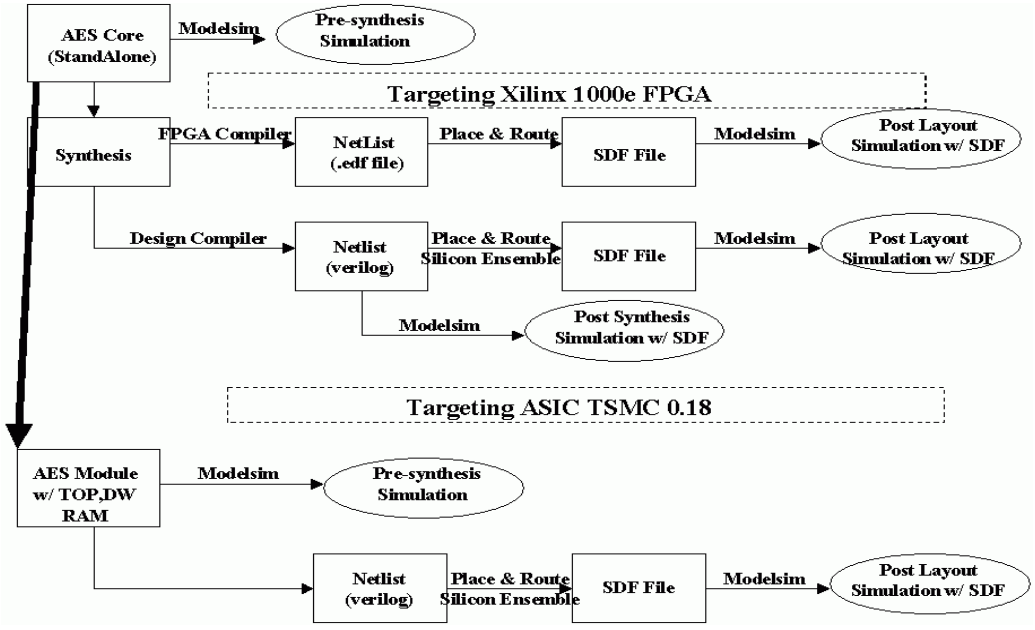


Figure 3.4.2 Design and verification tasks description

- Synthesize this system like Step 2 followed by Place and Route as in Step 3 and get the SDF files for both technologies

- Perform Post Layout Back Annotated Simulation using SDF File for both technologies.

3.5 Artisan RAM

The TSMC 0.18-micron synchronous dual-port SRAM is produced by a parameterized block generator, which allows great flexibility in the SRAM organization [22]. Three mux options are available which help in choosing the shape of the RAM.

The SRAM has two ports for the same memory locations. SRAM access is synchronous and is triggered by the rising edge of the clock, CLKA. Input address, input data, write enable, and chip enable are latched by the rising edge of the clock, respecting individual setup and hold times. The figure 3.5.1 shows the availability of the data on the output port after Ttz time period.

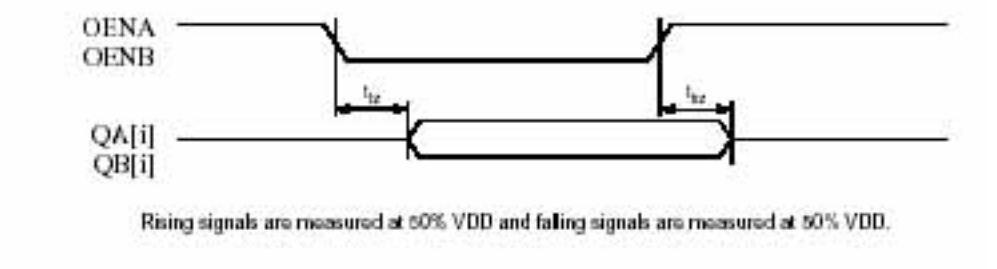


Figure 3.5.1 Artisan ram output set up time

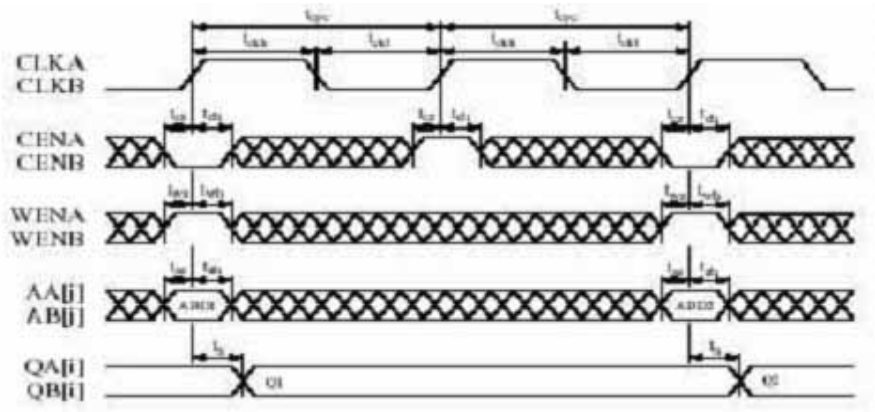


Figure 3.5.2 Artisan ram read cycle

To utilize this SRAM in our design we need to understand its read and write operation cycle and then create a wrapper to enable the communication with LEON-2 processor. LEON-2 processor is provided with a test bench to check if added RAM blocks are functioning in desired manner. Any error in meeting the timing constraints or data values results in cache failure. Figure 3.5.2 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 rising edge of the clock appears. Similarly while writing to a memory location at rising edge both data and address location should already be there at the data and address bus I/o ports. However if we see the simulation of LEON-2 processor read cycle in figure 3.5.3, it loads the address and data i/os at rising edge of the clock. This caused a failure in the LEON-2 processor. Therefore I have created a wrapper, which acts as an interface between LEON-2 and Artisan RAM.

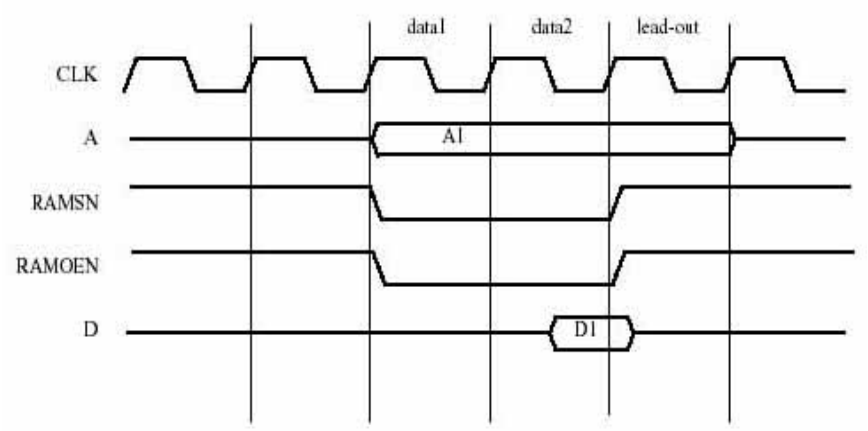


Figure 3.5.3 LEON-2 processor read cycle

This wrapper can be used with any Artisan RAM block since just the data width and address width need to be changed.

3.6 Clock Tree Generation

When complexity and size increase, the need to distribute clock signals in a controlled manner becomes very important. A large, pipelined chip may easily contain thousands of clocked elements (latches, flip-flops, etc.), and it is generally desired that the clocked parts switch at the same time, so it is obvious that a lot of buffering for clock signal is needed is needed.

In order to run *CT-Gen* [23], the normal design flow in *Silicon Ensemble* is broken up after the placement stage and a **DEF** file describing the design is saved. This file is then fed into *CT-Gen* along with some library files and after the

clock tree has been generated the design is imported back into *Silicon Ensemble* for routing (as shown in figure 3.6.1).

CT-Gen can be called from within *Silicon Ensemble* (this feature is available in version 5.4 or later) or it can be run as a stand-alone tool. For this thesis I tried both ways to implement a clock tree. For designs like “AES block” which are big enough to implement a clock tree but not as big as the complete System-on-Chip platform, both methods worked perfectly. However for a larger design with RAM blocks and hierarchy in the design I was not able to get a proper result using the *CT-Gen* tool. Therefore I used another tool by Cadence – Encounter [24], [25]. Using *Encounter* for large designs with hierarchy is really advantageous as it provides a very user-friendly interface to implement the clock tree.

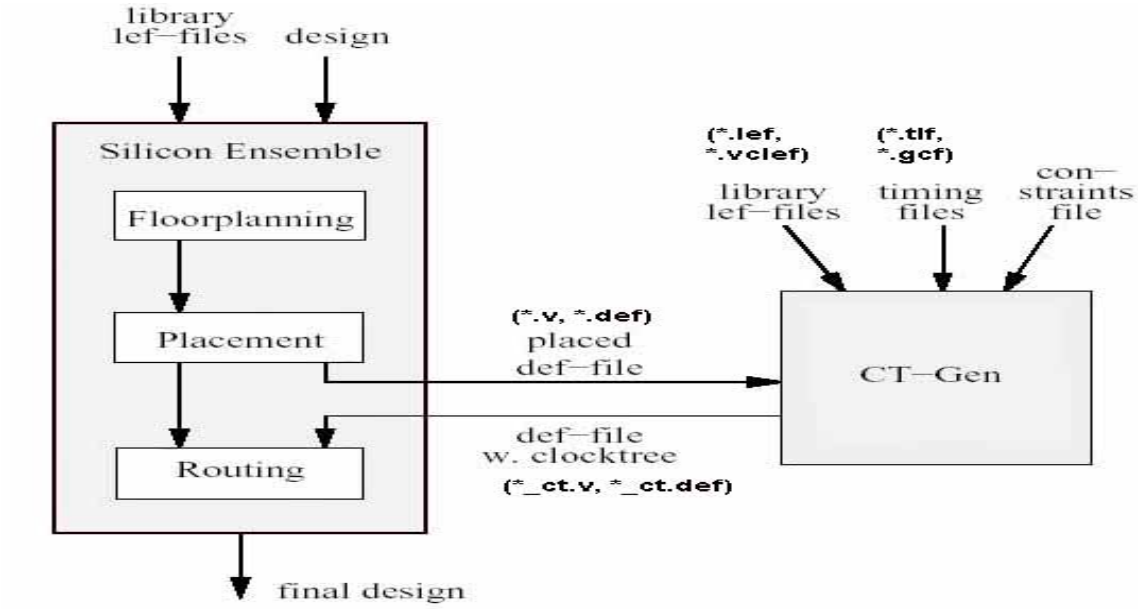


Figure 3.6.1 CT-Gen design flow

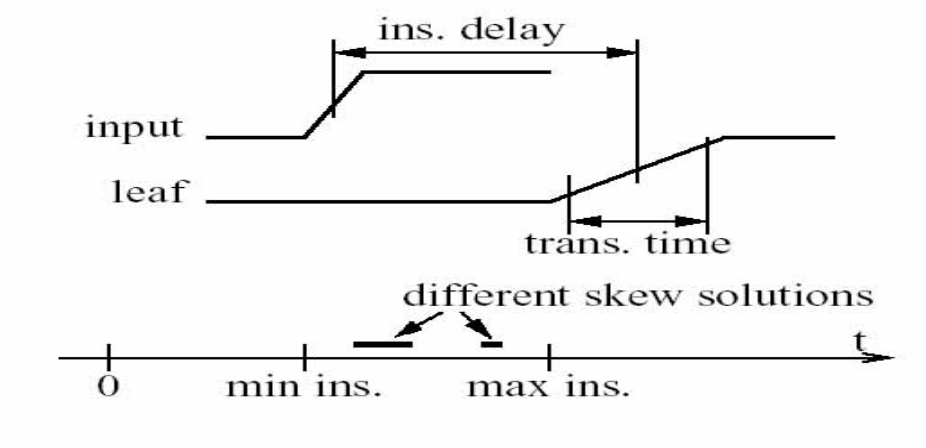


Figure 3.6.2 User constraints

Choosing the proper timing constraint for the clock implementation is really important since user-defined constraints force *CT-Gen* to insert buffers and inverters, forming a tree structure, into the clock distribution. Any existing buffering in the clock path will first be removed. The available components are picked from the timing file read into the generator. Constraints are the restrictions given to *CT-Gen*. These are in the form of what delays that can be accepted in the clock distribution. The constraints that the user can specify are as follows, figure 3.6.2.

