5-2005

Using Platform Express for System-on-Chip Design

Mardavsinh Harisinh Wala
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Mardavsinh Harisinh Wala entitled "Using Platform Express for System-on-Chip Design." 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, Mohammed Ferdjallah

Accepted for the Council:
Dixie L. Thompson

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 Mardavsinh Harisinh Wala entitled "*Using Platform Express for System-on-Chip Design*." 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

Mohammed Ferdjallah

Acceptance for the Council:

Anne Mayhew

Vice Chancellor and Dean of Graduate Studies

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

Using *Platform Express* for System-on-Chip Design

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

Mardavsinh Harisinh Wala
May 2005

To my parents and all my teachers

ACKNOWLEDGEMENTS

Silent gratitude isn't much use to anyone.
—Gladys Bertha Stern (1890-1973).

I would like to convey my sincere thanks to my academic and research advisor, Dr. Don Bouldin, for allowing me to pursue this research opportunity under his able guidance. I am thrilled that he let me lead the way here at UT, by being his first student to ever use *Platform Express* for SoC design. I gratefully acknowledge Dr. Gregory Peterson and Dr. Mohammed Ferdjallah for taking interest and serving on my committee.

Without the Mentor Graphics' excellent tech support team, I would never have been able to finish what I started. Special thanks to Mr. Tomas Thoresen for looking into each of my Service Requests and for patiently answering all my queries. Many thanks to Jiri "Mr. Leon" Gaisler of Gaisler Research, for the 'how to' on building the sparc-elf-gcc compiler for Solaris and to our System Administer, Mr. Matt Disney, for help in installing the build after finding several workarounds, suitable to our VLSI lab machines. Thanks also to Mr. Klaus Knopper for making *Knoppix*—Linux-on-a-CD.

I am very grateful to Mr. Mark Alexander and the Vice Chancellor's Office for the Division of Student Affairs, for providing financial support in the form of Graduate Technology Assistantship during my course of study. I am also grateful to the Associate Dean of Students, Mr. J. J. Brown and the Dean of Students, Dr. Maxine Thompson for offering me the GA opportunity during my final year. I also thank Dean Brown for his flexibility and understanding while I was working towards finishing this thesis. I consider myself to be very fortunate for having worked for all these outstanding people.

I express my special gratitude to my parents, Mr. Harish Wala and Mrs. Ranjan Wala, for giving me the freedom to choose and for believing in my choices. Without their moral, spiritual and lots of financial(!) support, I would not have come this far in life. A big 'Thank you!' to all my uncles, aunts and cousins—in India and in the United States—for making me feel at home every time I visited them.

I wish to thank my colleagues and friends in Knoxville for their friendship (and in no particular order): Ashwin Balakrishnan, Karthik Kirubakaran, Jeongmin Jeon, Rajgopal, Tanvir Alam, Wasima Khan and Yongpin Han, for their valuable tips in some of my classes. Chaya Chandrasekaran, Barrett Bogue, Jill Patterson, Catherine Elliott and Patrick Ladd, for being excellent work mates at the Dean of Students office. Faezeh Jalali, for free tickets to the Clarence Brown Theatre and for occasionally lending her wonderful 1992 Chrysler New Yorker. Rohan Thomas and Venkatesh Bhaskaran, for having me as their virtual roommate and for all the field trips in and around Knoxville. Ravi Aggarwal, Raghvendra Hegde, Gaurav Baone, Ikramuddin Shaik, Gagan Rajpal, Shraddha Deodhar-Oak, Manisha Gautam and Niti Sharan, for all the dinners and movies. Bhanuprasad Rekapalli, for being my personal trainer and gym buddy. Peter Koffman and Chaya, for getting me in*VOL*ved in UTHSMUN-2k5 and for inviting me to 'chill-n-grill' at the home and away football games.

Lastly, I want to thank Ashita Dave, for all her support and encouragement and for always being there for me.

ABSTRACT

The advent of nanoscale technology brings with it an increase in system complexity with integrated circuit transistor numbers reaching hundreds of millions. Systems-on-chip are attaining a level of complexity where design turn-around times are a major factor. Reusing existing intellectual property blocks that are already verified for functionality could help minimize the design time and increase system reliability. This allows the designers to focus on more important product design aspects. Platform-based design is an effective method to deal with the increasing pressure on time-to-market. The approach also provides a practical solution to reduce the design and manufacturing costs.

This thesis is a result of the of the ongoing *Volunteer SoC* project at the University of Tennessee and in this, we explore the possibility of employing the *Platform Express (PX)* tool for designing SoCs. The *PX* application enables system designers to rapidly build and verify SoC design concepts. The tool also promotes Intellectual Property (IP) integration within the built-in *PX* libraries. The tool utilizes XML for describing the IP data, which allows smooth integration of IP into a single design from many different sources.

We have followed the complete IP integration flow and have successfully installed a component into the tool's library and have also generated a system design using the same IP.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1

INTRODUCTION

There is nothing more difficult to take in hand, more perilous to conduct or more uncertain in its success than to take the lead in the introduction of a new order of things.

—Niccolo Machiavelli (1469-1527), *The Prince*.

With continued advancement in silicon process technologies, the data density on integrated circuit chips is growing by leaps and bounds in accordance to what is widely known as Moore's Law [1]—first stated by Intel founder Gordon Moore in 1965. During recent years, the sudden increase in gate count (*Figure 1.1-1*) and the steady demand for mobile, portable, high-speed gadgets has resulted in a large market for electronic consumables in the form of cell phones, PDAs, digital camcorders, personal CD/DVD players, video game consoles and the like. These factors have entailed chip designers to design exceedingly complex chips.

1.1 THE SOC DESIGN CHALLENGE

System-on-a-Chip (SoC) design refers to implementing an entire electronics sub-system on a single IC. Smaller feature sizes makes adding extra circuitry on a silicon die more cost-effective. Chips manufactured with these dies consist of one (or more) processor(s), a high-performance bus, custom logic (digital and analog), memory devices and peripherals along with software code. SoC design requires developing innovative techniques to tackle design complexity and its related risks. The semiconductor industry addresses these challenges by adopting new design schemes and by using improved electronic design automation (EDA) tools.

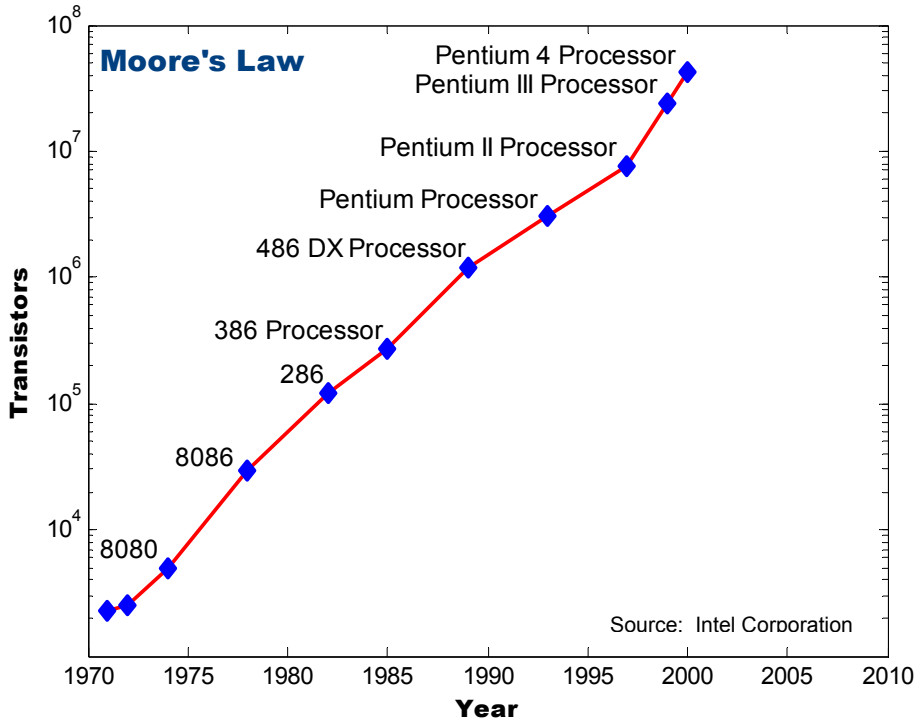


Figure 1.1-1: Moore's Law depicting increasing transistor complexity with advancement in semiconductor manufacturing process technology

The relentless progress in the semiconductor manufacturing process witnesses a continual reduction in the integrated circuit (IC) feature sizes for wires, transistors and contacts. The successive advancement to a smaller feature size requires altering thecomplete design and manufacture flow to accommodate the new physical effects associated with the decrease in size. Since the design methodologies and tools do not progress as swiftly as the process technology changes, there always exists a productivity gap, shown in *Figure 1.1-2*. The increased design complexity and slowly evolving design methodologies prevent the silicon design teams from exploiting the full potential for SoC design that is allowed by the advanced process technology.

Many corporations are now exploring a platform-based design (PBD) approach to address the growing complexity of SoC design [2]. Platform-based design methodology defines a robust, flexible design around a stable core platform, connected by means of

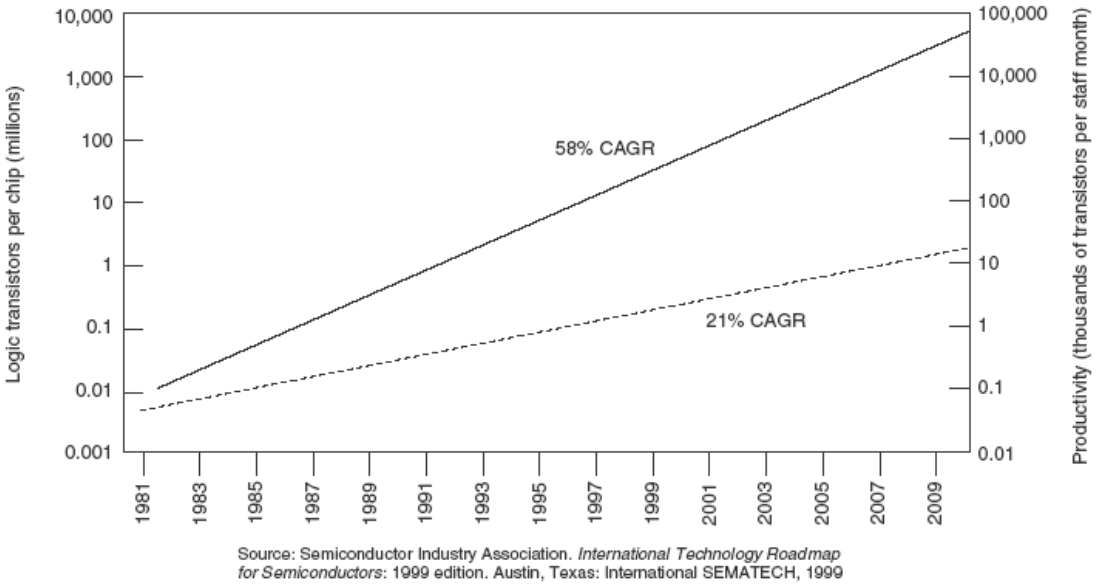


Figure 1.1-2: Design Productivity Gap – Difference between the number of physical transistors available on a chip (solid curve) and the number of transistors that can be handled by current design tools(dashed curve)

standard buses, which have been optimized for use with the processor core. Once a platform is created, design teams can produce a new SoC quickly using mostly existing, pre-verified intellectual property (IP) blocks or virtual components (VC) and hence complete the design without requiring much new circuitry or software. This approach helps reducing the time-to-market drastically and increasing the system reliability to a large extent.

For a PBD methodology to be robust, it must be able to adjust to design re-spins and enhancements without extra change to the base platform elements. The availability of a library of components, which include specialized microprocessors, digital signal processing IP cores and may be some internally generated, custom application specific integrated circuits, would help designers narrow down their design choices. Furthermore, the ability to perform several design iterations in a short period of time would allow them to determine a suitable configurable hardware platform.

1.2 MOTIVATION

There may be several high-end EDA tools available for research purposes to a single educational institution but what is generally missing is the availability of an internally designed library of custom IP blocks, which may be used to build a large SoC and its derivatives using those tools. The graduate program in the Electrical and Computer Engineering department at the University of Tennessee [3] spanning four semesters, addresses this issue by offering courses intended to equip individual students with the understanding of design *for* reuse and a team of students with the understanding of design *with* reuse.

In the spring of 2003, the graduate class consisting of sixteen students was split into groups of twos and fours and each group was assigned the task to simulate, synthesize and test, a single IP core—either internally generated or obtained for free. The intention was to verify each IP block for functionality before integrating it with the open core *Volunteer SoC* platform [4].

When the SoC platform was completed in August 2004, the next step in the design process was to raise the level of abstraction through which the platform designers integrated the IP blocks. This way the designers could work directly at the component level rather than at the VHDL-entry level and they could also rapidly identify, select and integrate (or remove) the required IP block into (or from) their design. *Figure 1.2-1* and *Figure 1.2-2* show the difference between the two design flows.