. **max insertion delay**: Maximum delay from root to leaf pin.

. **min insertion delay**: Minimum delay to leaf pin. This is usually set to 0 but in some cases a higher value is required.

. **max skew**: The time difference between the fastest and the slowest clock path.

. **max transition time**: The 10% to 90% transition time at a leaf pin.

This tool can be used on other heavily loaded signals, such as reset, but that is a more complicated procedure. And a work-around for this problem can be leaving the reset signal active for more than 3-4 clock cycles.

Chapter 4: Implementation

4.1 Introduction

This chapter is dual purpose. As we are trying to keep our open core System-on-Chip platform in public domain, this chapter can be used as a tutorial for further development of this platform. This chapter also serves as a detailed description of the implementation for this thesis. In this chapter I have described the customization of the LEON-2 processor as well as integration of the complete platform followed by physical synthesis, place and route targeting the TSMC 0.18-micron technology. The System-on-Chip design flow that was followed while developing this platform can be described with the help of figure 4.1.1.

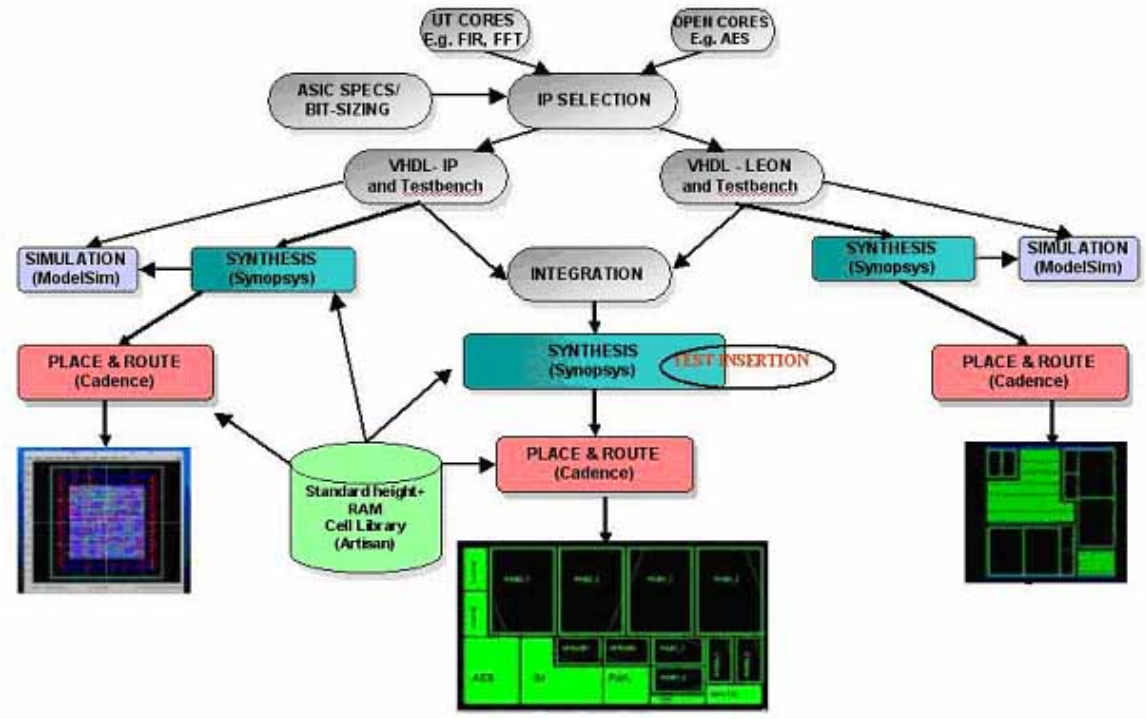


Figure 4.1.1 Flowchart for open core System-on-Chip platform

The entire development process is divided into three major steps.

1. Building the library of IP blocks to specifications.
2. Customizing and verifying the functionality of the LEON-2 processor.
3. Integrating the System-on-Chip platform and completing the chip design with physical synthesis, place and route and physical verification.

Building of the IP library and steps involved to determine correctness of their functionality was described in section 3.4 of this thesis. The next section begins with setting up the files for System-on-Chip implementation and discusses steps 2 and 3 mentioned above.

4.2 Setting up Files

I have used version LEON2-1.0.12 for my project. All files for this version and can be located at **/usr/cad/rishi/soc_research/leon2-1.0.12.tar.gz**

However, for our System-on-Chip project I had to modify various files and add VHDL models for various IP blocks into the existing files. Therefore, I have created another tar file, which contains all the files needed to implement this platform properly. These files are located at

/usr/cad/rishi/soc_research/soc.tar.gz

From your home directory proceed as follows.

- **gunzip -c soc.tar.gz | tar xvf –**

The open core System-on-Chip has the following directory structure:

soc	top directory
soc/Makefile	top-level makefile
soc/leon/	LEON-2 vhdl model
soc/modelsim/	*Modelsim* simulator support files
soc/pmon	Boot-monitor
soc/syn	Synthesis support files
soc/tbench	LEON-2 VHDL test bench
soc/tsource	LEON-2 test bench (C source)
soc/aes	AES vhdl model + AMBA wrapper for AES.
soc/fir	FIR vhdl model + AMBA wrapper for FIR.
soc/org_edit	Original files and edited files
soc/ram_tsmc25	ARTISAN RAM models to be used in SoC

4.3 Customizing the LEON-2 Processor

For TSMC-25 Technology

In /soc directory type following to start configuring the LEON-2 processor.

- **make xconfig**

Configuration window as shown in figure 4.3.1 should open.

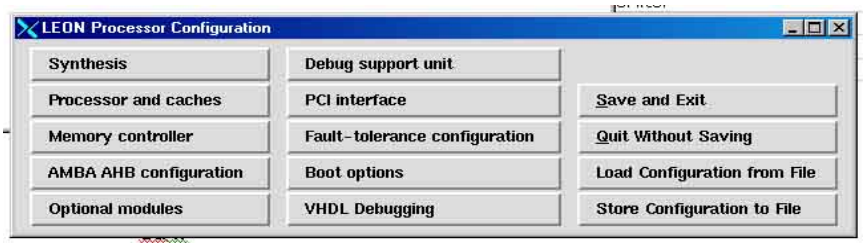


Figure 4.3.1 LEON-2 processor configuration window

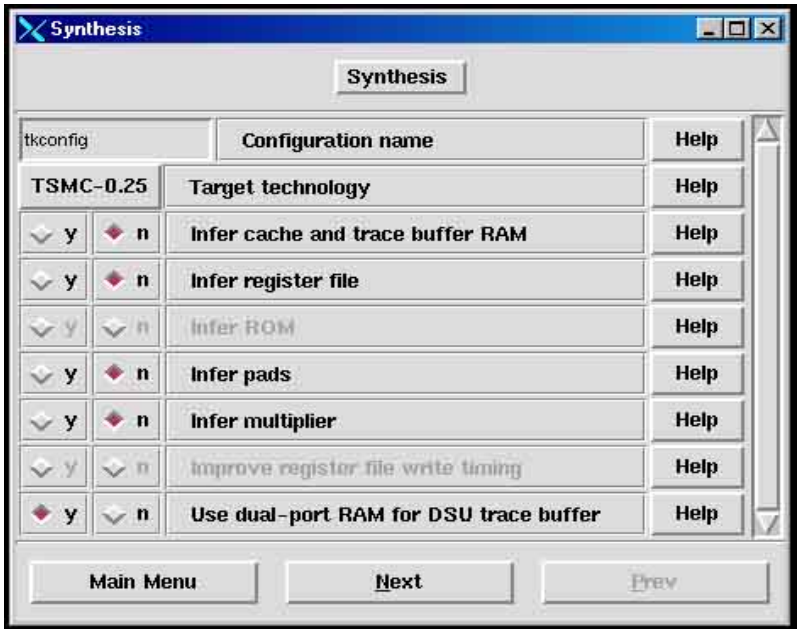


Figure 4.3.2 LEON-2 processor: Synthesis customization

In "**Main Menu**" click on "**Synthesis**" and a second window for synthesis customization will open. In that window select "**Target Technology**" to be "**TSMC25**" and you'll see all the variables in figure 4.3.2 below are automatically selected. For the time being, we'll use the default values for all the variables except one. We will configure the LEON-2 design without any pads. The reason for this is with pads we won't be able to simulate the design after synthesis. If we need to send this design for fabrication then we can add pads later.

Click on the "**Main Menu**" button and select "**Boot option**" in that window with "**Memory**" (Default is: Memory). Click on the "**Main Menu**" button and select "**Processor and caches**" and then select "**cache system**" and change the "**set**

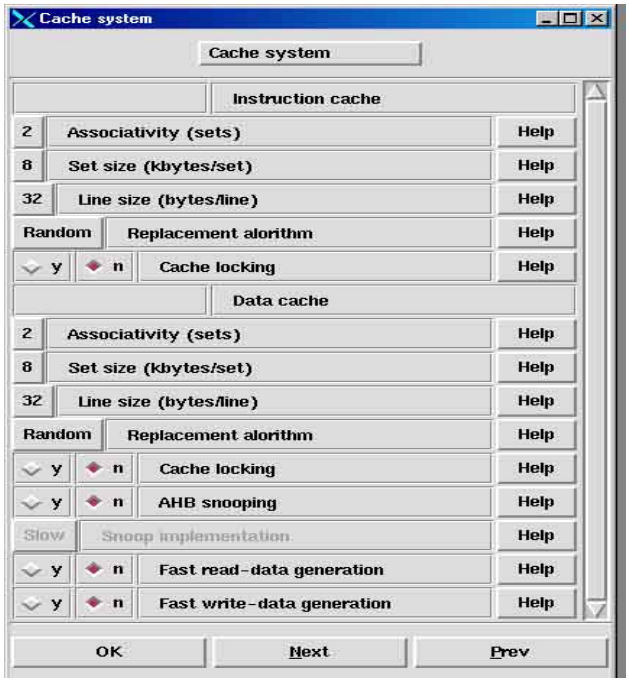


Figure 4.3.3 LEON-2 processor: Cache configuration

size" to **8k**. the entrees should be similar to the one shown in figure 4.3.3. Press "**OK**" and then "**Main Menu**".

Press "**Save and Exit**" button. This will prompt a new window informing you to type make dep. "**make dep**" which creates a DEVICE.VHD file, which contains the information about the customization that we have done.

- **make dep**
- **mentor_tools**
- **make all** (this will compile all the files)

Once the LEON-2 model has been compiled, use the TB_FUNC32 test bench to verify the behavior of the model. Simulation should be started in the top directory.

- **vsim tb_func32&**
- In the *modelsim* window type **run –all**

The output from the simulation should be similar to:

```
# # *** Starting LEON system test ***
# # Memory interface test
# # Cache memory
# # Register file
# # Interrupt controller
# # Timers, watchdog and power-down
# # Parallel I/O port
# # UARTs
# # Test completed OK, halting with failure
# ** Failure: TEST COMPLETED OK, ending with FAILURE
```

Simulation is halted by generating a failure.

4.4 Customizing Artisan RAM

Behavioral models for various RAMs that are needed to implement the data and instruction caches are provided in LEON-2 files. Even the registers in LEON-2 processors are implemented as dual-port RAMs. These behavioral models are technology-specific and are provided in the TECH_*.VHD files. Since we will synthesize the design generated for the TSMC25 process, all of the behavioral models can be found in TECH_TSMC25.VHD. To be able to synthesize the design and place and route it, we need to replace the behavioral models by

corresponding RTL models of Artisan RAM. To find out what size of RAMs we need in our design we may have to go back one step.

- **make all**
- **vsim tb_func32&**

In the *Modelsim* window we can see the size of the RAMs it is using, by going to the "**proc0**" model as shown in the figure 4.4.1. As we can see we need to use DPRAM of size 136x32 and SDRAM of size 256x27 & 2048x32. However we will use DPRAM instead of SDRAM too.

Exit from the *Modelsim* window. And in the main directory proceed as follows:

- **cd ram_tsmc25**
- **/sw/CDS/ARTISAN/TSMC18/aci/ra2sh/bin/ra2sh**

Figure 4.4.2 describes the Artisan RAM generator window that opens up. Entries specific to this project are described in table 4.4.1. We need to generate following views for each of our RAM design. 1. Verilog Model, 2. Synopsys Model, 3. TLF Model, 4. VCLEF footprint, 5. GDSII Layout.

Table 4.4.1 Entries for Artisan RAM generator

PARAMETERS	**DPRAM 136x32**	**RAM 256x27**	**RAM 2048x32**
Instance Name	dpram136x32_inst	ram256x27_inst	ram2048x32_inst
Number of words	256	256	2048
Number of width	32	27	32
Frequency (Mhz)	50	50	50
Multiplexer Width	4	4	8
Library Name	DPRAM1	RAM2	RAM3

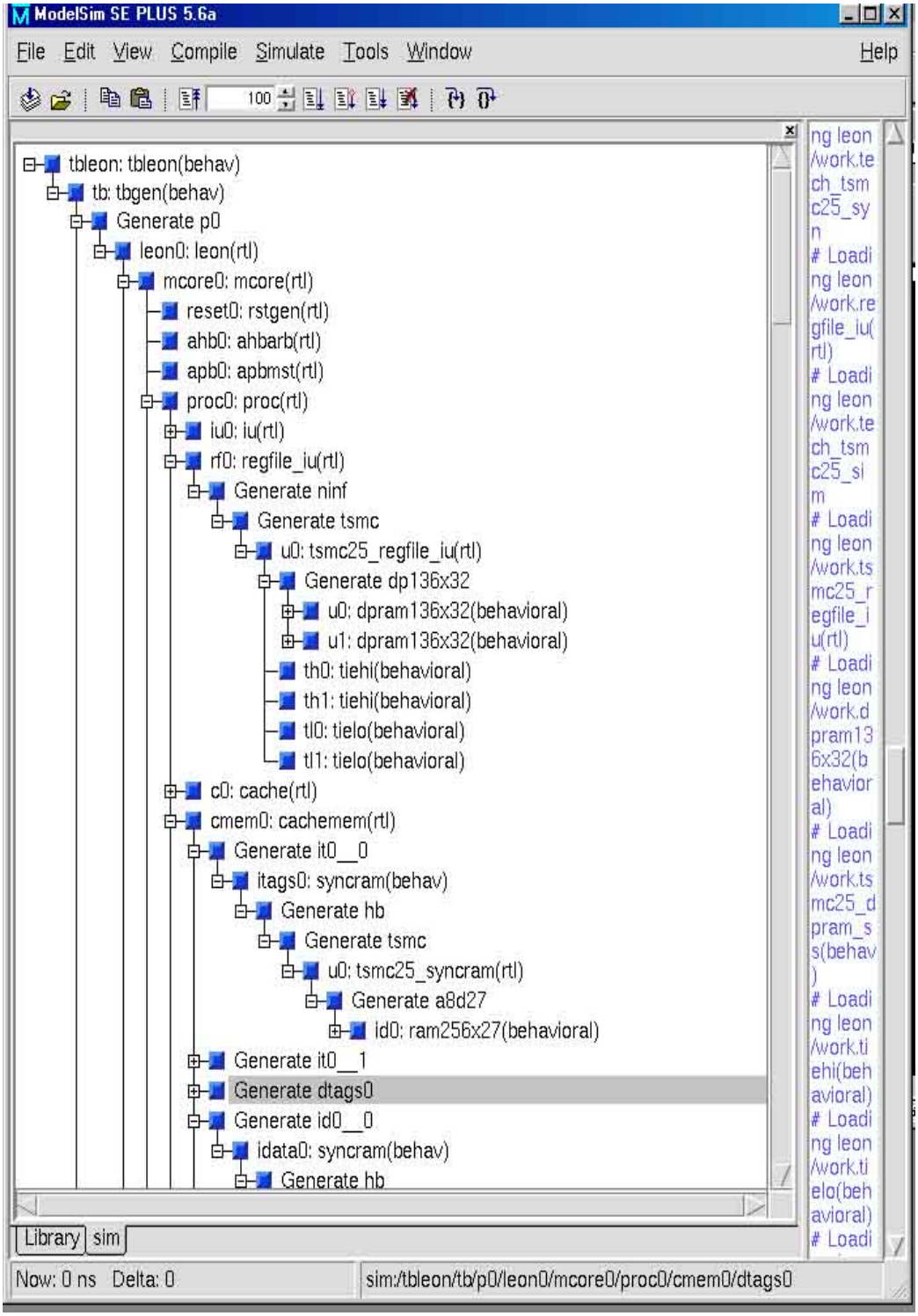


Figure 4.4.1 *Modelsim* window: RAM Size

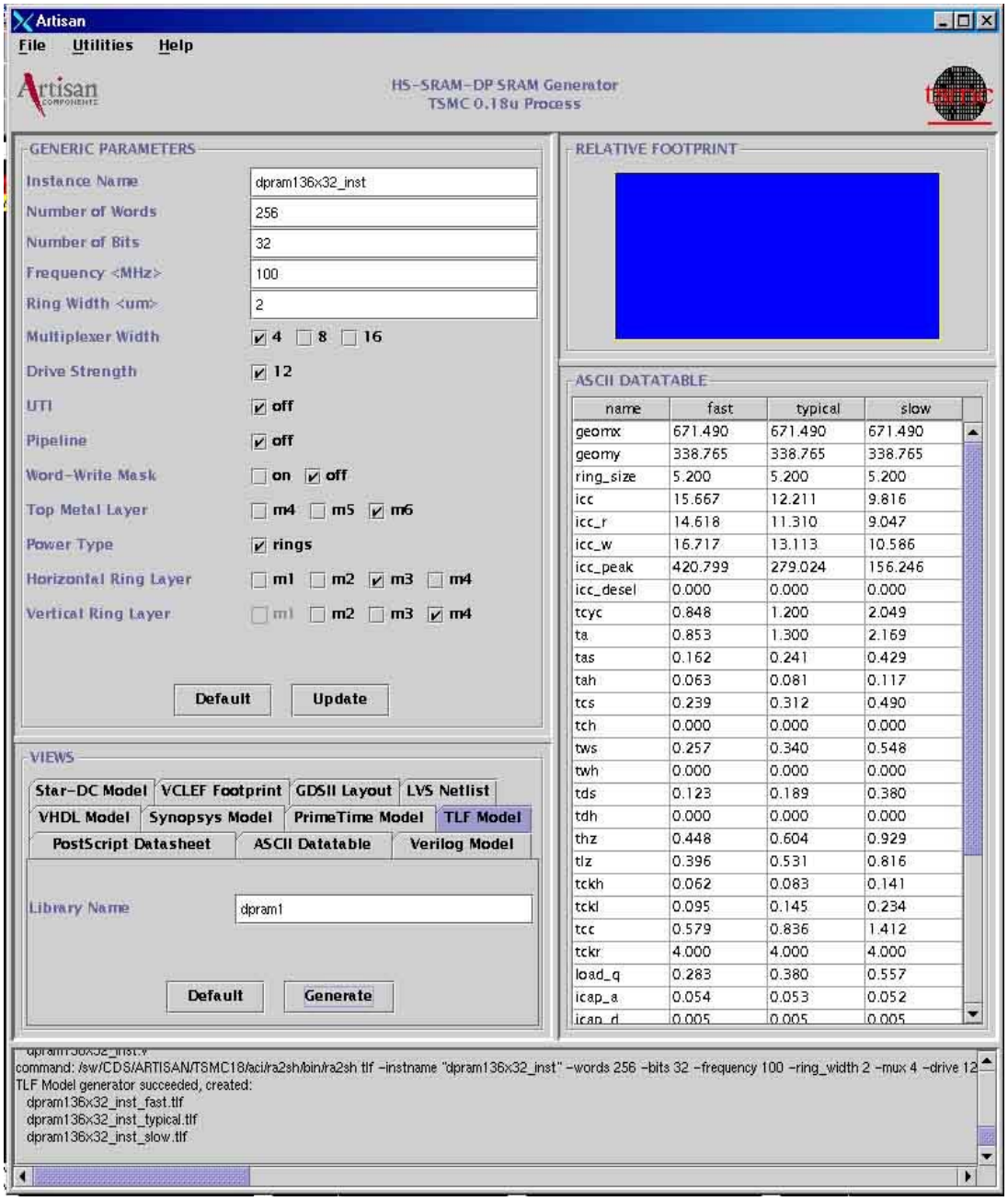


Figure 4.4.2 Artisan RAM generator

As discussed in section 3.5 these RAMs cannot be used as it is. We will have to create a wrapper around these block RAMs so that they are able to communicate with the LEON-2 processor in same fashion as the behavioral models do. These wrappers are provided in ram_tsmc25 directory.

dpram136x32_box0.vhd **ram2048x32_box0.vhd**
dpram136x32_box1.vhd **ram256x27_box0.vhd**

4.5 Synthesis: LEON-2 Processor

Make sure you have "**.synopsys_dc.setup**" & "**.synopsys_vss.setup**" already there in /syn directory. Design with Artisan RAM components complicates the synthesis process. The Verilog model of Artisan RAM is for simulation purposes only. We already have synthesized library for our Artisan RAM in the form "dpram136x32_inst_typical_syn.lib". Following is the way to use designs in .lib files for synthesis purposes. We first add the designs in the library (i.e. *.lib files) to a database (i.e. *.db files) and then add that database format to our tsmc18 cell database.

(Library to database conversion) **File name: lib2db.dcsh**

```
define_design_lib WORK -path WORK
read_lib ../ram_tsmc25/dpram136x32_inst_typical_syn.lib
write_lib DPRAM1 -format db -output ../ram_tsmc25/dpram136x32_inst_typical.db
read_lib ../ram_tsmc25/ram256x27_inst_typical_syn.lib
write_lib RAM2 -format db -output ../ram_tsmc25/ram256x27_inst_typical.db
read_lib ../ram_tsmc25/ram2048x32_inst_typical_syn.lib
write_lib RAM3 -format db -output ../ram_tsmc25/ram2048x32_inst_typical.db
quit
```

- **synopsys_tools**
- **dc_shell -f lib2db.dcsh**
- **rm –r WORK**

Now we need to edit "**.synopsys_dc.setup"** file to add the database of rams to tsmc18 cell database.