The idea thus conceived, was the major motivational factor for taking the *Volunteer SoC* project to the next level and selecting Platform Express™ [5], an EDA tool by Mentor Graphics®, which our department had acquired in 2002 for this purpose. The PX environment, as shown in *Figure 1.2-3,* presents users with a graphical interface and allows them to enter designs as block diagrams by selecting processors, memories and peripherals from a library of components. The tool includes a memory map display for assessing address space. Upon successful completion of graphical entry, PX automati

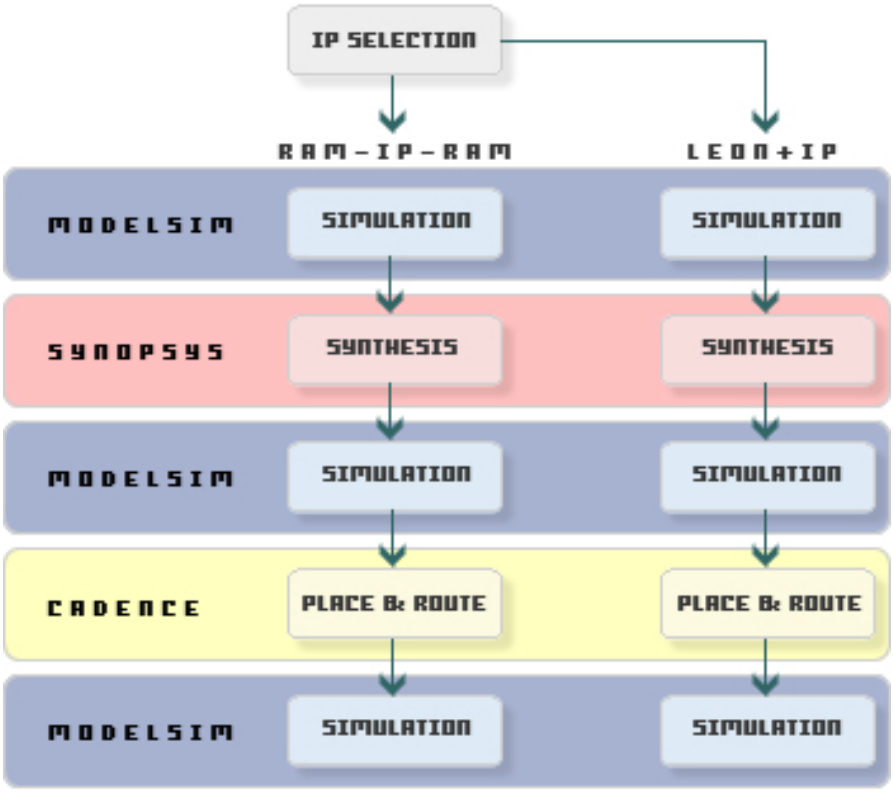


Figure 1.2-1: Design flow for *Volunteer SoC*. The selected IP is already verified for correct functionality.

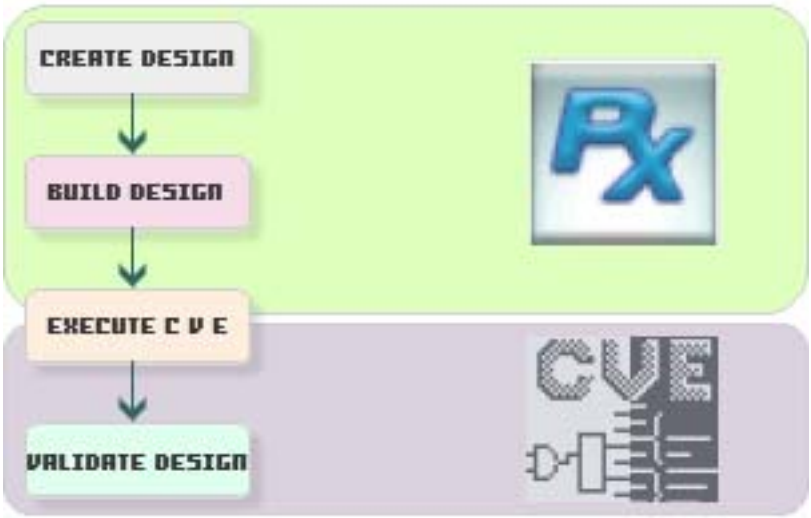


Figure 1.2-2: Design flow using Platform Express™. The design is created from the pre-installed IP components into the Platform Express™ library.

Design Editor

Component Browser

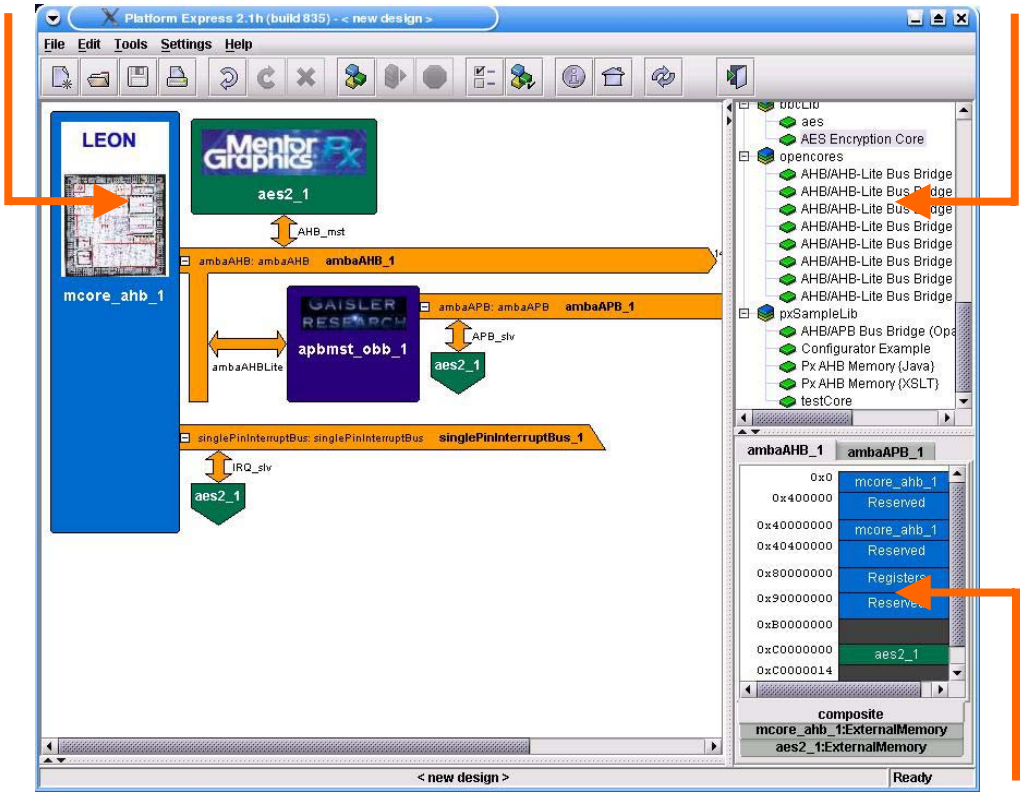


Figure 1.2-3: The Platform Express™ Environment

Memory Map Pane

cally generates the design along with software to run on the design and a test bench to drive it. The environment then calls upon verification tools and rapidly generates the otherwise time-consuming verification scripts and diagnostic code for each peripheral and memory component in the generated design. The designers have an option of proceeding on to hardware/software co-verification using the Seamless® co-verification environment or using the ModelSim® simulation tool to perform RT level hardware verification. Such a PBD methodology supported with *Platform Express* generates hardware and software designs together with the custom execution environment required to verify the designs. The latest release of *Platform Express* at the time of this project was version 2.1h.

1.3 THESIS GOALS

This thesis is intended to demonstrate the use of the *Platform Express* environment for developing system platforms for SoC designs. It is also expected to serve as a guide to platform-based SoC design using PX.

The goals of this project were:

- To install a pre-verified IP core into the PX component library and follow the design flow described previously in *Figure 1.2-2*
- To prepare an instructional write-up explaining the complete IP integration and platform building process

1.4 PROJECT COMPONENTS

Since the baseline platform is kept in the public domain and also given the fact that in an academic research environment cost is always a major constraint, only freely available IP cores have been used.

1.4.1 Leon2 Processor IP Core

Initially developed by Jiri Gaisler during his work at the European Space Agency (ESA), the 32-bit SPARC compatible Leon2 processor [6] is now maintained under contract by Gaisler Research in Sweden. ESA promotes development of SoC designs using the SPARC architecture; therefore, Leon2 is available for free download under GNU Lesser General Public License (LGPL) and GNU General Public License (GPL).

The Leon2 processor, shown in *Figure 1.4-1* has the following noteworthy features:

- SPARC V8 compliant integer unit with 5-stage pipeline
- Hardware multiply, divide and MAC units
- Separate instruction and data cache (Harvard architecture)
- AMBA 2.0 AHB and APB on-chip buses
- 8/16/32-bits memory controller for external PROM and SRAM
- On-chip peripherals such as uarts, timers, interrupt controller and 16-bit I/O port

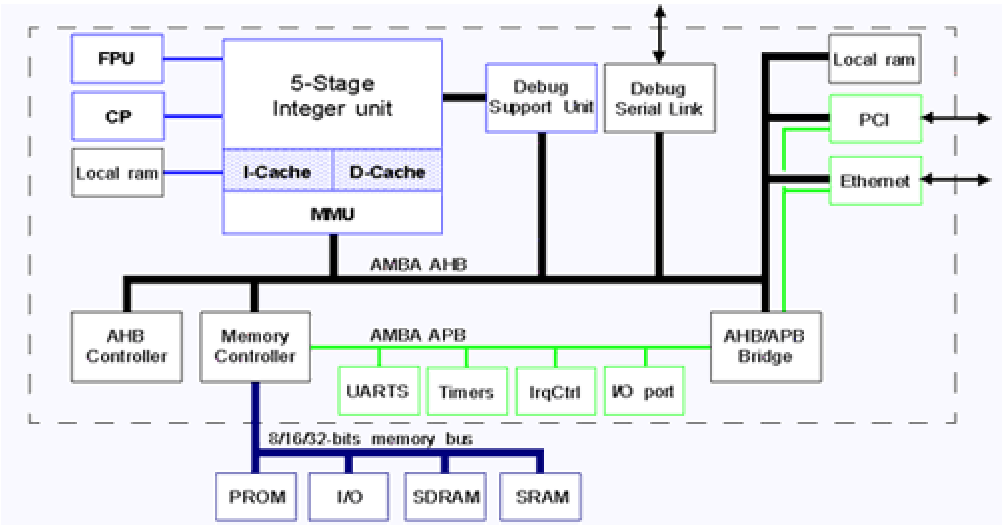


Figure 1.4-1: Leon2 Architecture

1.4.2 The AMBA Bus Interface

The Advanced Microcontroller Bus Architecture (AMBA) [7] is ARM's no-cost, open specification, which defines an on-chip communications standard for designing high-performance embedded microcontrollers. The AMBA specification has become a de facto standard for the semiconductor industry, and has been adopted by more than 90% of ARM's partners and a number of IP providers. The specification has been successfully implemented in several ASIC designs. Since the AMBA interface is processor and technology independent, it enhances the reusability of peripheral and system components across a wide range of applications. Three distinct buses are defined within the AMBA specification:

- **Advanced High-performance Bus (AHB)**

  The AMBA AHB is suited 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. AHB is also specified to ensure ease of use in an efficient design flow using synthesis and automated test techniques.

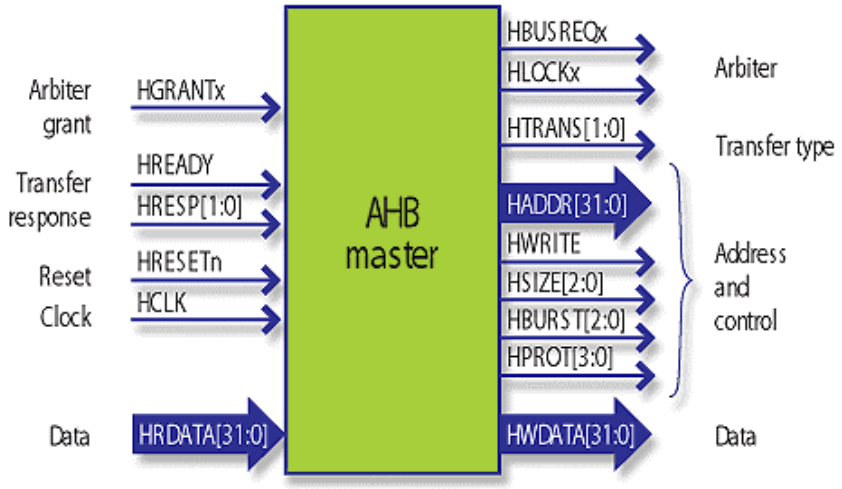


Figure 1.4-2: The Advanced High-performance Bus Signals [8]

The AHB is recommended for all new designs, not only because it provides a higher bandwidth solution, but also because the single-clock-edge protocol results in a smoother integration with design automation tools used during a typical ASIC development. *Figure 1.4-2* illustrates the bus signals for AHB.

- **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. ASB also supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macrocell functions.

A full AHB/ASB interface is recommended for bus masters, on-chip memory blocks, external memory interfaces, high-bandwidth peripherals with FIFO interfaces and DMA slave peripherals. (Note that ASB is not implemented on Leon2 and therefore not used in our project.)

- **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.

A simple AHB/APB interface is suggested for register-mapped slave devices (shown in *Figure 1.5-1*) and low power interfaces where clocks cannot be globally routed and grouping narrow-bus peripherals to avoid loading the bus.

1.5 THESIS ORGANIZATION

Chapter 2, *"Background"*, discusses the factors that led to the emergence and adoption of platform-based design approach as a preferred method for designing complex SoCs. It also acquaints the reader with the several definitions of *platform*—both commercial and academic, and explains the role of IP in helping improve existing platforms. It then provides an overview of the ongoing *Volunteer SoC* project at the University of Tennessee's Microelectronics Systems Laboratory.

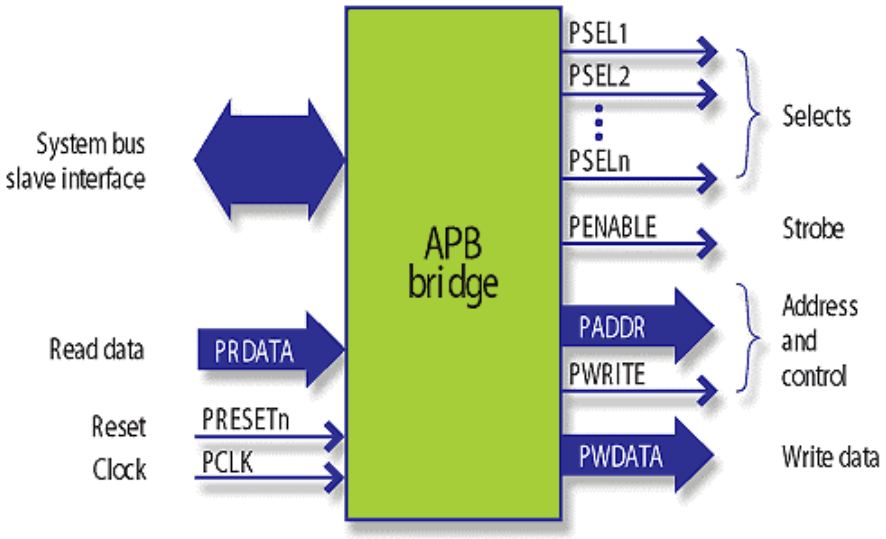


Figure 1.5-1: The Advanced Peripheral Bus Bridge Signals [8]

Chapter 3, *"Methodology"*, outlines the platform-based design flow and describes the AES IP core, available from OpenCores.org, to be integrated with our baseline platform. It also describes the *Platform Express* EDA tool and explains the steps needed to be taken before the IP component can be integrated for use with the *Platform Express* environment.