File Name: .synopsys_dc.setup

search_path = {} + search_path + /sw/CDS/ARTISAN/TSMC18/aci/sc/synopsys +
/sw/CDS/ARTISAN/TSMC18/PADS/synopsys/tpz973g_200c +
/home/rishi/652/soc/ram_tsmc25 + /home/rishi/652/soc/ram_virtex2
link_library = {typical.db"*"}
target_library = typical.db
symbol_library = typical.db
syntetic_library = { /sw/synopsys/libraries/syn/dw06.sldb + /sw/synopsys/libraries/syn/dw02.sldb
+ /sw/synopsys/libraries/syn/dw01.sldb }
link_library = target_library + synthetic_library + dw06.sldb + dw03.sldb + dw02.sldb +
dw01.sldb + tpz973gtc.db + **dpram136x32_inst_typical.db +**
dpram512x36_inst_typical.db + ram2048x32_inst_typical.db +
ram256x27_inst_typical.db
search_path = search_path + {synopsys_root + "/dw/sim_ver"}

Generating Black Boxes for each of RAM component.

- **cd syn**
- **rm –r WORK**
- **mkdir WORK**

File Name: ram_box.dcsh

define_design_lib WORK -path WORK
analyze -f vhdl -library WORK ../ram_tsmc25/**ram256x27_box0.vhd**
elaborate **ram256x27_box0**

```
uniquify
compile -map_effort high
write -f verilog -hierarchy -o ../leon/ram256x27_box0.v
quit
```

Note# Please substitute the name of ram file in ram_box.dcsh also delete WORK directory after every run.

- **synopsys_tools**
- **dc_shell –f ram_box.dcsh** (do it for each ram model)

Replacing the LEON-2 files so that new files use these ram black boxes instead of original behavioral models. For that purpose we will have to replace original tech_tsmc25.vhd with a modified module.

- **cd leon**
- **cp ../org_edit/tech_tsmc25-rishi.vhd tech_tsmc25.vhd**
- **cd syn**
- **cp /org_edit/leon-syn.dcsh leon.dcsh**
- **rm –r WORK**
- **mkdir WORK**
- **synopsys_tools**
- **dc_shell –f leon.dcsh > zm01.txt**

This is going to take a while and we can keep checking the output file (zm01.txt) for errors. To get a post synthesis simulation of the netlist:

- **cd leon**
- **cp ../syn/leon.v .**
- **rm leon.vhd**
- **cp /org_edit/Makefile_synth Makefile**

- **cp /org_edit/tsmc18.v .**
- **cp /org_edit/tp*.v .**
- **cp ../ram_tsmc25/ram*box0*.v .**
- **cd ..**
- **cd tbench**

Editing the testbench (tbgen.vhd) to specify the clock speed.

File Name: tbgen.vhd

```
Note: We need to edit the frequency( clkperiod = 50 ; ie freq =25MHz)
entity tbgen is
 generic (
    msg1      : string := "32 kbyte 32-bit rom, 0-ws";
   msg2      : string := "2x128 kbyte 32-bit ram, 0-ws";
   pci       : boolean := false;       -- use the PCI version of leon
    DISASS    : integer := 0;   -- enable disassembly to stdout
   clkperiod : integer := 50;        -- system clock period
   romfile   : string := "tsource/rom.dat";  -- rom contents
   ramfile   : string := "tsource/ram.dat";  -- ram contents
```

- **cd ..**
- **make clean**
- **mentor_tools**
- **make all**
- **vsim tb_func32**
- In *modelsim* window type **run –all**

If synthesis was done properly then the netlist should pass the entire component test provided by the LEON-2.

```
# # *** Starting LEON system test ***
# # Memory interface test
# # Cache memory
```

```
# # Register file
# # Interrupt controller
# # Timers, watchdog and power-down
# # Parallel I/O port
# # UARTs
# # Test completed OK, halting with failure
# ** Failure: TEST COMPLETED OK, ending with FAILURE
```

Simulation is halted by generating a failure.

4.6 System-on-Chip platform: Adding IP blocks

For adding an IP block to LEON-2 processor we have to complete the following two tasks.

- Creating a bus master.
- Preparing LEON-2 files to recognize new bus master.

For AES to act as a bus master I have created a wrapper that would enable it to communicate through AMBA busses. This wrapper is in two parts – AES.VHD and AES_CTRL.VHD (FIR.VHD and FIR_CTRL.VHD).

- **cd leon**
- **cp ../aes/DW_ram*.vhd .**
- **cp ../aes/aes*.vhd .**
- **cp ../aes/controller.v .**
- **cp ../aes/topmodule.v .**
- **cp ../aes/aes.vhd .**
- **cp ../aes/aes_ctrl.vhd .**

Second step involves modifying the LEON-2 processor files to include AES as bus master. For this purpose files that need to be changed are - MCORE.VHD, TARGET.VHD, AMBACOMP.VHD, and DEVICE.VHD. Copying the modified files:

- **rm ambacomp.vhd mcore.vhd target.vhd device.vhd Makefile***
- **cp ../org_edit/ambacomp-soc.vhd ambacomp.vhd**
- **cp ../org_edit/mcore-soc.vhd mcore.vhd**
- **cp ../org_edit/target-soc.vhd target.vhd**
- **cp ../org_edit/device-soc.vhd device.vhd**
- **cp ../org_edit/Makefile-soc Makefile**
- **cd..**
- **make clean**

Now we need to change the software for the LEON-2 processor so that we can program the transfer of data from registers in LEON-2 to the memory of the IP blocks. Once this operation is complete, LEON-2 will have to generate control signals for the respective IP blocks corresponding to the operation it wants to be done. The first task is deleting the original RAM.DAT file. The second task is cross compiling the software files to generate a new RAM.DAT file containing the information about the operation to be performed by the LEON-2 processor.

- **cd tsource**
- **rm ram.dat**
- **make clean**
- **cp leon_test.c leon_test-org.c**
- **cp /org_edit/leon_test.c .**
- **bash**

In response to the bash prompt, please set following path:

- **export PATH=$PATH:/opt/rtems/bin**
- **make all**

After **make all,** it should compile without errors.

- **exit**

Now we have set all the files and we can simulate the design. All the relevant signals can be seen by running the wave file aes.do

- **cd ..**
- **mentor_tools**
- **make all**
- **do aes.do** (in the *modelsim* window)
- **run -all** (in the *modelsim* window)

You can see how data communication is taking place the between bus-master (in this case AES) and LEON-2 by watching the simulation results of signals in the AES_CTRL module. As shown in figure 4.6.1, the control signals being received by the IP block from the LEON-2 processor through APB bus at Register ports 0x80000300-318. Any change in the value of these ports triggers a corresponding operation in the AES_CTRL module. In figure 4.6.2 the IP block is requesting the bus and after the AHB bus has been granted it is accessing the data from memory and loading onto the AES block rams which is followed by a "go" signal.

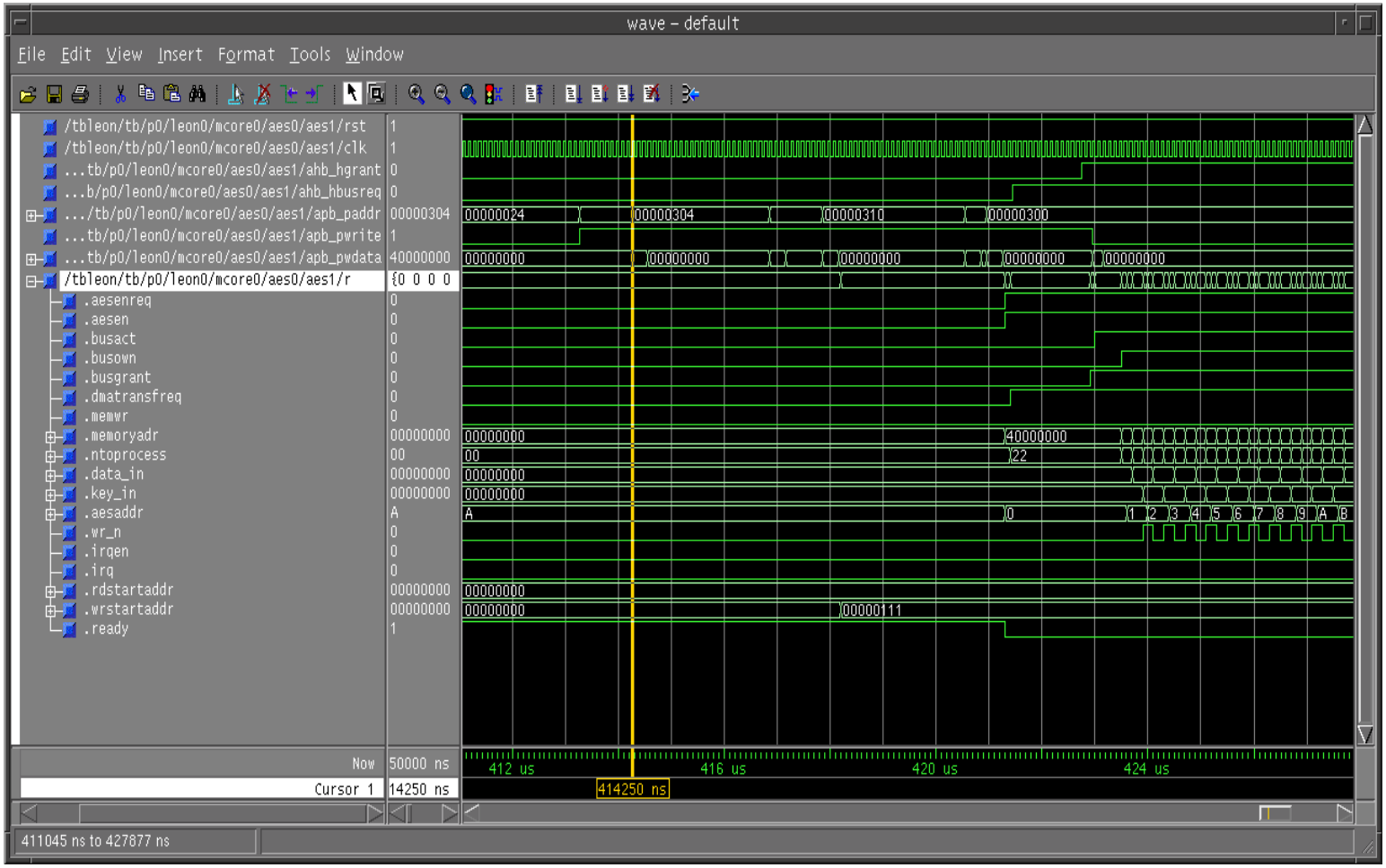


Figure 4.6.1 LEON-2 Processor: Transmitting control signals to IP Block

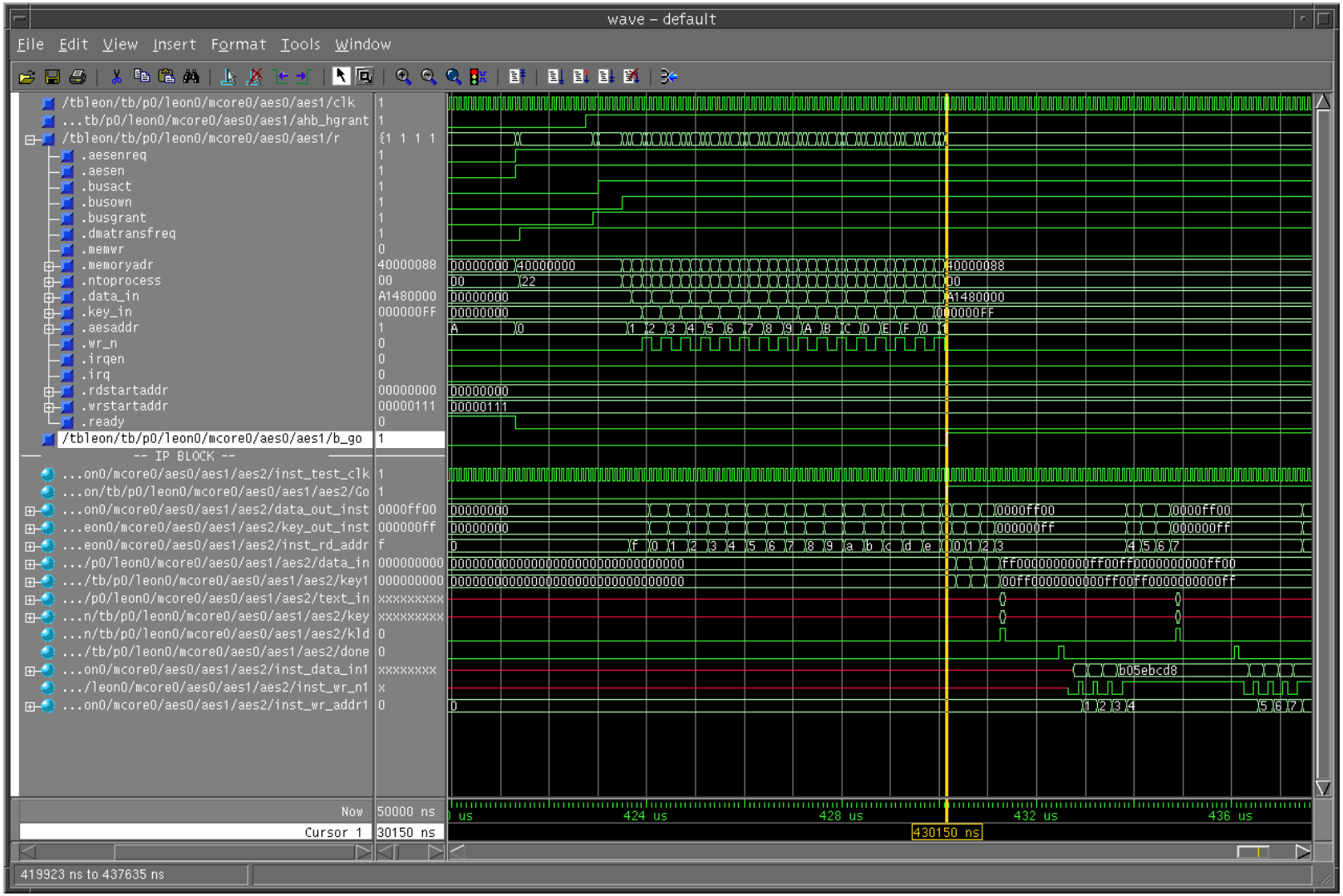


Figure 4.6.2 IP Block: Performing tasks assigned

4.7 System-on-Chip platform: Synthesis

Once synthesis for the LEON-2 processor in section 4.4 is successful, then the tasks in this section are straightforward.

Remove and create a new WORK directory.

- **synopsys_tools**
- **dc_shell –f soc.dcsh** > zm-03.txt

Check the file zm-03.txt for any errors from synthesis and simulate the file in similar fashion as in section 4.4. Figure 4.7.1 shows the control signals being received by the IP block.

Figure 4.7.2 shows the IP block performing the tasks assigned and loading the data onto the RAM blocks. It also shows the data being loaded on to the IP core AES and giving load signal which is followed by a done signal in approximately 12 clock signals which indicates the 128 bit data has been encrypted.

4.8 System-on-Chip platform: Place & Route

Place and route is an elaborate process and to discuss each detail is out of the scope of this document. I will however, highlight the steps followed during layout generation.

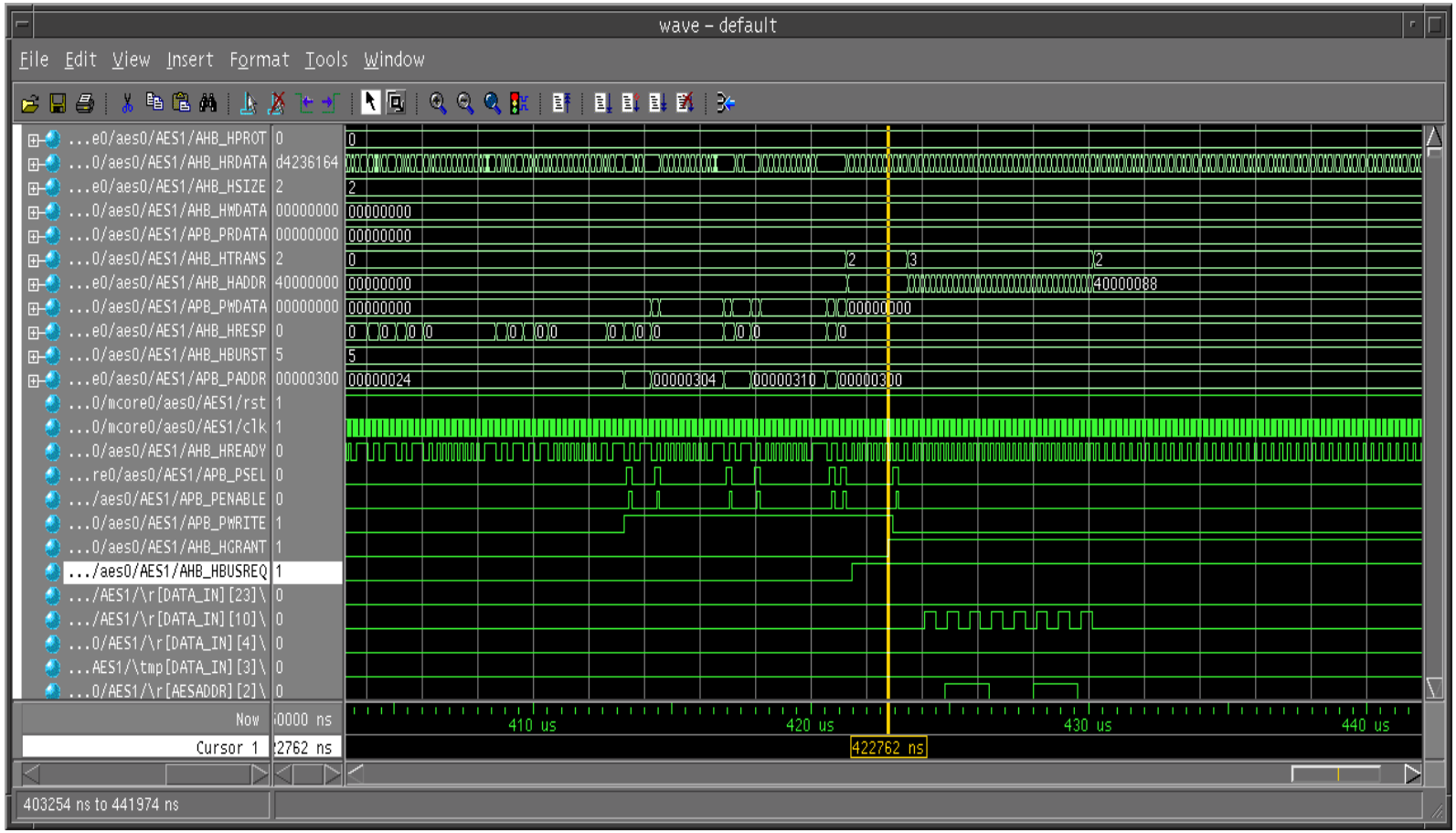
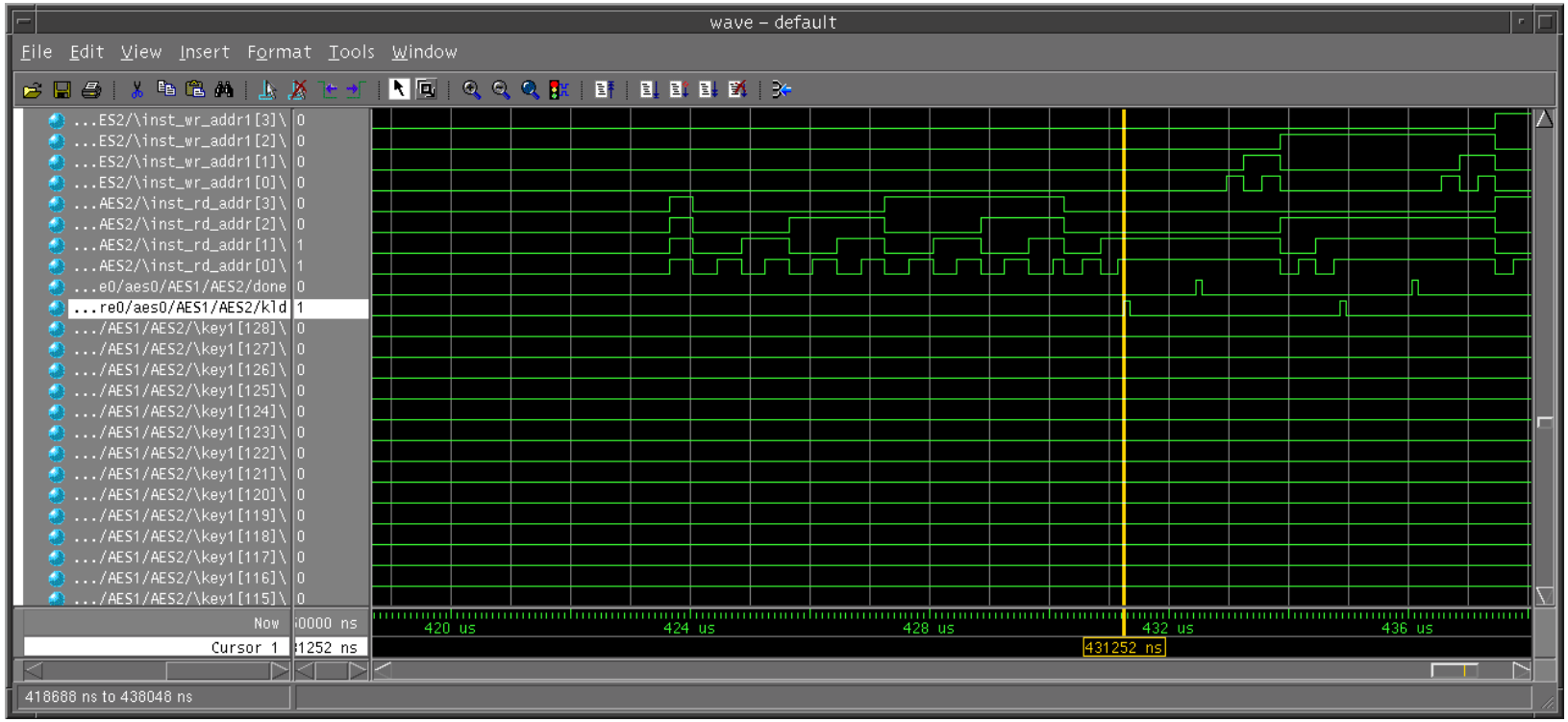


Figure 4.7.1 LEON-2 Processor: Transmitting control signals to IP Block



4.7.2 AES wrapper: Completing the tasks assigned

- Floorplanning – *Encounter*: Sown in figure 4.8.1
- Power Planning – *Encounter*: Shown in figure 4.8.2
- Place – *Encounter*: Shown in figure 4.8.3 and figure 4.8.4
- Clocktree insertion – *Encounter*
- Add filler cells – *Encounter*
- Export design – *Encounter*
- Import design – Design in *Silicon Ensemble* is shown in figure 4.8.5.
- Connect Rings – *Silicon Ensemble*
- Verify Geometry – *Silicon Ensemble*
- Verify connectivity – *Silicon Ensemble*
- Route – *Silicon Ensemble*: Shown in figure 4.8.6
- Verify Design – Silicon Ensemble
- Export def, gdsII formats.
- Perform post-layout back annotation simulation using SDF file.