Chapter 4, *"Implementation"*, describes the process of integration of the AMBA-compliant IP core, as a component inside the *Platform Express* library created by the IP integrator. It also describes using the installed component to build a test design from the point of view of a system designer. The *Seamless CVE* interface is used to validate the component for functionality using the *ModelSim* application.

Lastly, chapter 5, *"Conclusions"*, summarizes and concludes the thesis with recommendations for future enhancements for the *Volunteer SoC* platform.

2

BACKGROUND

If you try to build everything from scratch, you'll never get to the market.

-- Ronnie Vasishta, LSI Logic.

Smaller integrated circuit (IC) feature sizes, increased time-to-market (TTM) pressures, coupled with prohibitive costs of ownership for IC masks have pushed the semiconductor industry to look for design alternatives that use an existing base of components and architectures. The pursuit for flexible yet economically feasible design approach along with the development towards higher level of design abstractions has led to the emergence of a platform-based design methodology.

2.1 FROM SCHEMATICS TO SOCS

As depicted in *Figure 2.1-1*, the semiconductor industry has come a long way from considering schematics as state-of-the-art for system implementation and then adopting a register-transfer level (RTL) design entry mechanism with the advent of hardware description languages (HDLs). This transition proved advantageous because it was possible to build and test, larger, more complex designs in comparatively less amount of time using RTL HDL descriptions instead of schematics. The availability of a broad range of simulation and verification tools to help ease the RTL HDL verification also contributed to the wide acceptance and success of this methodology [9].

In the current scenario when it is typical for an SoC to contain tens of millions of gates combined in processor cores, on-chip interconnects, specialized DSP units and analog components, it is a challenging task for the chip design teams to completely design all the components from scratch. Moreover, they have a product launch deadline to meet. Circumstances such as these have resulted in a trend towards increased IP

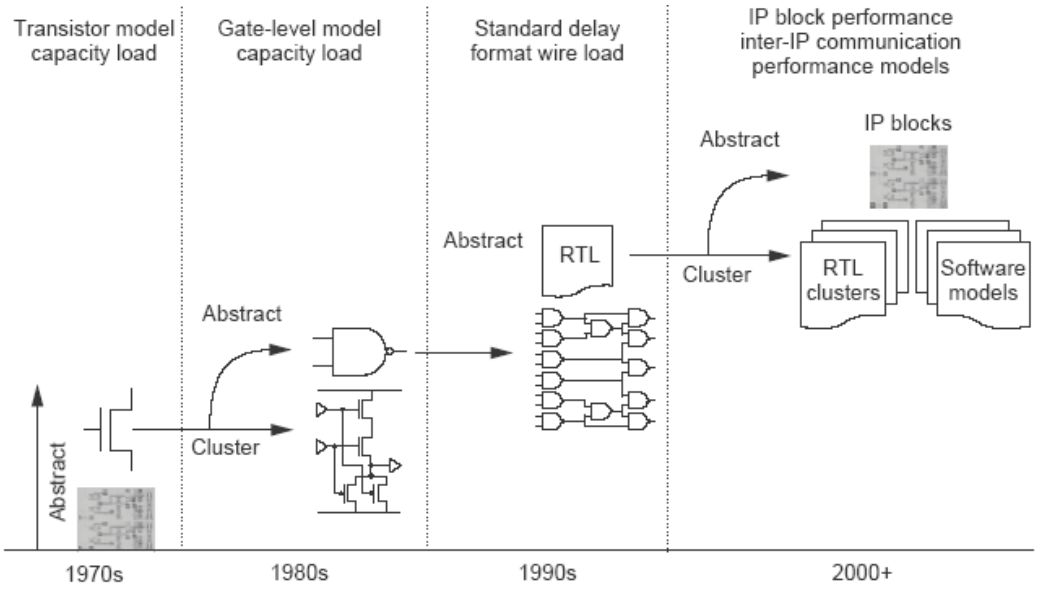


Figure 2.1-1: From Schematics to SoCs [10]

reuse, which requires little or no modification on the reusable IP blocks. A major benefit of this approach is that properly defined IP blocks can be reused across multiple designs. *Figure 2.1-2* illustrates the role of IP reuse methodology in closing the design productivity gap.

There are however, some setbacks associated with the IP reuse methodology. For example, designing a system having multiple IP blocks obtained from different sources, would call for extensive training for the design team members in each of the specialized hardware/software IP protocols. Additionally, some components may also require extended licensing negotiations. Meeting all these requirements could result in a few months of dead time — before designers even get started on the project!

The platform-based design approach enables design teams to rapidly integrate multiple functionalities on a single chip from a library of specialized IP using standard interfaces. More importantly, by providing the designers up to 90% of the required hardware and software in an integrated SoC platform, it allows them to focus design resources on differentiating their product [11].

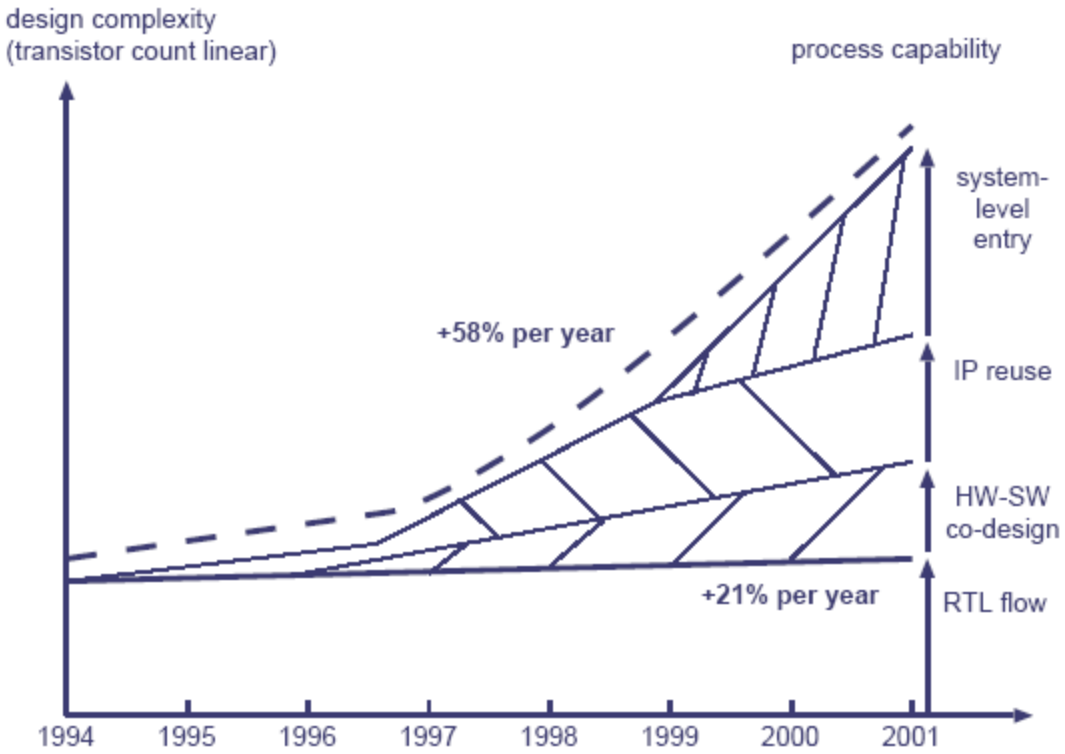


Figure 2.1-2: Bridging the Design Productivity Gap [12]

On the whole, a platform-based SoC design approach can be translated into four major gains:

- **Shorter time-to-market:** The intentional and extensive reuse of preexisting known verified IP — when designing platform-based solutions — permits design cycles of six months or less [13] and reduces overall design risk to a great extent [14]. The widespread reuse of standardized IP blocks and software eliminates the need for training design engineers in discrete protocols, thus enabling companies to introduce their product in the market on time.
- **Reduced development costs:** Employing a PBD approach significantly limits the number of third-party virtual components to be integrated on the SoC. This translates to huge savings in development costs.
- **Minimal verification time:** PBD methodology facilitates hardware/software co-simulation by providing the integrated SoC platform as a total development environment for system-level functional verification [15]. Considerable savings in verification time can be realized by employing this technique.

- **Lower power consumption:** Since all the separate components on an integrated platform are optimized to minimize gate-level power dissipation and to lower false signal fluctuations, the overall power consumption in a platform-based SoC is substantially reduced [11].

2.2 WHAT IS A PLATFORM?

There have been several attempts by the semiconductor industry to define the term – *platform*. The Platform-based Design Development Working Group (PBD DWG) of the Virtual Socket Interface Alliance (VSIA) – formed with a vision to standardize platform engineering for SoC-based systems – defines a **platform** as a "library of virtual components and an architectural framework consisting of a set of integrated and pre-qualified software and hardware IP blocks, models, EDA and software tools, libraries and methodology to support rapid product development through architectural exploration, integration and verification". The study group extends this definition a little further to explain **platform-based design** as an "integration-oriented designed approach emphasizing systematic reuse, for developing complex products based upon platforms, intended to reduce development risks, costs and time to market" [16].

Despite efforts by the PBD DWG, the definition of platform is still unclear because various semiconductor disciplines prefer their own version for the meaning of platform. With tool companies, SoC providers and manufacturing companies offering platform-based solutions depending on their area of technical expertise [17], it is very crucial for the customer to understand how each of them defines a platform. *Figure 2.2-1* lists some of the definitions provided by the industry and by academia.

In the sub-sections that follow, we enumerate some of the platform postulates put forward by Bob Altizer and his VSIA PBD Study Group [16]. We also discuss some of the platform types characterized by Frank Schirrmeister of Cadence Labs [18].

"An integration platform is a reuse mix-n-match environment designed specifically to target an application domain. The domain is selected based on market objectives and is focused to yield a high probability of reuse over a period of time." – Cadence White Paper, *The IP Reuse Evolution*.
"A platform is a collection of assets, which can be used to leverage reuse and rapidly develop new products. At a minimum, it defines the operating environment, high level product architecture for all products developed based on this platform, and set of development policies for extending the platform and developing point products from the platform." – Motorola PCS/ATSO, *Reuse Lifecycle Model-v1.0*.
"An embedded system platform is an architectural framework for rapid integration of embedded SoC-based designs, consisting of a set of pre-qualified software and hardware IP blocks and a methodology to support rapid architectural exploration, integration, and verification." – Frank Pospiech, Alcatel.
"We define platform-based design as the creation of a stable microprocessor-based architecture that can be rapidly extended, customized for a range of applications and delivered to customers for quick deployment." – Jean-Marc Chateau, STMicroelectronics.
"A platform is, in general, an abstraction that covers a number of possible refinements into a lower level. For every platform, there is a view that is used to map the upper layers of abstraction into the platform and a view that is used to define the class of lower level abstractions implied by the platform." – Alberto Sangiovanni-Vincentelli, University of California at Berkeley.

Figure 2.2-1: Platform Definitions

2.2.1 Platform Postulates

The following set of postulates were developed by the VSIA PBD DWG [16] to help recognize, appreciate and understand the finer points of PBD.

- A platform can be viewed as an integration-ready ensemble of hardware and software components that would act as a starting point for future derivative product designs.
- To be successful – in addition to pre-verified and pre-defined platforms – PBD approach depends heavily on the availability of product differentiating IP components, an integration-oriented design flow along with the accessibility to support on issues regarding tools usage, applications and systems.
- From the economic standpoint, PBD can help increase profits since it allows systematic and planned IP reuse, improves successive product capabilities and quality and reduces the overall TTM drastically.
- Finally, the profits should be large enough to rationalize the investment in platform development and procurement of special IP blocks, integration tools and support services.

2.2.2 Platform Types

Depending on the suitability to a particular specification and the availability of customization options, platforms can be classified into four categories [18].

- **Full-application Platforms:** These platforms allow designers to develop full applications on top of hardware-software architectures. To facilitate users in derivative-product design, full-application platforms generally contain a library of hardware modules, with each module having multiple design schemes. Designers can choose from this broad range of available modules to build complex
- **Processor-centric Platforms:** These platforms concentrate more on specific processor cores and also focus on the software access to the processor. Designers oftentimes require additional application-specific hardware blocks and in some cases, a different real-time operating system (RTOS), to achieve full applications. Improv Jazz and ST StarCore best illustrate this platform type.

- **Communication-centric Platforms:** This design approach offers consumers an optimized, customizable communications platform, suitable for a specific application. Here again the derivative-product designer is required to include components to obtain a complete application. Sonics and PalmChip architectures are the prominent examples.

- **Fully Programmable Platforms:** These platforms are similar to full-application and processor-centric platforms except that these also include embedded reconfigurable logic. The addition of programmable logic enables designers to customize the platform with both hardware and software. Examples include Triscend, Altera Excalibur and Xilinx P-FPGA platforms. *Table 2.2-1* lists some commercially available platform cores and designs.

Table 2.2-1: Some of the Many Commercially Available Reference Designs and Platforms

CHIPMAKER	REFERENCE DESIGN or PLATFORM	END-USER MARKET	SYSTEM CUSTOMERS
TI	OMAP	Cell phone handsets	Ericsson, Nokia, Sony Ericsson
Philips	Velocity	Wireless, Consumer	NEC-Matshushiti, HP
	nExperia	Home Network Gateways	AOL
Qualcomm	Binary Routine Environment for Wireless (BREW)	Cell phones, Cellular infrastructure	Verizon, Sony
Intel	Xscale	Cell phone handsets, PDA	HP-Compaq, Toshiba
	PC Motherboards	Personal computers	Many
Portal Player	Digital Media Player	MP3 devices	Apple

Source: International Business Strategies Inc.

For platform-based designs employing the *Volunteer SoC* platform, the Leon processor will be common to all derivative designs and therefore, our platform—due to its focus on the processor core—is *processor-centric.* An *application-oriented* platform can be realized by adding specialized IP cores to extend the capabilities of our *processor-centric* baseline platform.

3

METHODOLOGY

… with proper design, the features come cheaply. This approach is arduous, but continues to succeed.

—Dennis Ritchie, AT&T Bell Labs.

Our design methodology covers design aspects ranging from specification to implementation. While the discussion of requirements for our open SoC platform presented in the previous chapter fixes the platform specifications for our design, this chapter explains the subsequent steps needed to be taken in the PBD flow proposed by Kuetzer, et al. [19].

3.1 PLATFORM-BASED DESIGN FLOW

The PBD flow comprises of four phases: phase **1** deals with identifying the function that the system will eventually implement. Phase **2** involves identifying the system architectures through which the functionality can be implemented. Phase **3** involves selecting the optimal architecture from the set of previously identified architectures deemed suitable for system implementation, as well as selecting the system components that effectively meet the necessary specifications. Finally, phase **4** focuses on realizing the implementation of the system function on the chosen architecture through hardware synthesis and software assembly of system components. *Figure 3.2-1* illustrates the design flow to be followed while designing a platform-based system.

3.2 ENHANCING THE *VOLUNTEER SOC* PLATFORM

We will now describe the approach to add functionality to the *Volunteer SoC* platform for implementing a desired application. Since the platform is kept in the public domain and due to the lack of design guidelines for packaging and incorporating IP for reuse with the *Volunteer SoC*, this thesis is a result of our effort to outline a procedure for adding specialized IP blocks to enhance the capabilities of our platform.

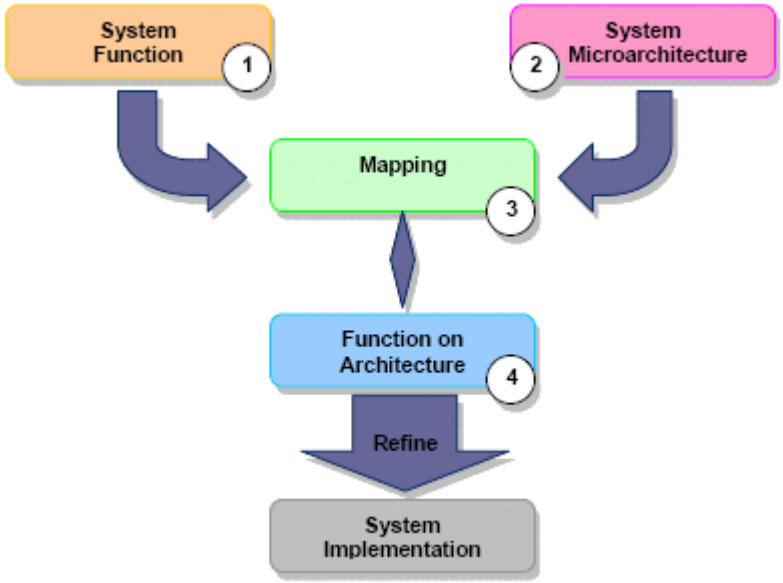


Figure 3.2-1: Platform-based Design Flow [19]

Table 3.2-1: AES (Rijndael) Encryption Core integrated with Platform Express Release Information

ITEM	DESCRIPTION
Version	1.0
Release Date	November 2004

As an illustration, we will use *Platform Express* to include the AES encryption IP core, available from OpenCores.org, to add encryption functionality to the *Volunteer SoC* platform. The following sections introduce the AES IP core and the *Platform Express* tool and explain the process of making the AES core AMBA-compliant before being added as a peripheral to the Leon CPU core.

3.2.1 The AES (Rijndael) IP Core

Release Information

Table 3.2-1 provides information about this release of the AES encryption core when integrated with *Platform Express*.

General Description

The AES encryption core available from OpenCores.org implements the Rijndael standard with a 128-bit key expansion module. In addition to the key expansion module, the core also consists of an initial permutation module, a round permutation module and a final permutation module. The round permutation module loops internally to perform ten iterations on the 128-bit key and data inputs. *Figure 3.2-2* illustrates the overall architecture of the AES encryption core.

The core requires a key and a plain text input at the start of each encryption sequence. The start is indicated by asserting the **ld** pin high. Upon encryption the **done** pin is asserted high for one clock cycle. The core completes a single encryption sequence in twelve clock cycles (ten for the round permutation module and one each for the initial and final permutation modules). The user may choose to ignore the **done** output and can opt to time the completion of encryption sequence externally. *Figure 3.2-3* shows the hierarchy structure for the AES (Rijndael) encryption core Verilog source files. A thorough description of the Rijndael standard is provided in this paper by Daemen, et al. [20].

Before adding the AES core to the Leon CPU as a peripheral, the AES core had been modified by interfacing it with an input RAM to store the 128-bit key and 128-bit data and an output RAM to store the 128-bit encrypted data. *Figure 3.2-4* shows the *RAM-IP-RAM* block diagram.

3.2.2 The Platform Express Environment

Besides being one of the seven founding members of the Structure for Packaging, Integrating and Re-using IP within Tool-flows (SPIRIT) Consortium [21], Mentor Graphics Corporation is also one of the steering committee members of the conglomerate. This group comprising of leading EDA vendors (Mentor Graphics, Cadence Design Systems and Synopsys), a leading star IP provider (ARM Ltd) and

leading SoC integrators and manufacturers (STMicroelectronics and Philips), aims at setting stan-

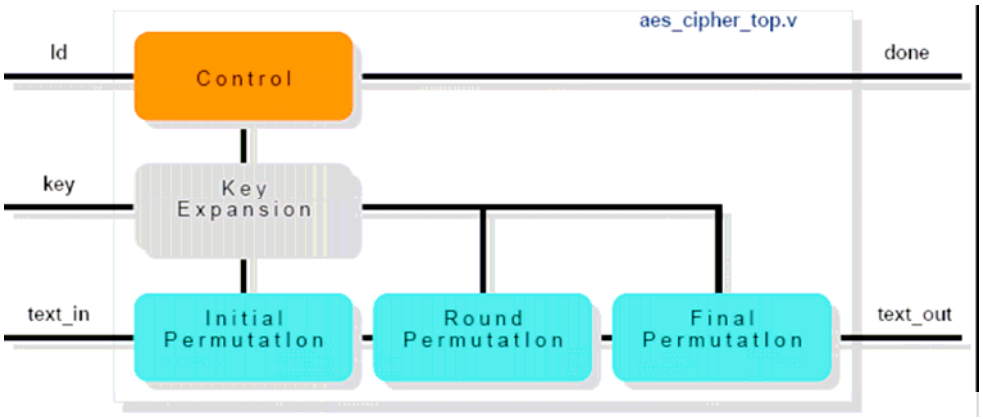


Figure 3.2-2: AES Encryption Core Architecture

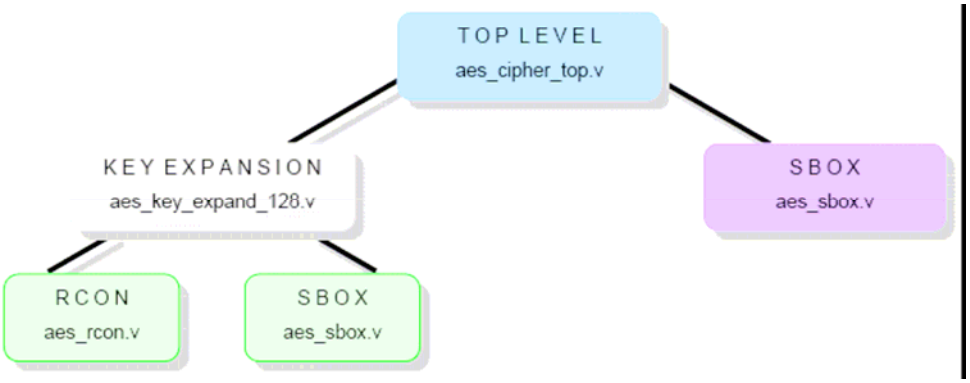


Figure 3.2-3: AES Encryption Core File Hierarchy

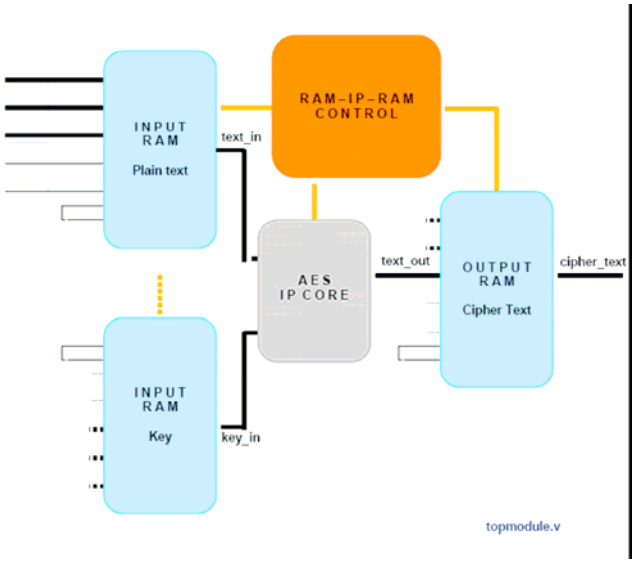


Figure 3.2-4: AES Core Interfaced with Input and Output RAM Blocks

-dards for IP description and IP packaging, to enable an efficient and a cost-effective IP integration process with tools and IP from multiple vendors. *Figure 3.2-5* depicts the SPIRIT schema and generator interface.

Platform Express is one such SPIRIT-compliant EDA tool, which allows the system designer to quickly build a system design using the components that the IP integrators have created from their own hardware designs. The *PX* interface presents the facility of selecting a platform core (such as ARM926, ARM966) for use as a design foundation. The platform core components are available via the libraries from licensed component library developers, in addition to the various demonstration libraries that *PX* ships with. Most platform cores are provided as 'open source', however source code for proprietary components is not supplied and they are available only for simulation purposes.

The *PX* application allows creating and implementing user-defined libraries and provides a built-in IP metadata generation interface—*PxEdit*—to realize that objective. The IP metadata describes the characteristics of the IP components; this includes information about invoking simulation and verification environment that the component requires, and allows setting up and logging of design configuration. *Platform Express* uses the open source Extensible Markup Language (XML) as the metadata language (also a SPIRIT standard) to describe the IP components for integration with the *PX* component libraries. The XML metadata, in association with the *PX* application also initiates other code written in Java, VHDL and Verilog that allow components to function in a design.

The *PX* application speeds up design creation by presenting the significant design elements in detail, within the *PX* application. The *PX* Design Editor is context-aware and allows immediate configuration. Once the design is created, *PX* provides tools for automating the build process. The resulting build files also include ones that could be used for validation with *Seamless CVE*. *PX* offers automatic bus decoding and automatic bus and interrupt-bridging.

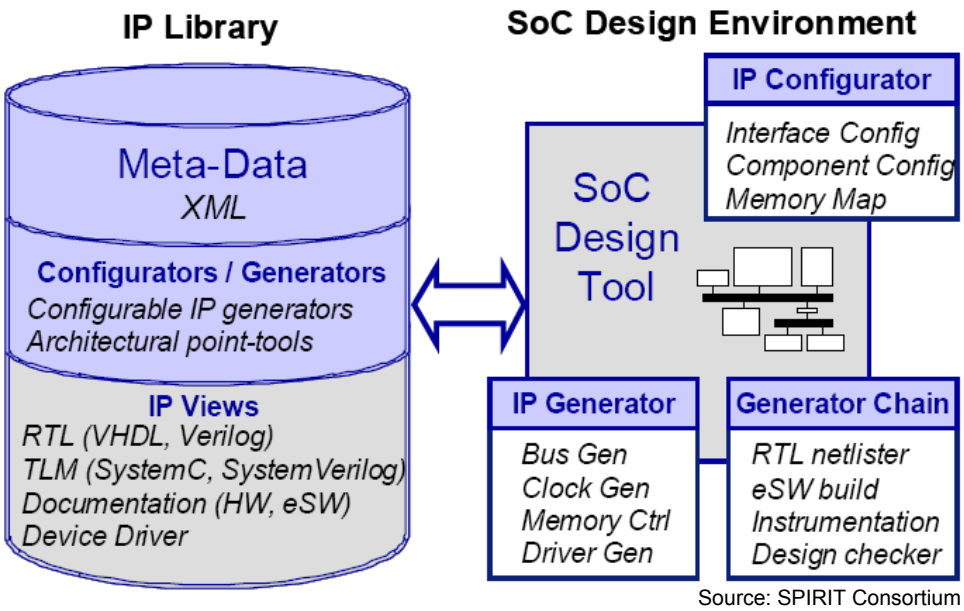


Figure 3.2-5: SPIRIT Schema and Generator Interface

Thus, when used in conjunction with *Seamless CVE*, *PX* not only presents the vital functionality for designing and building complex SoC subsystems but also provides access to software debugging tools, such as *XRAY Debugger*, and hardware logic simulation tools, such as *ModelSim* and *NCSim*

3.2.3 Defining the Bus Interface

Building a platform around a standard bus architecture allows flexibility and ease of extension. Since Leon has adopted the AMBA AHB and APB as an on-chip bus standard, it was natural to opt for the same standard for integrating our IP core. A wrapper was written in VHDL to package the *RAM-IP-RAM* module for effortless integration.

The AES component is an AHB master connected as a peripheral. It communicates with the Leon processor via the AHB and uses a peripheral bus bridge for data transfer between Leon and itself. *Table 3.2-2* lists the bus interface signals of the top entity aes.vhd.