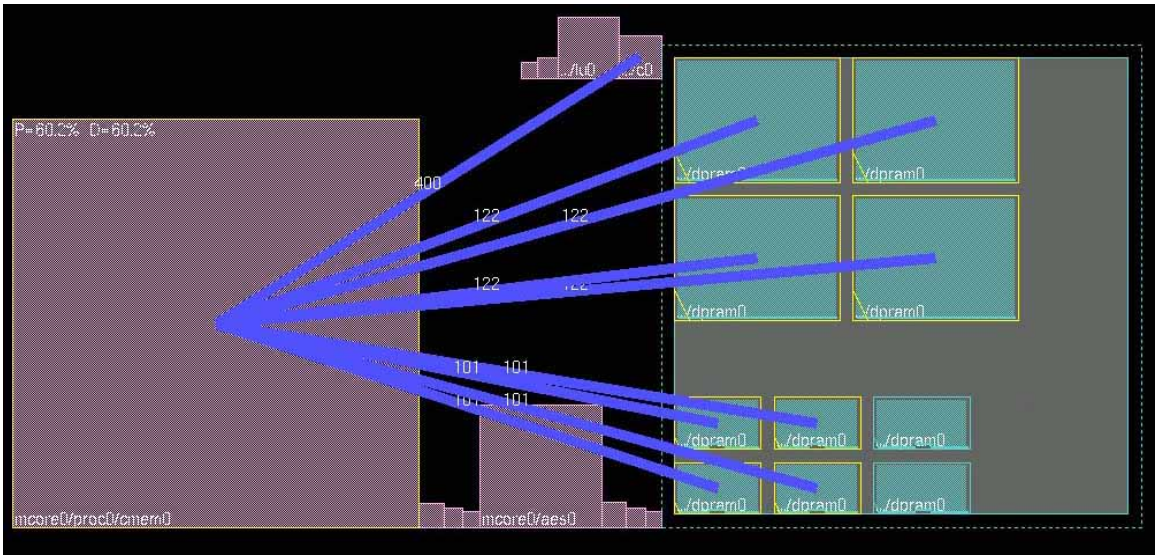


Figure 4.8.1 System-on-Chip platform: Initial Floorplanning

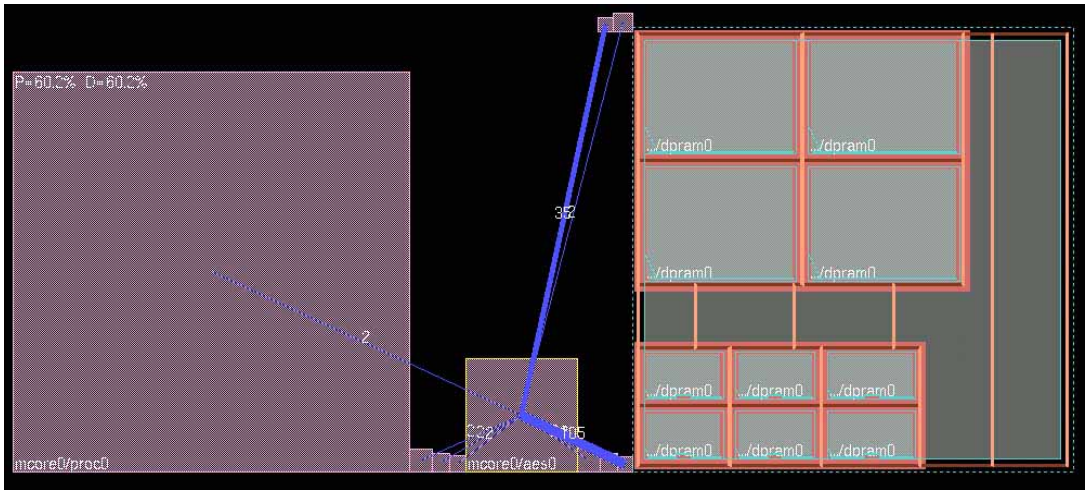


Figure 4.8.2 System-on-Chip platform: Power planning

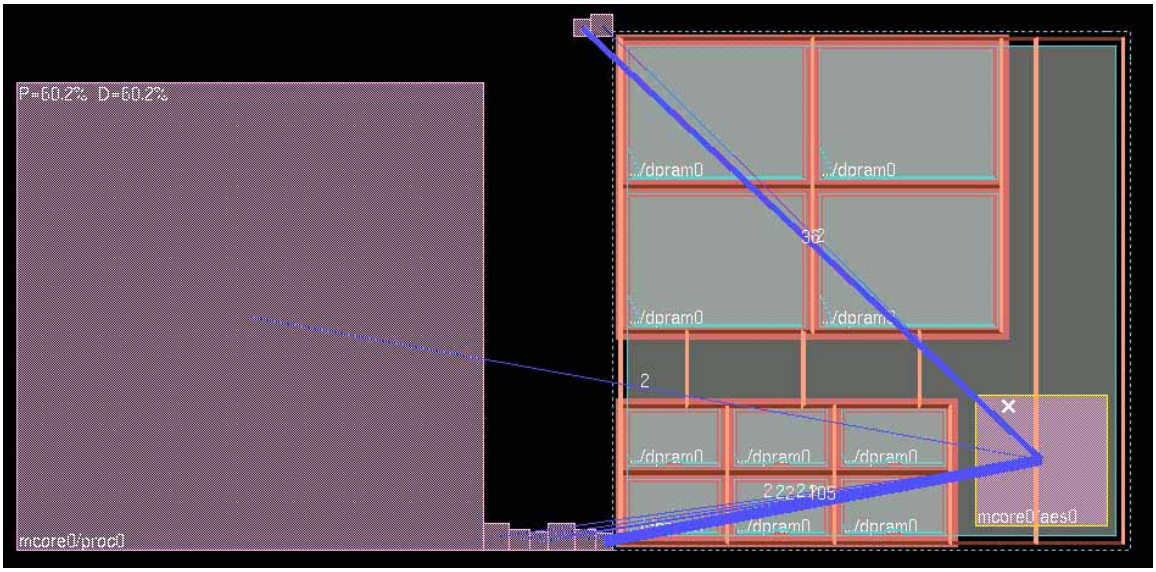


Figure 4.8.3 System-on-Chip platform: Place customization

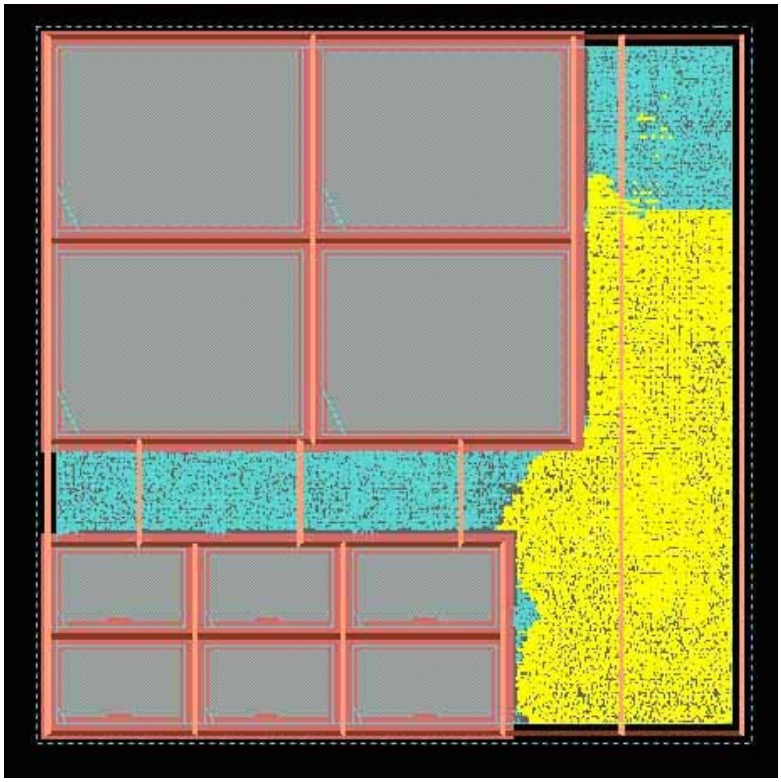


Figure 4.8.4 System-on-Chip platform: Placed

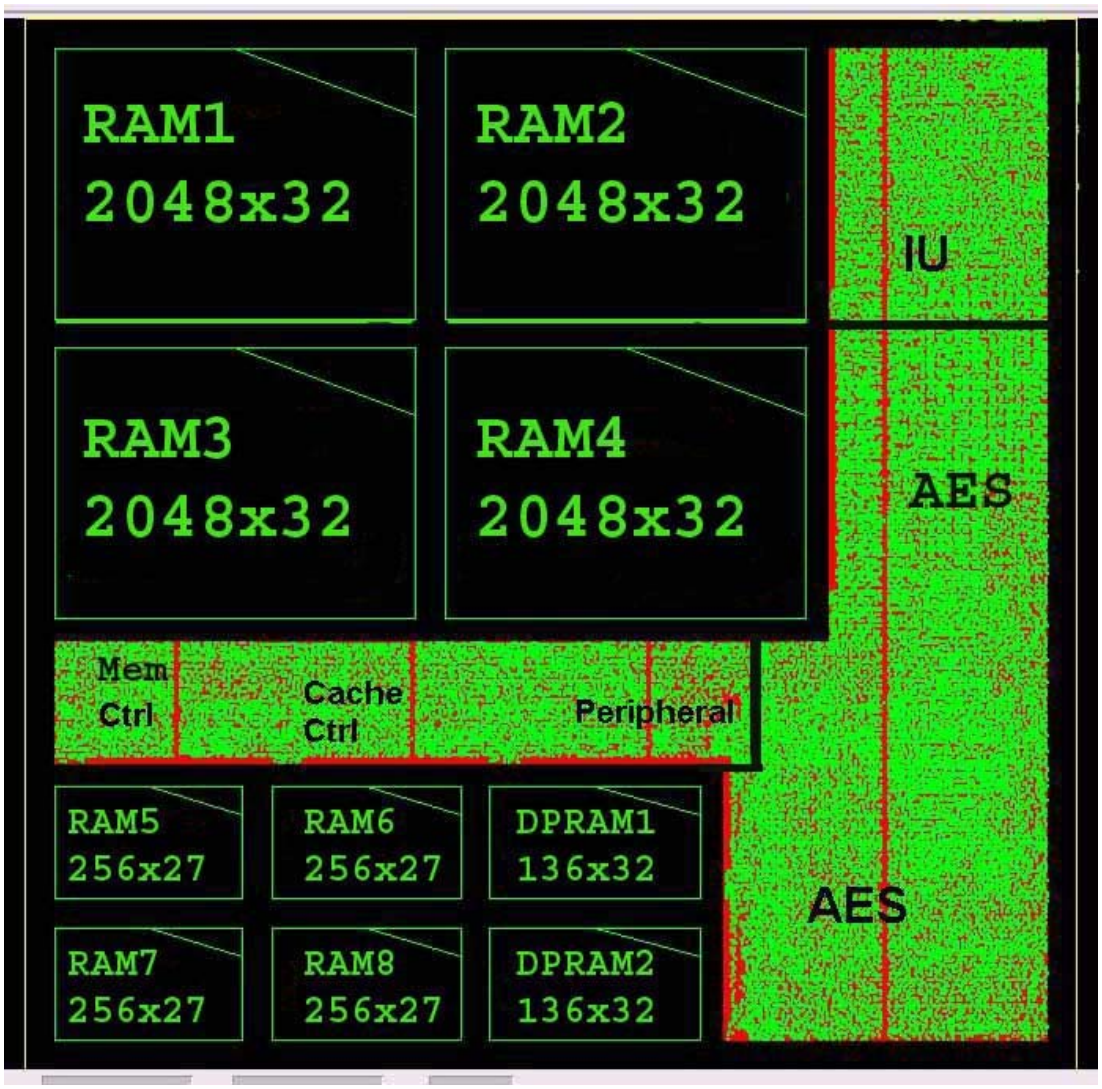


Figure 4.8.5 System-on-Chip platform: Import placed file

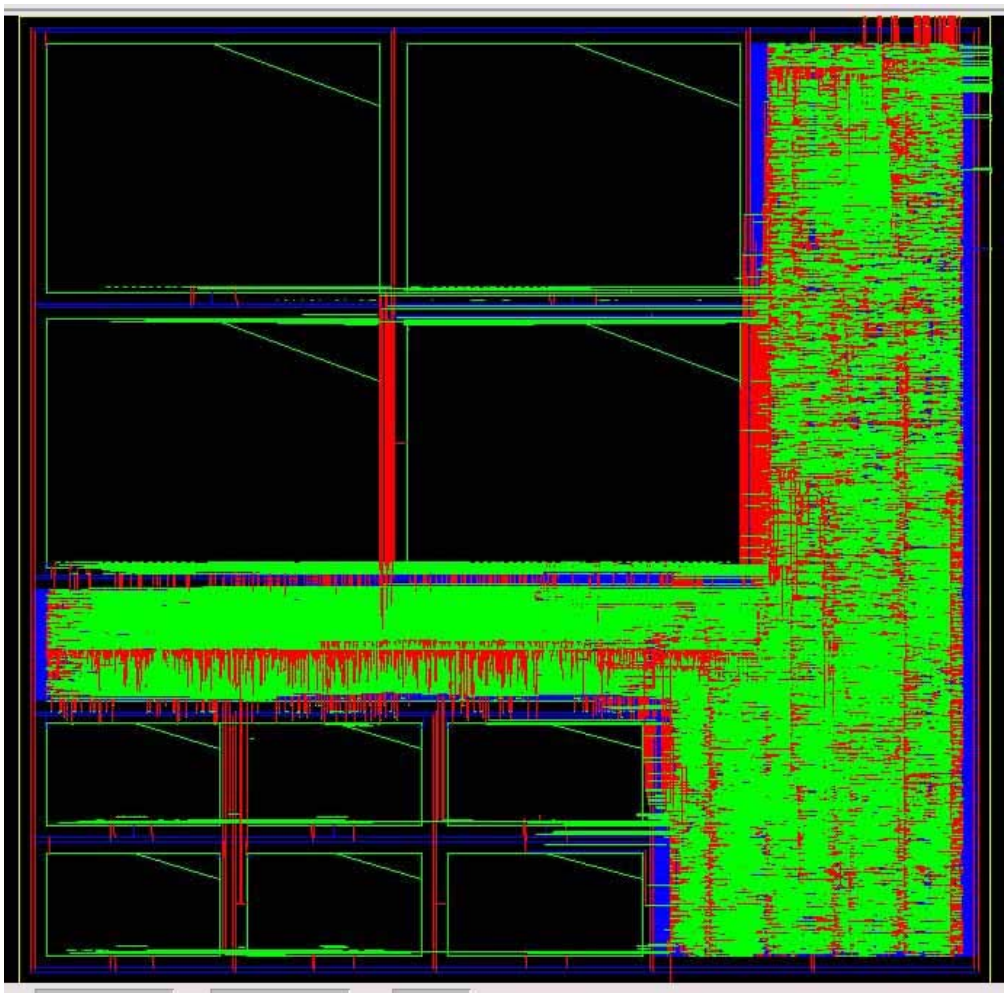


Figure 4.8.6 System-on-Chip platform: Route completed

Chapter 5: Results, Conclusion and Future Work

5.1 Results

As mentioned earlier, the objective of this thesis was to implement a System-on-Chip platform. To achieve this goal we not only had to have a working RTL model but also synthesize, place & route the design. This design, which has more than four million transistors, posed various design problems. Therefore, testing the functionality of the design after each design step – synthesis, clock tree insertion, and routing was important. For this purpose simulation of the design net-list was done using *Modelsim*.