Table 3.2-2: AES Core Bus Interface Signals

SIGNAL NAME	TYPE	BUS TYPE	DESCRIPTION
HCLK	Input	AHB	*Bus Clock:* Times all transfers. All signal timings are related to the rising edge of HCLK.
HRESETN	Input	AHB	*Reset:* Bus reset signal used to reset the system and the bus. This is the only active LOW signal.
HGRANT	Input	AHB	*Bus Grant:* Indicates that bus master X is currently the highest priority master. Ownership of the address/control signals changes at the end of a transfer when HREADYi is HIGH, so the master gets access to the bus when both HREADYi and HGRANTx are HIGH.
HREADYi	Input	AHB	*Transfer Done*: Indicates that a transfer is finished on the bus when HIGH. The signal may be driven LOW to extend a transfer. HREADYi is the HREADY input to a slave.
HRESP [1:0]	Input	AHB	*Transfer Response:* Provides information on the status of the transfer.
HRDATA [31:0]	Input	AHB	*Read Data Bus*: Used to transfer data from bus slaves to bus master during read operation.
HREADYo	Output	AHB	*Transfer Done*: Indicates that a transfer is finished on the bus when HIGH. The signal may be driven LOW to extend a transfer. HREADYo is the HREADY input from a slave.
HBUSREQ	Output	AHB	*Bus Request:* Indicates to the bus arbiter that a bus master X requires the bus. A maximum number of 16 bus masters are possible in the system.
HTRANS [1:0]	Output	AHB	*Transfer Type:* Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.
HADDR [31:0]	Output	AHB	*Address Bus:* The 32-bit system address bus.
HWRITE	Output	AHB	*Transfer Direction:* Indicates a write transfer when HIGH and a read transfer when LOW.
HSIZE [2:0]	Output	AHB	*Transfer Size:* Indicates the transfer size, which is typically byte (8-bit), halfword (16-bit) or word (32-bit).
HBURST [2:0]	Output	AHB	*Burst Type:* Indicates if the transfer forms part of a burst. 4, 8, 16 beat transfers are supported, with the burst being either incrementing or wrapping.
HWDATA [31:0]	Output	AHB	*Write Data Bus:* Used to transfer data from the master to the bus slaves during write operations.
PSELx	Input	APB	*APB Select:* Indicates that the slave device is selected and a data transfer is required. Each bus slave has a PSELx signal.

Continued

SIGNAL NAME	TYPE	BUS TYPE	DESCRIPTION
PENABLE	Input	APB	*APB Strobe:* Used to time all accesses on the peripheral bus. The enable signal is used to indicate the second cycle of the APB transfer. The rising edge of PENABLE occurs in the middle of the APB transfer.
PADDR [31:0]	Input	APB	*APB Address Bus:* 32-bit APB address bus driven by a peripheral bus bridge unit.
PWRITE	Input	APB	*APB Transfer Direction:* Indicates APB write access when HIGH and a read access when LOW.
PWDATA [31:0]	Input	APB	*APB Write Data Bus:* Driven by the peripheral bus bridge unit during write cycle. PWRITE is HIGH.
PRDATA [31:0]	Output	APB	*APB read Data Bus:* Driven by the peripheral bus bridge unit during read cycles. PWRITE is LOW.
IRQ	Output	-	*Interrupt:* Active HIGH interrupt output. The chosen bus interface type does not affect the function of this signal.

The top entity aes.vhd contains the bus interface signals and a DMA-like controller aes_enc_ctrl_struct.vhd. This module contains registers, required to setup, control and monitor the data transfer process. The master is connected to the slave interface of the peripheral bus bridge. The master initializes the bridge to receive data from the buffer located in the SDRAM. For initialization, the master has to specify signals to indicate the data size (HSIZE [2:0]) and burst type (HBURST [2:0]).

Both incrementing and wrapping bursts for 4-, 8-, and 16-beat bursts are supported in the AMBA AHB protocol, in addition to undefined-length bursts and single transfers. A beat is a transfer of data packets, thus an 8-beat wrapping burst is a transfer of 8 packets. Incrementing bursts access sequential locations and the address of each transfer in the burst is just an increment of the previous address. Wrapping bursts also access sequential locations, but if the start address of the transfer is not aligned with the total number of bytes in the bursts, then the address of the transfers in the bursts will wrap when the boundary is reached. For instance, if the starting address of a 4-beat wrapping burst of data size 4-bytes (32 bits) is `0x34`, four transfers occur on addresses

`0x34`, `0x38`, `0x3C` and `0x30`, thus, the address wrap at 16-byte (128-bit) boundaries (*Note*: the amount of data transferred, *i.e.* the number of beats times data size, is also 128 bits; $4 \times 32 = 128$). *Table 3.2-3* presents the burst signal encoding useful for defining burst types on the AHB interface.

While specifying addresses for access during a burst transfer, one must conform to the restrictions that exist regarding burst transfer addressing on the AHB interface. One of which is that bursts must not cross a 1kB address boundary. This condition sets the upper limit on the on the length of an incrementing burst. Another one states that all transfers within a burst must be aligned to the address boundary equal to the size of the transfer. For example, word (32-bit) transfers must be aligned to 32-bit address boundaries (i.e., HADDR [1:0] = 00).

The DMA-like controller implemented in aes_enc_ctrl_struct.vhd uses the 8-beat incrementing burst transfer (HBURST = INCR8) to fetch a total of 256-bit data comprising of the 128-bit *Plain text* and the 128-bit *Key*, via eight sequential, word (32-bit) accesses (HSIZE = 32) to the SDRAM location starting 0x40000000. The data is stored in the two input RAM blocks, internal to the AES top entity—topmodule.v. The AES core reads the input data from the RAMs and generates the encrypted *Cipher text* output, which is stored in the output RAM block. The DMA-like controller then transfers the 128-bit encrypted data to the other SDRAM location, 0x40001000. Information regarding the data transfer base addresses is contained in the registers in aes_enc_ctrl_struct.vhd. *Table 3.2-4* provides the register information of aes_enc_ctrl_struct.vhd.

The address offset is the offset with respect to the AHB/APB peripheral bus bridge address, which in our case is 0xc0000000. A complete listing of the source code of the AMBA-compliant wrapper, aes.vhd, along with the controller module, aes_enc_ctrl_struct.vhd, is provided in *Appendix A*.

Table 3.2-3: Burst Signal Encoding

HBURST [2:0]	TYPE	DESCRIPTION
000	SINGLE	Single transfer
001	INCR	Incrementing transfer of unspecified length
010	WRAP4	4-beat wrapping burst
011	INCR4	4-beat incrementing burst
100	WRAP8	8-beat wrapping burst
101	INCR8	8-beat incrementing burst
110	WRAP16	16-beat wrapping burst
111	INCR16	16-beat incrementing burst

Table 3.2-4: AES Core Register Information

REGISTER NAME	ADDRESS OFFSET	DESCRIPTION
Control Register	0x00	Bit 0 is used to enable the AES core: 1-enable, 0-disable Bit 2 is used to enable IRQ: 1-enable, 0-disable Bit 3 used to generate IRQ request: 1-enable, 0-disable
Key / Plain text Input Address	0x04	This register contains the 32-bit base address of the Key and Plain text input from the SDRAM location starting 0x40000000
Cipher text Output Address	0x08	This register contains the 32-bit base address of the Cipher text output to the SDRAM location starting 0x40001000
Status Register	0x0C	Bit 0 contains the status of transfer done: 1-ready, 0-busy Bit 1 contains the status of transfer direction: 1-write, 0-read
Current Address	0x10	This register contains the 32-bit address from (to) which the DMA-like controller (aes_enc_ctrl_struct.vhd) reads (writes) data

It can be observed that the controller implements a finite state machine (FSM) for accesses to the Leon SDRAM:

```
TYPE state_type IS (idle, bus_req, bus_grant, bus_own, load_key, load_text,
xfer_end);
```

The FSM consists of seven states and starts in the IDLE state upon being reset. In this state, all registers and bus contents are initialized to zero.

One clock cycle later, the FSM moves to the BUS_REQ state, where it checks for internal signal conditions. When the signal dma_xfer_req of record, **r**, is asserted high, the internal signal sig_HBUSREQ is set high. Then, if the AES core is selected and enabled (aes_en) and data is ready for transfer to the AES core (sig_dataRdy), signals in the other record, tmp, are backed up. The FSM goes to the next state BUS_GRANT if all these signals and the ahb_hgrant signal are high. The register that counts the number of data transfers, n, is initialized to seven in the record, tmp, so that the count becomes zero after a total of eight transfers. The eight transfers consist of four transfers each, of width 32 bits, for *Key* and *Plain text*.

When in the BUS_GRANT state, the FSM simply skips one clock cycle before accepting ownership of the bus in the next state, BUS_OWN.

In this state, the FSM waits until the AHB master asserts HIGH on the signal ahb_hready. The FSM makes the signal sig_dataRdy LOW as an indication to the AES core that all data is still not present for encryption. The FSM also sets the transfer type for the first data transfer as *Nonsequential* (HTRANS = HTRANS_NONSEQ). The data transfer begins in the next state, LOAD_KEY.

During the LOAD_KEY state, the FSM loops for four clock cycles to load the 128-bit *Key* in one of the input RAM blocks. This has been implemented by a two-state minor FSM within this state, which also sets the subsequent transfer types as *Sequential* (HTRANS

= HTRANS_SEQ). The FSM loads the 128-bit *Plain Text* the same way in the other input RAM block in the next state, LOAD_TEXT.

With all data now stored in the internal input RAMs, it is available for encryption by the AES core. The FSM pulses the signal, sig_dataRdy, HIGH to convey this to the to AES core, and also sets the signal, sig_Go, HIGH to begin encryption. The FSM then moves on to the final state, XFER_END.

The last state of the FSM signals the completion of the encryption process and all the data transfers that occurred during that process (sig_finish = '1'). The FSM disables the AES core and returns to the IDLE state.

3.2.4 Platform Express: Concepts and Objects

A system *designer* using the *PX* application needs *objects* to quickly create a design. A system *integrator* creates these objects from the available hardware designs. The *PX* object types can be categorized as components, buses and routines. *Table 3.2-5,* from the *Platform Express Integrator's Guide* [22], shows the object and routine types that can be defined into a component library.

Object Types

PX objects are classified as components and buses, and are defined using schemas that the *PX* software can use.

Schemas

The *PX* schemas are based on the World Wide Web Consortium (W3C) standard for XML 1.0, accepted in 2001. The schema for component files is located in the `$PXHOME/schema/3.5` directory. A more easy-to-read version in the HTML format can be found in `$PXHOME/doc/schema`.

Table 3.2-5: Platform Express Object and Routine Types

TYPE	CREATION PROCESS	NOTES
Component: A set of files containing all information required by the Platform Express application to use the component in a design. Subtypes: Platform core Hierarchical Component Bus Bridge Peripheral	1. Locate a similar component to use as a template. 2. Make a component directory. 3. Edit the component's XML file. 4. Add necessary support files – Generators, Configurators, HDL or C code. 5. Test the component. 6. Package the library.	Use *PxEdit* to quickly enter basic component information. Hierarchical Components can be created using a HC Generator.
Buses: Used to connect components together.	1. Create a bus definition based on an official specification 2. Create a decoder so that components can connect to the bus. 3. Write a component that references the bus so that the bus can be tested. 4. Test the bus. 5. Package the library.	Bus interface definitions and decoders must be available before a component can be tested. *Platform Express* libraries provide several common types.
Generators: Invoked by a user to perform actions. For instance, the *Platform Express* application uses generators to create design documentation and to build the HDL model of the final design.	1. Define the function in its simplest terms. It is advisable to write many small generators than a single large one. 2. Write the generator(s). 3. Attach the generator(s) to the components, or make them accessible through a visible generator chain.	Most generators are written for components. Users can create stand-alone generators such as the *PxDoc* library. The *PxDoc* and *checkEnvironment* libraries contain <u>*only*</u> generators.
Configurators: Used to instantiate elements. Invoked when a component is added to a design.	1. Decide on the type of the configurator required. 2. Write a configurator in Java, if the provided default configurator is unusable. 3. Attach the configurator to the component.	The default configurator is very flexible and can handle most needs in the component's XML. The *pxSampleLib* library contains a component with a custom configurator.
Platform Metadata (PMD): Changes a component based upon what other components are present in the design, in a manner specified by the PMD writer. Changes can be as simple as restricting choices in a dialog box, or as extreme as adding or deleting a bus interface.	1. Define the triggering component, the affected components and the changes to be made. 2. Create transformers either in Java or XLST. 3. Package the PMD in its own directory in a component library.	The components referenced by a PMD need not be in the same component library as the PMD.

Source: Platform Express Integrator's Guide

Components

These represent the different types of IP blocks that can be included in the chip design. Since they appear in the Component Browser as icons, they are the most visible among all *PX* objects. Users can drag-n-drop components into the Design Editor. Platform cores, hierarchical components and bus bridges are subtypes that can be used to start a design. Peripherals form the other subtype, which can only be added to existing buses.

Buses

Buses are a fundamental notion of the *PX* code structure, because all components are connected using buses. A component is not displayed in the Component Browser, if it cannot connect to any of the active bus types (in the current design) using any of the bus bridges. The bus implementation is a three-step process—signals are described in bus definition files, the HDL for the component-bus connection is generated from bus decoder templates and finally, pins to connect the signals are set up in the component's bus interface section.

Note: Writing the *PX* Routine Types (Generators, Configurators and PMD) are beyond the scope of this thesis and hence, not discussed here. Readers are strongly encouraged to refer to the *Platform Express Integrator's Guide* [22] for information on Routine Types.

Platform Express Directory Structure

The exploded view of the *PX* directory structure can be observed in *Figure 3.2-6*. The default *PX* installation results in two top-level file directories pxhome and pxLibraries.

The pxhome directory contains all the core information including integrator scripts and the *Platform Express* application code. pxLibraries contains all the libraries that came with the *PX* environment. However, not having any of these libraries does not influence *PX* performance. It is under pxLibraries directory that you will package your components in a physical library.

```
Platform Express
+- pxhome // Contains Core Information
|      +- api [Platform Express API Documentation]
|      |
|      +- bin [Startup Scripts]
|      |
|      +- class [3rd Party Java Code]
|      |
|      +- componentLibrary [Generator Chains & Kernel Software]
|      |
|      +- doc [User's Guide, Integrator's Guide, ReadMe Files for Installation, Setup
|      |  |      and Licensing]
|      |  +- schema [XML Schemas for all Objects; Presented in a Tree Diagram]
|      +- etc [Default User Settings]
|      |
|      +- images [Icons for GUI]
|      |
|      +- mgls [License Server Utilities]
|      |
|      +- schema [Platform Express' Official Schema; Expressed as XSD Files]
|      |
|      +- tools [Integrator Scripts]
|         +- bin [pxedit, mkIndex, Pxkeygen and Converters]
|
+- pxLibraries // Contains Libraries that Integrators Write
      +- PxArm9 [ARM Processor Library]
      |
      +- Leon2 [Leon Processor Library]
      |
      +- AMBA [AMBA Bus Library]
      |
      +- bbcLib [Black Box Component Library]
      .
      .        // Other Libraries
      .
      +- <yourLibrary>
            +- componentLibrary [Holds Subdirectories for Components and Routine
            |     |              Types]
            |     +- component [Holds Various Component Directories]
            |        +- <yourComponent> [Your Component]
            |           +- <version> [Name this as X.Y; Use only digits]
            |              +- hdlsrc [All HDL Source Code for the Component]
            |              |- <yourComponent>.xml [Generated Using pxedit]
            |
            +- doc [Optional Directory; Component Documentation]
            |
            +- images [Optional Directory; .gif or .jpeg Files]
            |
            |- index.xml [Optional File; Speeds up Loading; Generated Using mkIndex]
            |- Pxkey [Contains Licensing Information; Generated Using Pxkeygen]
```

Figure 3.2-6: The *Platform Express* Directory Structure

Integrators should package their components according to a specific directory structure that allows the included *PX* utilities to locate supporting files. This generic directory structure is illustrated in under <yourLibrary>, where <yourLibrary> is a single distinctively named directory within pxLibraries. The name of <yourLibrary> should be suggestive of the components it may include.

The required directory, componentLibrary, under <yourLibrary> contains the subdirectories for components, bus definitions and routine types. However, all of the subdirectories may not be required. The other required element in <yourLibrary> is the Pxkey file, which holds the licensing information for your packaged component. <yourLibrary> may also contain other optional directories and support files such as index.xml or Makefile.

The component subdirectory under componentLibrary may contain multiple components, one of which could be <yourComponent>. Each of <yourComponent> must have at least one <version> directory, where <version> is a number in the form of X.Y. If multiple <version> directories are present then *PX* will use the most recent version (the one with the highest number).

The <version> directory contains <yourComponent>.XML file and all the supporting files or directories that you want to protect using Pxkey.

Figure 3.2-7 illustrates the directory structure of an AES IP component that was packaged into a physical library named VOLIPository (equivalent to <yourLibrary>) and later integrated into the *Platform Express* environment.

The next chapter discusses the details of preparing the IP component before installing it in a component library.

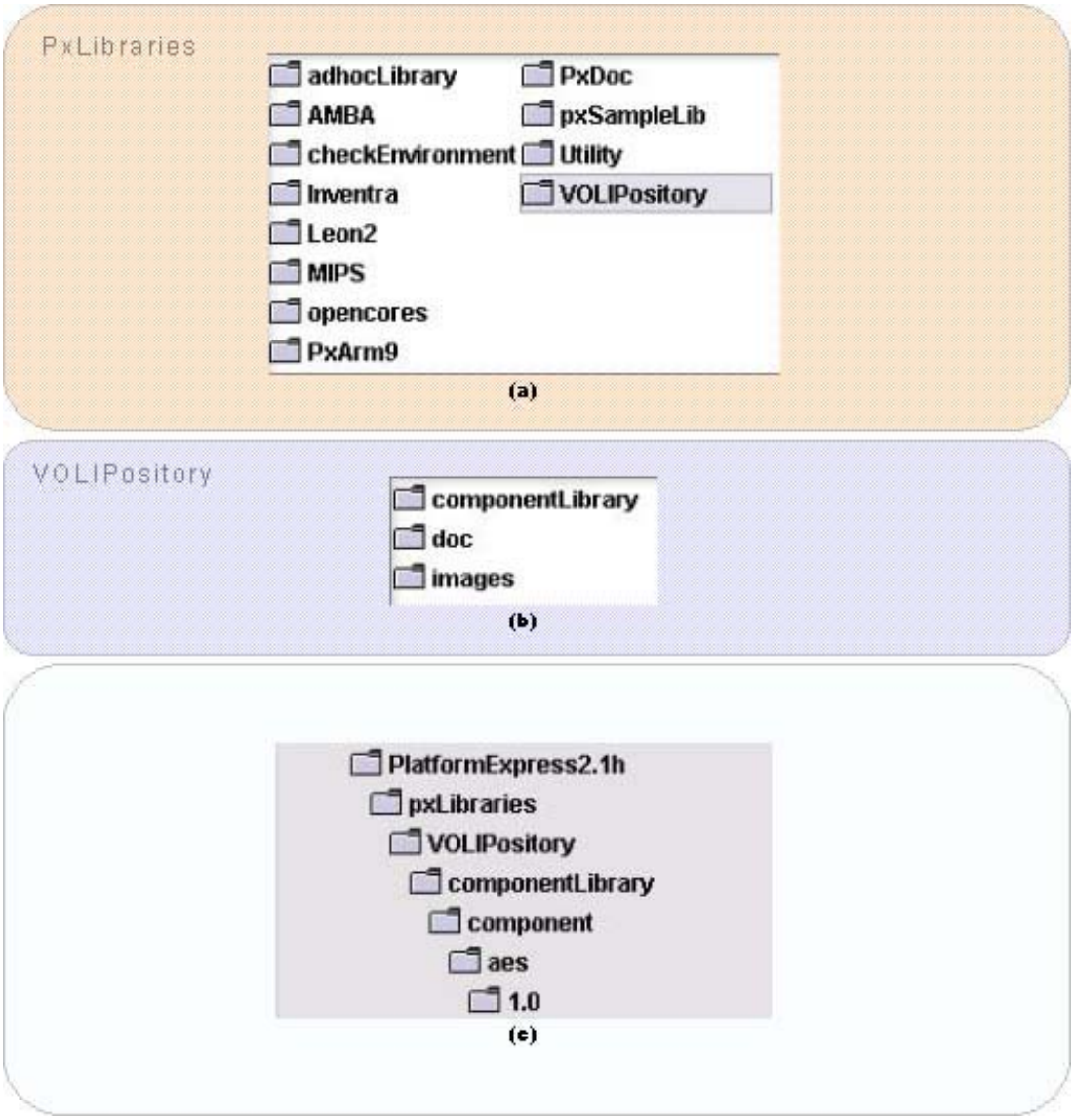
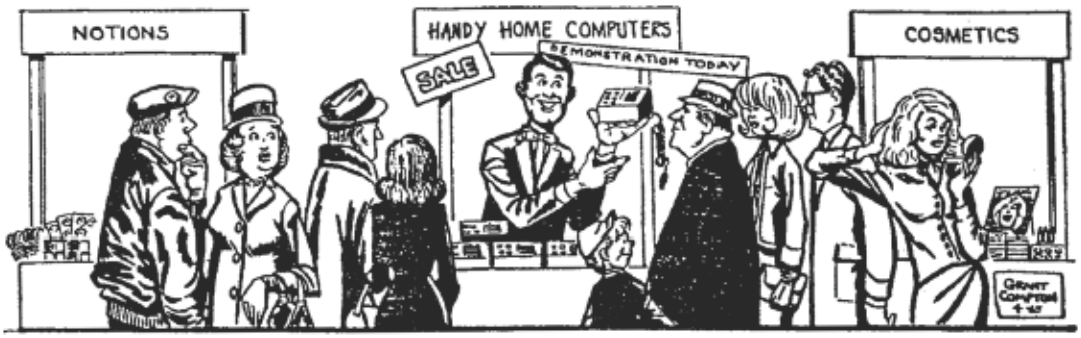


Figure 3.2-7: (a)Creating a **VOLIPository** library into pxLibraries; (b)Creating subdirectories in VOLIPository; (c)Directory Structure Showing Location of the **aes.xml** File of the **aes** Component.

4

IMPLEMENTATION



--From Gordon E. Moore's paper, *"Cramming more components onto integrated circuits"*

This chapter explains the use of the *Platform Express* application from two different design perspectives – that of a *System Designer* and another of an *IP Integrator*. The orange arrows in *Figure 4.1-1* indicate the complete process – from IP integration to platform conception – followed in this project. As an IP integrator we need to carry out steps **1** through **7** to be able to use the installed IP, when performing the role of a system designer in step **8**.

Step **1** indicates that the raw IP can be described either using VHDL or Verilog. In step **2**, if the IP needs memory for storing data before and after processing it then input and output RAMs are added. We are executing step **3** because our choice of HDL is VHDL while our IP is described in Verilog. (If your IP along with its bus-compliant wrapper is described using the same HDL then step 3 can be omitted). In step **4** we add the AMBA-compliant wrapper to our peripheral module. In step **5**, we use the *PxEdit* tool to generate our IP's metadata file in XML (explained in *Section 4.4*) before installing it into the *pxLibraries* (step **6**). Next we generate a *Pxkey* (step **7**) to protect our IP from modifications and also so that it appears in the Component Browser of the *PX* application.

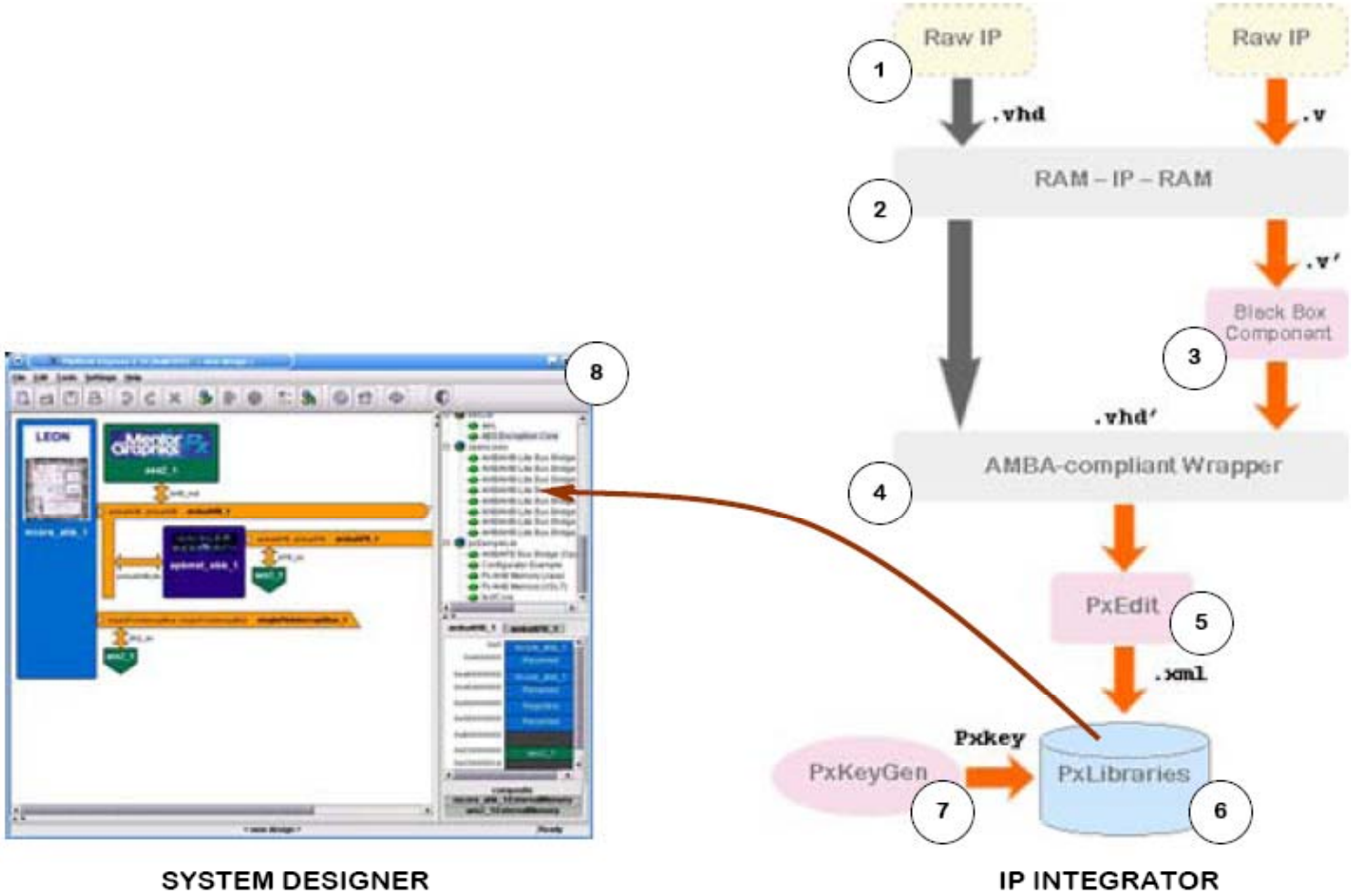


Figure 4.1-1: IP Integration and Platform Creation using Platform Express

This exercise describes the following steps:

- Obtaining the AES IP core
- Obtaining the *Platform Express* design environment
- Compiling the IP Core using *ModelSim*
- Installing the AES component using *pxedit, mkIndex* and *Pxkeygen*
- Generating a test design using *Platform Express*
- Verifying the design for correctness using *Seamless CVE*

4.1 OBTAINING THE *AES* IP CORE

The required AES IP component for integration with *Platform Express* can be obtained from the OpenCores website. The following instructions describe the process of obtaining the IP core via the Internet.