Simulation results after the stages – RTL design, synthesis, clock tree insertions were shown in the previous chapter. Final post-layout back-annotation simulation was done to test if the design was properly placed and routed. For this purpose a standard delay file (SDF) of the design was generated using the Hyper-extract tool. Figure 5.1.1 and 5.1.2 shows the simulation using *Modelsim* with delay information provided by the SDF file. The baseline System-on-Chip was operated up to a speed of 25 MHz with basic timing constraints.

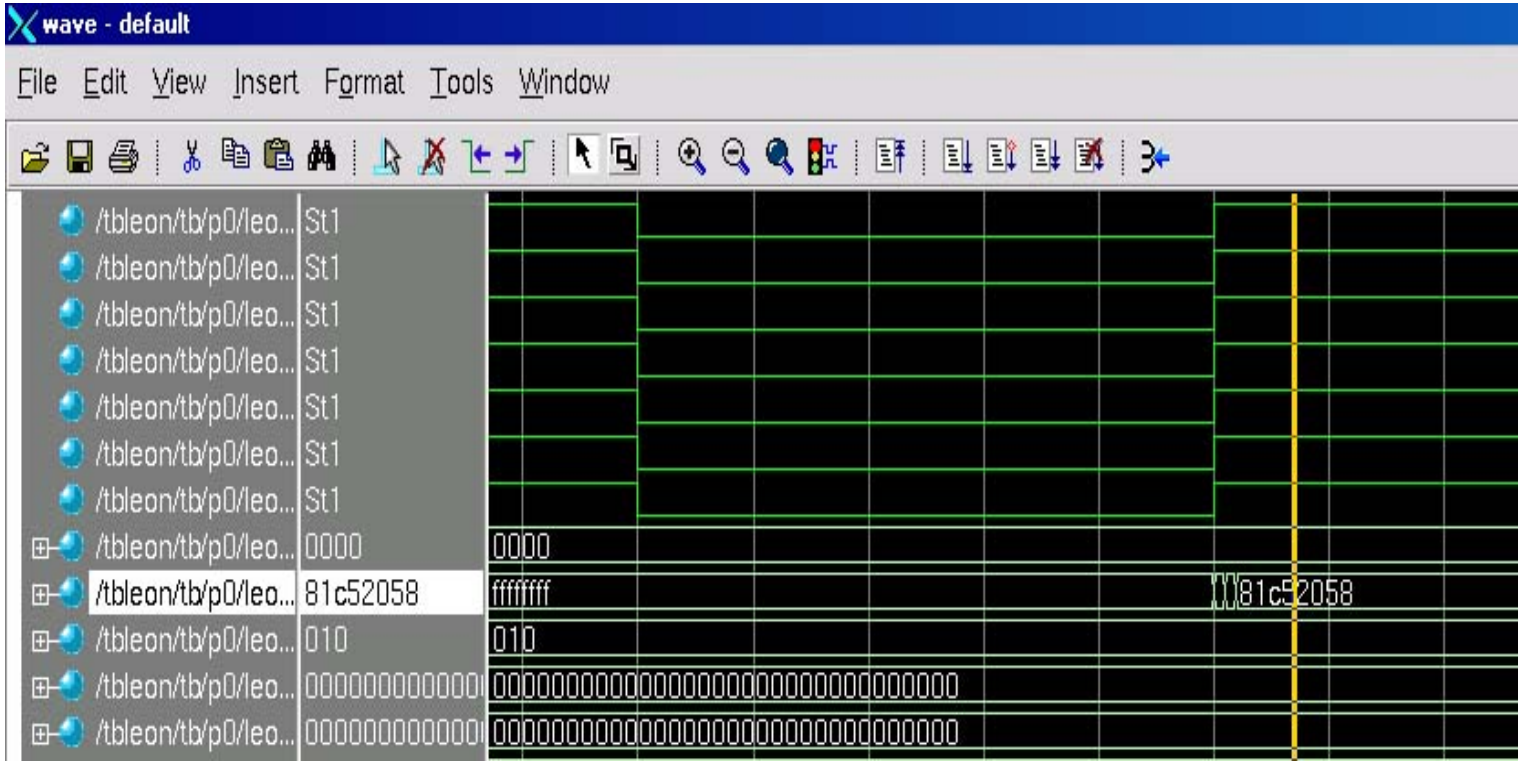


Figure 5.1.1 Back-annotated simulation results

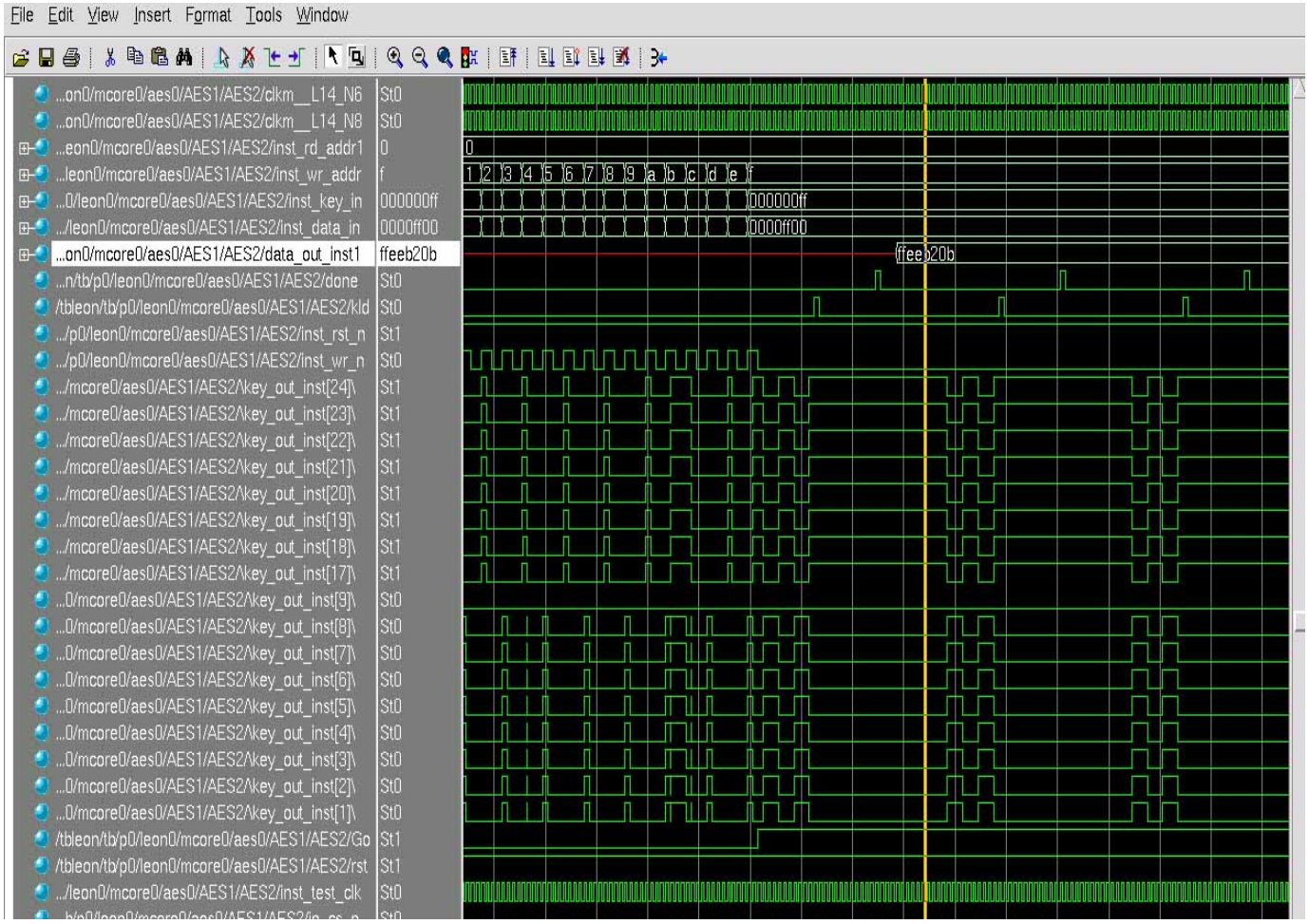


Figure 5.1.2 Back-annotated simulation for correct functionality

To calculate the number of transistors and standard-cell instances in the design I have used a script generated internally at the University of Tennessee to actually count the number of standard-cell instances in the net-list. A second script then substitutes the number of transistors in each standard-cell instance based on the information from the spice net-list (*.CDL) to calculate the number of transistors in the design. These scripts are provided in the soc/count/ directory. Table 5.1.1 provides the comparison of standard-cell instance count and transistor count after adding each IP to the LEON-2 processor.