1. Point your web browser to http://www.opencores.org/
2. Click **CVSGet** under **Tools** menu
3. On the **CVS Module Download** page type aes_core in the **Module Name** box and click the **Create module.tar.gz** button
4. On the **Download** page, enter the required information and the click the **Download** button
5. Save the aes_core.tar.gz file in your home directory
6. At the Unix prompt, type the following commands

```
Mkdir test; cd test
gunzip -c /home/<yourUsername>/aes_core.tar.gz | xvf -
cd aes_core/rtl/verilog
cp /home/wala/test/hdl/aes/controller.v .
cp /home/wala/test/hdl/aes/topmodule.v .
cp /home/wala/test/hdl/aes/aes.vhd .
cp /home/wala/test/hdl/aes/aes_enc_ctrl_struct.vhd .
cd home
```

4.2 OBTAINING PLATFORM EXPRESS

The *Platform Express* version used at the time of this project was 2.1h. A fully functional, latest version of *Platform Express* can be obtained for free from the Mentor Graphics website. The following instructions describe the process of obtaining the *Platform Express* environment via the Internet.

1. Point your web browser to http://www.mentor.com/products/embedded_software/platform_baseddesign/download.cfm
2. On the **Product Download** page, enter the required information; check the **Platform Express with all libraries for Solaris 2.8** box under **Combined Software Plus Libraries** option and click the **Get Software** button and save the pxplus_ss5_<*ver*>.exe in your home directory
3. At the Unix prompt, type the following commands in the exact sequence to install the PX environment

```
chmod 775 pxplus_ss5_<ver>.exe
pxplus_ss5_<ver>.exe
```

4. Type either D or P when the installation executable starts and then type Agree to accept the license terms. Finally, just hit the **Enter** key to accept the default installation home directory

Alternately, a copy of *Platform Express* version 2.1h can be obtained by executing the following command. For installing the software, follow steps 3 and 4 described above.

```
cp /home/wala/PlatformExpress2.1h/pxplus_ss5_2.1h.exe .
```

4.3 COMPILING THE IP CORE

Before we begin installing our IP component into our *Platform Express* library, we need to compile it for a smooth integration process. The following instructions describe the compilation process using *ModelSim*.

1. Copy the `compile` script file into your test/aes_core/rtl/verilog directory.

```
cd test/aes_core/rtl/verilog
cp /home/wala/test/hdl/aes/compile .
```

Compile Script

```
#!
source ~cad/.cshrc
mentor_tools
vlib work
vmap dware /home/wala/dware
vmap dw06 /home/wala/dw06
vmap work work
vcom -work work target.vhd device.vhd amba.vhd
vcom -work work config.vhd sparcv8.vhd iface.vhd
vcom -work work DW_ram_r_w_a_dff_inst.vhd
vlog -work work timescale.v aes_rcon.v aes_sbox.v
vlog -work work aes_key_expand_128.v aes_cipher_top.v
vlog -work work controller.v topmodule.v
vcom -work work aes_enc_ctrl_struct.vhd aes.vhd
```

The script instantiates the necessary tools, creates a work library, maps the work library to compilation library and defines the compilation order according to component hierarchy.

2. Make an executable compile file and start compile process

```
chmod 775 compile
compile
```

The *ModelSim* output should be identical to:

Compilation Log

```
Copying /sw/mentor/ModelSim_SE5.8d/modeltech/sunos5/../modelsim.ini to
modelsim.ini
Modifying modelsim.ini
Modifying modelsim.ini
Modifying modelsim.ini
Model Technology ModelSim SE vcom 5.8d Compiler 2004.06 Jun 12 2004
-- Loading package standard
-- Loading package std_logic_1164
-- Loading package attributes
-- Loading package std_logic_misc
-- Loading package std_logic_arith
-- Loading package dwpackages
-- Loading package dw06_components
-- Compiling entity dw_ram_r_w_a_dff_inst
-- Compiling architecture inst of dw_ram_r_w_a_dff_inst
-- Loading entity dw_ram_r_w_a_dff
-- Compiling configuration dw_ram_r_w_a_dff_inst_cfg_inst
-- Loading entity dw_ram_r_w_a_dff_inst
-- Loading architecture inst of dw_ram_r_w_a_dff_inst
-- Loading configuration dw_ram_r_w_a_dff_cfg_sim
Model Technology ModelSim SE vlog 5.8d Compiler 2004.06 Jun 12 2004
-- Compiling module aes_rcon
-- Compiling module aes_sbox

Top level modules:
        aes_rcon
        aes_sbox
Model Technology ModelSim SE vlog 5.8d Compiler 2004.06 Jun 12 2004
-- Compiling module aes_key_expand_128
-- Compiling module aes_cipher_top

Top level modules:
        aes_cipher_top
Model Technology ModelSim SE vlog 5.8d Compiler 2004.06 Jun 12 2004
-- Compiling module controller
-- Compiling module topmodule

Top level modules:
        topmodule
Model Technology ModelSim SE vcom 5.8d Compiler 2004.06 Jun 12 2004
-- Loading package standard
-- Loading package std_logic_1164
-- Loading package std_logic_arith
-- Loading package std_logic_signed
-- Compiling entity aes_enc_ctrl_struct
-- Compiling architecture structural of aes_enc_ctrl_struct
-- Loading package vl_types
-- Loading entity topmodule
-- Compiling entity aes
-- Compiling architecture rtl of aes
-- Loading entity aes_enc_ctrl_struct
```

4.4 INTEGRATING THE IP CORE

The *PxEdit* tool supplied with the *PX* software significantly reduces the amount of typing required in creating component definition files. The tool allows the user to fill in fields for standard elements of the component, and then generates the XML for these areas. The XML file created is a valid XML file and can be customized according to needs.

4.4.1 Starting with the Compiled HDL Model

1. Copy the .plex_rc script file into your test/aes_core/rtl/verilog directory.

```
cp /home/wala/.plex_rc .
```

.plex_rc Script

```
setenv PXHOME /home/wala/PlatformExpress2.1h/pxhome
setenv PXPATH
      /home/wala/PlatformExpress2.1h/pxLibraries/AMBA:
      /home/wala/PlatformExpress2.1h/pxLibraries/Inventra:
      /home/wala/PlatformExpress2.1h/pxLibraries/Leon2:
      /home/wala/PlatformExpress2.1h/pxLibraries/PxArm9:
      /home/wala/PlatformExpress2.1h/pxLibraries/Utility:
setenv MODELTECH /sw/mentor/ModelSim_SE5.8d/modeltech
setenv CVE_HOME /sw/mentor/CVE5.0/cve_home.ss5
setenv JAVAHOME $CVE_HOME/jre
setenv PATH $JAVAHOME/bin:$PATH
```

2. Invoke the PxEdit tool

```
source ~cad/.cshrc
mentor_tools
source .plex_rc
$PXHOME/tools/bin/px&
```

   The *PxEdit* window will appear as shown in *Figure 4.4-1*. Select option **New** under the **File** tab.

3. Fill in the dialog box with the definition information as shown in *Figure 4.4-2* and *Table 4.4-1*.
4. Click **OK**. *PxEdit* now extracts the top-level signal names from the HDL library. Maximize the window to view the navigation tabs located on the right side.

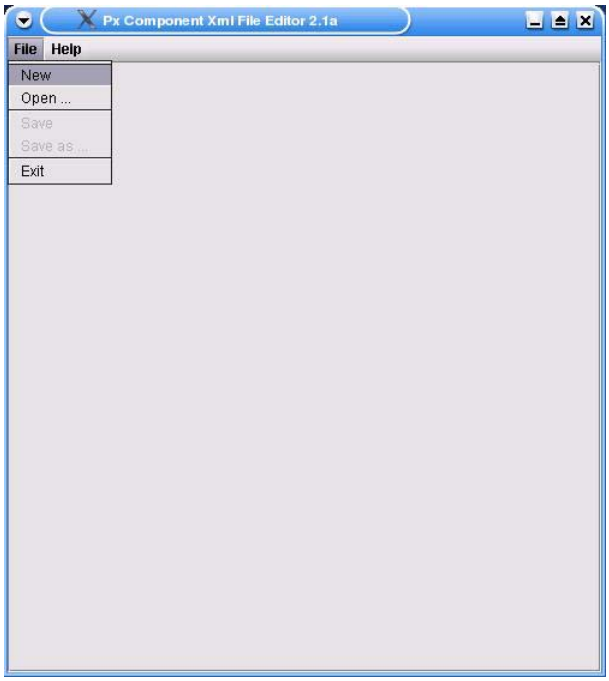


Figure 4.4-1: PxEdit Environment

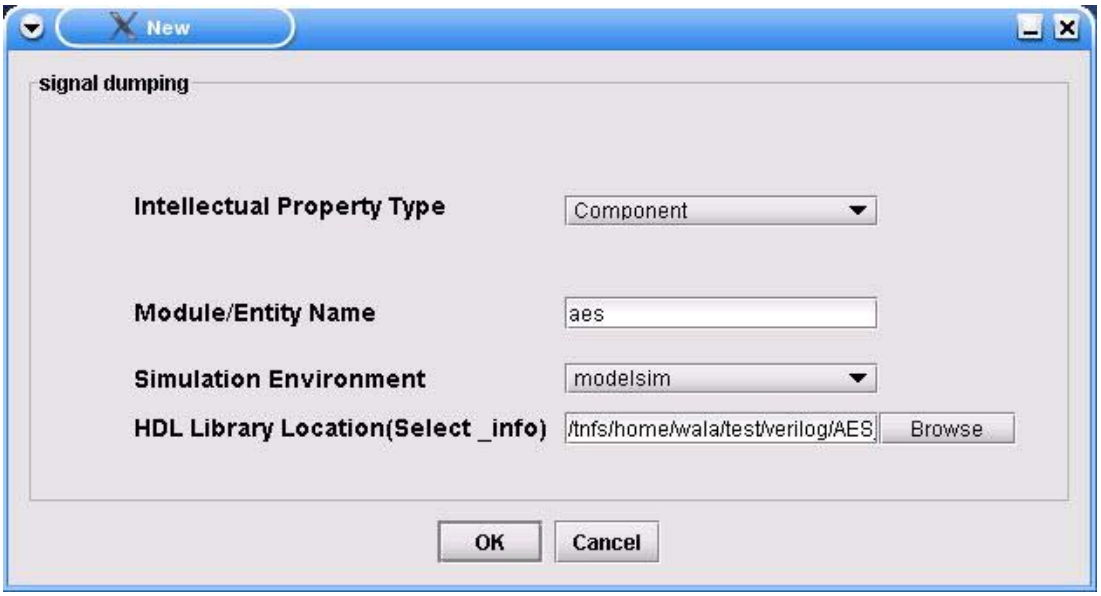


Figure 4.4-2: Signal Dumping Dialog Box

Table 4.4-1: Signal Dumping Dialog Box Information

Intellectual Property Type	Select *Component*. The *Platform Core* option can only be selected if the IP is a core or a CPU.
Module/Entity Name	Enter the topmost HDL model name exactly; this field is case-sensitive.
Simulation Environment	Select *modelsim*. The *modelsimcve* option is selected only when the HDL model is compiled using Seamless PSP.
HDL Library Location	This refers to the directory containing the HDL library compiled by ModelSim—usually *work*. Use the **Browse** button to select any file under the *work* directory. *Figure 4.4-3* shows the *_info* file selected inside the *work* directory for illustration purposes.

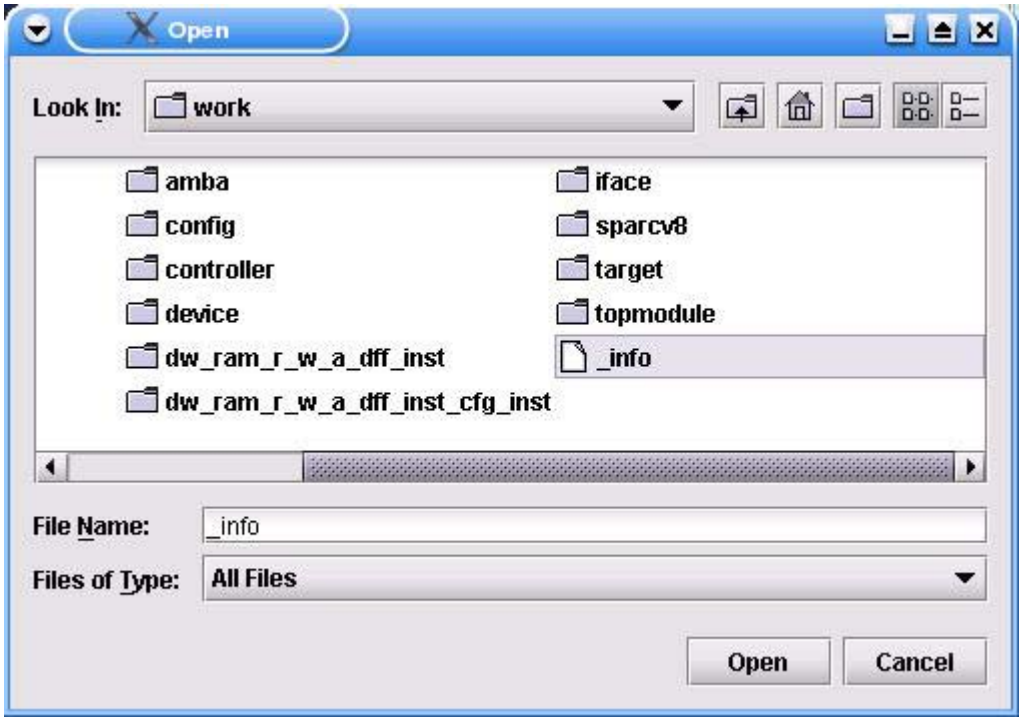


Figure 4.4-3: HDL Location Specification

4.4.2 Configuring Buses

PX uses the bus information to connect components together. Buses are also essential because they define the address spaces of the components. Standard bus information, such as signals, is defined separately in a bus definition (busdef) file, and must be in $PXPATH for *PxEdit* to work correctly (see, amba*.xml under pxLibraries/AMBA/ componentLibrary/busdef or pVCI.xml under pxLibraries/Inventra/componentLibrary/ busdef).