Figure 5.1.3 shows the layout after importing the DEF file containing the placement information of the design into the *Virtuoso* layout editor.

Table 5.1.1: Standard-Cell instances & Transistor count of the design

Design	Instance Count	Transistor Count
LEON-2 CPU	10022	117038
LEON-2 CACHE/RAM	10	2730118
LEON-2 Processor	10032	2847226
AES	17496	565800
FIR	96108	1016734
LEON-2 Processor + AES	30034	3075779
LEON-2 Processor + AES + FIR	130008	4132824

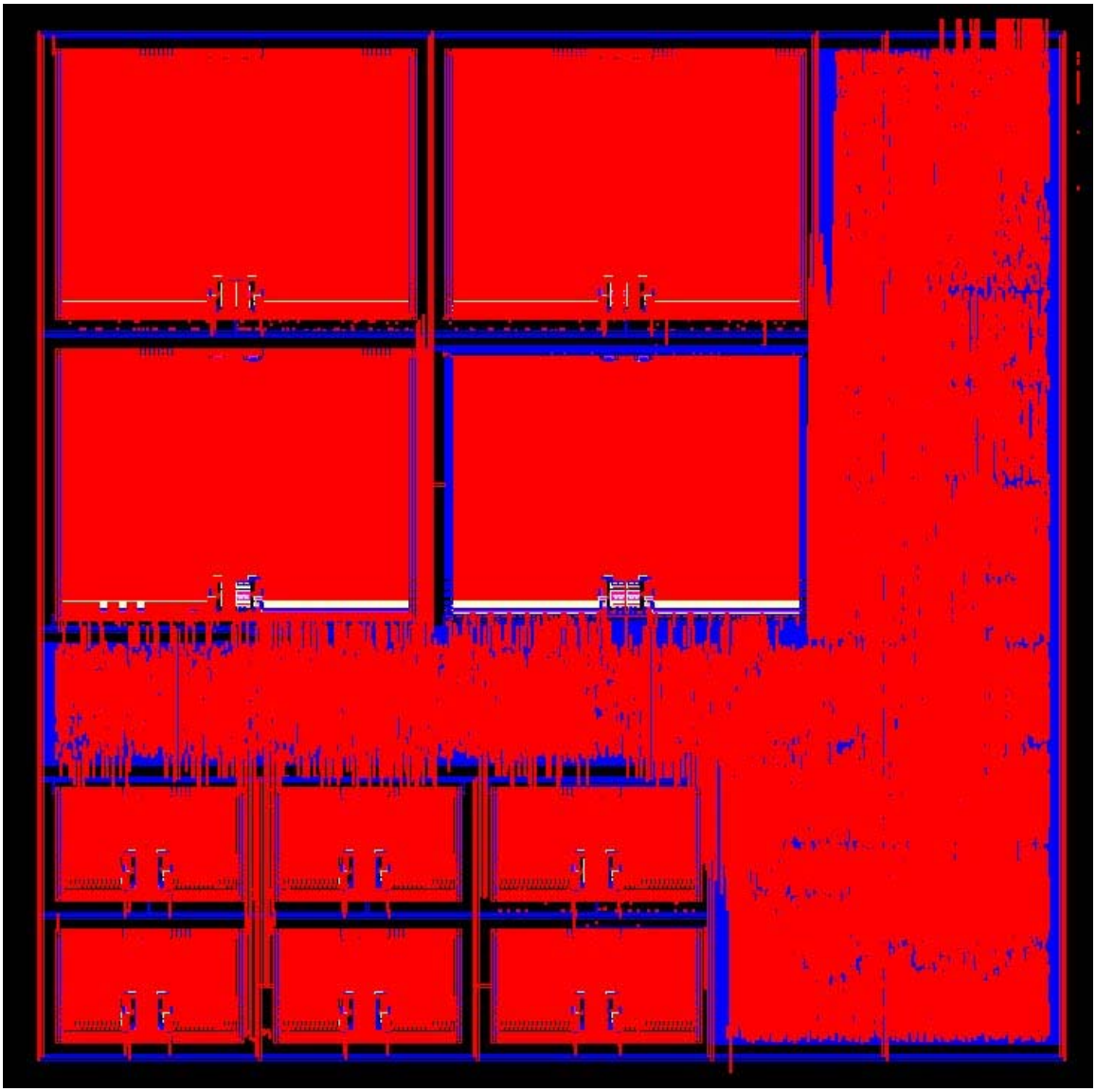


Figure 5.1.3 System-on-Chip platform: Final layout

5.2 Conclusion

- All major goals for this project were realized.
- Base platform established for further enhancements.
- Current implementations and future scope comprehensively documented with a supporting tutorial.
- The integrated System-on-Chip platform contains approximately four million transistors and around one hundred and thirty thousand standard-cell instances.

5.3 Future Work

This platform is ready for further development and at the University of Tennessee further work is being done on this platform under the title “The Volunteer SoC”. Some preparations that I have made to enable performance improvement as a part of future work are located in the soc/timing/ directory which contains files for performance improvement using a timing constraint that can be provided during synthesis and files for performing Static Timing Analysis (STA) using the Synopsys - *Primetime* tool. These tasks will require various teams working on different components of this System-on-Chip platform and integrating them on a later stage and therefore were not taken as part of this thesis.

List of References

[1] S. Thakur, Jeremy Buckle, "Introduction to Human Computer Interaction: Moore's Law," [Online]. Available: http://www.cs.indiana.edu/~sithakur/l542_p2/1jepbuckl-2sithakur.html, unpublished.

[2] R. Nair, "Effect of increasing chip density on the evolution of computer architectures," IBM J. RES. & DEV. VOL. 46 NO. 2/3 MARCH/MAY 2002. [Online]. Available: http://www.research.ibm.com/journal/rd/462/nair.pdf.

[3] Pieter Burggraaf, "ASIC Highlights," in *STATUS 1998,* ISBN: 1-877750-65-4, Integrated Circuit Engineering Corporation, AZ, 1998.

[4] Udaya Kamath, Rajita Kaundin, "System-on-Chip Designs," White Paper, June 2001. [Online]. Available: http://www.wipro.com/pdf_files/Wipro_System_on_Chip.pdf, unpublished.

[5] Rick Mosher, AMI Semiconductor, "*Structured ASIC Based SoC Design,*" [Online]. Available: http://www.us.design-reuse.com/articles/article7058.html, unpublished.

[6] Jiri Gaisler, Gaisler Research, The LEON-2 Processor. [Online]. Available: http://www.gaisler.com/.

[7] Jiri Gaisler, Gaisler Research The LEON/ERC32 GNU Cross-Compiler System. [Online]. Available: http://www.gaisler.com/doc/leccs-1.2.2.pdf.

[8] Magma Design Automation, Inc. "A Full-Chip Hierarchical Design System," White Paper, 2002. [Online]. Available: http://www.magma-da.com/res/Pages/articles/HierarchicalDesign_Overview.pdf.

[9] James Wall and Anne Macdonald. (1993) "The NASA ASIC Guide," Internet Draft. [Online]. Available: http://nppp.jpl.nasa.gov/asic/title.page.html.

[10] Allen C. -H. Wu, Tsing Hua University. "Business Trends and Design Methodologies for IP Reuse". [Online]. Available: http://larc.ee.nthu.edu.tw/dtc/doc/soc_dream.pdf, unpublished.

[11] Henry Chang, Larry Cooke, Merrill Hunt, Grant Martin, Andrew McNelly, Lee Todd, Surviving the SOC Revolution: A Guide to Platform Based

Design. Reading, Kluwer Academic Publishers (1999).

[12] Aurangzeb Khan, Cadence Design Systems “Addressing Design Challenges at 130-Nanometer and Below: The Silicon Readiness Approach,” White paper, Fabless Semiconductor Association, October 2003. [Online]. Available: http://www.fsa.org/resources/whitepapers/Cadence10-03.pdf.

[13] “International Technology Roadmap for Semiconductors,” 2003 Edition, Reading. [Online]. Available: http://public.itrs.net/.

[14] Smith, M. J. S., “Application-Specific Integrated Circuits,” Reading. Addison-Wesley, Boston, MA, 1997.

[15] D. W. Bouldin, University of Tennessee. “Enhancing System-Level Education with Reusable Designs,” Proceedings of European Workshop on Microelectronics Education (EWME), Aix-en-Provence, France, pp. 5-8 (May 18, 2000). [Online]. Available: http://microsys6.engr.utk.edu/ece/ewme00.pdf.

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

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

[18] OAR Corporation, “Embedded with RTEMS,”. [Online]. Available: http://www.rtems.com/. unpublished.

[19] Donald W. Bouldin and Rishi R. Srivastava, "An Open System-on-Chip Platform for Education," Proceedings of 2004 European Workshop on Microelectronics Education (EWME), Lausanne, Switzerland, (April 15-16, 2004). [Online]. Available: http://microsys6.engr.utk.edu/ece/ewme04-soc.pdf.

[20] OpenCores.Org, [Online]. Available: http://www.opencores.com.

[21] Advanced Encryption Standard, Federal Information Processing Standards Publication 197 November 26, 2001. [Online]. Available: http://csrc.nist.gov/cryptval/aes/aesval.html

[22] Artisan Components, Inc. “Generator User Manual”.

[23] Envisia Technologies Inc. “Envisia Clock Tree Generator: User Manual”. [Online]. Available: http://www.es.lth.se/ugradcourses/cadsys/Ectgen.pdf.

[24] Cadence Design Systems. “Encounter Digital IC Design Platform”. [Online]. Available: http://www.cadence.com/products/digital_ic/index.aspx. unpublished.

[25] University of California San Diego. Place and Route with SoC Encounter. [Online]. Available: http://vlsicad.ucsd.edu/courses/ece260b-w04/Lab2/Lab2.php. unpublished.

Vita

Rishi R. Srivastava was born in Bhopal, India. He received his Bachelor of Engineering in Electronics and Telecommunication engineering from Pt. Ravi Shankar Shukla University Raipur, India. He joined the University of Tennessee, Knoxville as a Graduate student (Masters program) in August 2001. Subsequently he has been doing his research under the guidance of Prof. Donald W. Bouldin. His research interests are in field of ASIC Design and Verification. He plans to graduate with a Master's degree in Electrical engineering in Aug 2004.