1. Click the **busInterfaces** tab at the right. For our design we are going to connect the components using the industry standard AMBA bus. Primarily three bus types will be used for connecting the IP core with the Leon2 CPU—the AMBA AHB, the AMBA APB and a single pin interrupt bus for generating IRQ requests.
2. Pick these buses by highlighting the appropriate options under the **Bus Available** list and clicking **Select**. *Figure 4.4-4* depicts the bus name input dialog box that comes up after bus selection. The role of the component as a bus master or a bus slave is defined in the next resulting dialog box, shown in *Figure 4.4-5*.

   For the sake of convenience, name the AHB master bus as AHB_mst, the APB slave bus as APB_slv and the Single Pin Interrupt bus as IRQ_slv.
3. Click on a cell in the **Signal Mapping** table to list the set of signals for that bus type. Map each bus signal in the table to the corresponding bus signal listed in the drop-down menu (see *Figure 4.4-6*).
4. The bus connections can be specified as **required** or **optional** in the **Select Master/Slave/System** bus table. The default value for all connections is **optional**. Set this to **required**, for the APB slave and the single pin interrupt bus, as shown in *Figure 4.4-7*.
5. Specify the connection of the AES core on the APB slave bus with respect to the Leon address space. Select the APB_slv bus in the **Select Master/ Slave/System** table. Click **Add** above the **MemMap/AddressBlock** table and fill in the values in the table as shown in *Figure 4.4-7*.

Figure 4.4-4: Bus Name Input Dialog Box

Figure 4.4-5: Bus Interface Specification

Figure 4.4-6: Signal Mapping

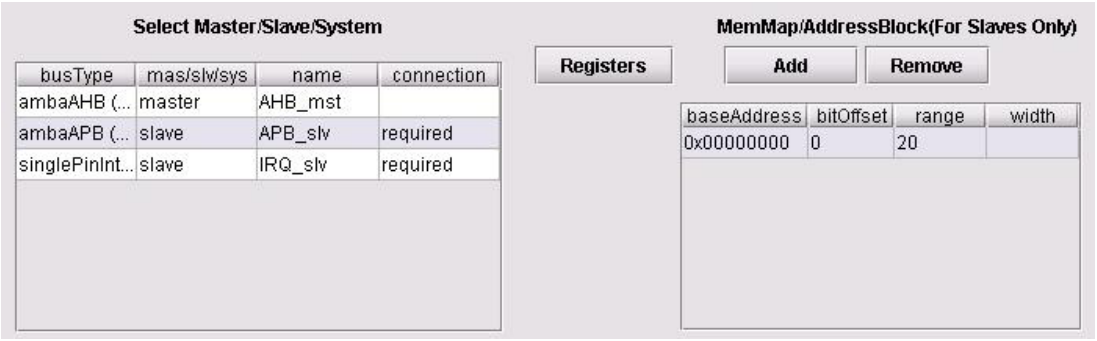
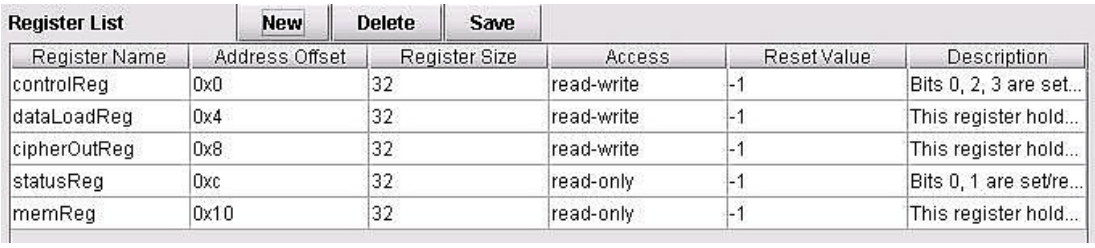


Figure 4.4-7: Bus Connection and Memory Map Specification

6. In the **registerBank/name** window, click **Add** and name the field as **registers**, as shown in *Figure 4.4-8*. Click **Registers** and enter the register information in the **Register List** table as shown in *Figure 4.4-9*. Use **New** to add registers and save your work frequently. Specify the register bit information in the **Field List** for each register. Click **Save** before closing the window to avoid validation errors. The detailed bit information for controlReg, dataLoadReg, cipherOutReg, statusReg and memReg is shown in *Figure 4.4-10 – 14,* respectively.

Figure 4.4-8: Register Bank Window



Register Name	Address Offset	Register Size	Access	Reset Value	Description
controlReg	0x0	32	read-write	-1	Bits 0, 2, 3 are set...
dataLoadReg	0x4	32	read-write	-1	This register hold...
cipherOutReg	0x8	32	read-write	-1	This register hold...
statusReg	0xc	32	read-only	-1	Bits 0, 1 are set/re...
memReg	0x10	32	read-only	-1	This register hold...

Figure 4.4-9: Register List Input Table

Field List | New | Delete

Field Name	Bit Offset	Bit Width	Read/Write	Description
coreEnable	0	1	read-write	set if AES core...
Reserved	1	1	read-write	Reserved
irqEnable	2	1	read-write	set if IRQ enab...
irqRequest	3	1	read-write	set if IRQ requ...
Reserved	4	28	read-write	

Figure 4.4-10: Field List Information for controlReg

Field List | New | Delete

Field Name	Bit Offset	Bit Width	Read/Write	Description
rdStartAddr	0	32	read-write	Input KEY and PLAIN from this...

Value List | New | Delete

Value Name	Value	Description
rdAddr	0x40000000	Leon2 Inte...

Figure 4.4-11: Field List Information for dataLoadReg

Field List | New | Delete

Field Name	Bit Offset	Bit Width	Read/Write	Description
wrStartAddr	0	32	read-write	Cipher output stored to this ad...

Value List | New | Delete

Value Name	Value	Description
wrAddr	0x40001000	Leon2 Inte...

Figure 4.4-12: Field List Information for cipherOutReg

Field List | New | Delete

Field Name	Bit Offset	Bit Width	Read/Write	Description
ready	0	1	read-only	Ready/Busy status indicator
memwr	1	1	read-only	Writing/Reading status indicator
Reserved	2	30	read-write	Reserved

Figure 4.4-13: Field List Information for statusReg

Field List | New | Delete

Field Name	Bit Offset	Bit Width	Read/Write	Description
currentHADDR	0	32	read-write	Current Address on AHB HAD...

Figure 4.4-14: Field List Information for memReg

4.4.3 Describing the Component Appearance

In the *Platform Express* environment, the appearance of your component can be described under the **presentation** tab. *Table 4.4-2* lists the input fields found under this item, while *Figure 4.4-15* actually shows the **presentation** interface, with information entered in all fields with respect to our AES component.

4.4.4 Setting up the Verification Environment

The details required to create your component in the final design are to be provided under this option. It is always better to have a component support more than one simulation environment and language, this way it stands a better chance to be incorporated in many designs. Various fields and entries in this tab are illustrated in *Figure 4.4-16*.

Table 4.4-2: Presentation Information

Display Label	The preferred name for the component is to be entered here. The component will be referenced by this name in the Component Browser. If multiple names are entered, only component will be referred by the first one. Spaces are valid in Display Labels (as opposed to Register Names and Register Field Names).
Icon	The icon shows up in the block diagram of the component, when it is dragged into the Design Editor. Enter the relative path to the icon in the library directory. Users can create icons in GIF or JPEG formats measuring 100 pixels by 35 pixels (width X height). Leaving this field blank automatically allows *Platform Express* to use the default icon.
Document Location	Any support information for the component – web URL, relative path to the location of datasheets or application notes – goes in this field. The information will be available to end-users when they right-click your component and select **Browse**.

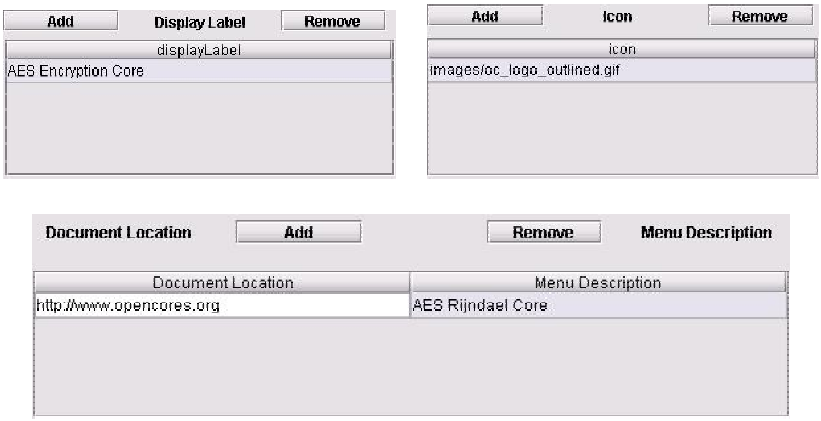


Figure 4.4-15: Component Appearance Description

Figure 4.4-16: Verification Environment Specifications

1. Click the **hwModel** tab and click **Add** under the **VerificationEnv** field. Enter any name under **id** (preferably indicating simulator and language type). Select the HDL of your component description from the drop-down menu under **language**.
2. **Add** a field under **EnvironmentId** and select an option from the drop-down menu for **envIdentifier**.
3. **Add** a field under **Parameter** and enter the entity names of the modules in your component – in the bottom-up hierarchical order – under **value**. Select **entityName** from the drop-down list under **name.**
4. From a list of options under **fileType**, select the HDL source in which your component is described.
5. Specify a fileset for your **Parameter value** under **fileSetRef**. This exact fileset ID will be referenced under the **fileSets** tab.

4.4.5 Adding Supporting Files

Under **fileSets**, you will specify the location of all files under **fileSetRef** in **hwModel**. Additionally, you may also specify supporting software files for your component if it generates its own drivers.

1. **Add** a field under **File Set** and enter the exact **fileSetId** as the one under **fileSetRef**. Make sure that the spelling and the case are identical.
2. **Add** the relative location of each file belonging to this fileset. In our case, it is the top-level file aes.vhd under hdlsrc directory, as shown in *Figure 4.4-17*. Under the other required field, **fileType**, specify all HDL of all your files.
3. After adding all files for a particular fileset, click **Add** to specify another fileset.
4. **Save** your work and eliminate validation errors, if any. Usually *pxEdit* will generate a message similar to the one shown in *Figure 4.4-18*.

For more information on adding supporting filesets, see mcore_ahb.xml under pxLibraries/Leon2/componentLibrary/component/mcore_ahb/1.0.

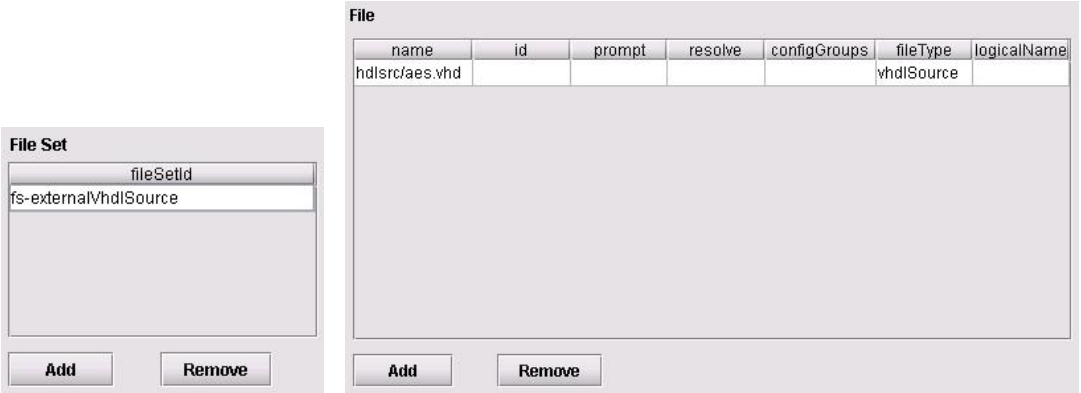


Figure 4.4-17: Fileset Specification

Figure 4.4-18: PxEdit Validation Message

A component wrapper recognized by the *Platform Express* application can be generated using *PxEdit*. The tool only results in a bare minimum XML structure of the component. The XML file, attached in *Appendix B*, is editable outside of the *PxEdit* environment in any text editor, if the component requires any dependencies to be added. In our case however, no additional generators/configurators or dependencies are required for the AES IP core. For details regarding adding CPU cores, writing generators and configurators, resolving dependencies, adding bus interfaces (other than the included AMBA and Inventra buses) and to read more on adding components, please refer to the *Platform Express Integrator's Guide* [22].

Once the component XML file is created, make a new component library under pxLibraries and package your component into that library as described in section 3.3. Next, edit .plex_rc by adding the new component library location to $PXPATH. In order to reduce the *Platform Express* loading time, generate an index.xml file for the component using mkIndex. From the command prompt, traverse to the pxLibraries directory and enter the following command:

```
$PXHOME/tools/bin/mkIndex <libraryName>
```

IP integrators can prevent end-users from modifying their libraries by using Pxkeygen utility to generate a Pxkey file. Modification is only possible when end-users buy the Pxkey license. To generate Pxkey, go to the pxLibraries directory and enter the following command:

```
$PXHOME/tools/bin/Pxkeygen.sh <libraryName>
```

The next section illustrates the process of integrating a component described using multiple HDLs (VHDL + Verilog) and can be skipped if you are working with a component written entirely in just one HDL.

4.4.6 Generating a Black Box Component

Since most companies prefer using only one HDL to describe their IP cores, the current version of *Platform Express* only supports verification of a component described in a single HDL. In our case however, since we have used the AES IP core available on OpenCores.org, written in Verilog and the AMBA wrapper written in-house using VHDL, the design verification process is not possible. (This issue has been reported to the *Platform Express* support team and is likely to be resolved in the future releases.)

Nevertheless, the current version of the *Platform Express* environment provides a smart way to describe a component as a black box in which only the bus interface signals are imported in the XML file and no filesets are created.

The **Generate Black Box** option under **Tools** menu (see *Figure 4.4-19*) in *Platform Express* application brings up the black box generator interface that allows users to create components for installation under the black box library (bbcLib).

1. Click  on the generator interface and in the resulting **BBC Component Editor** dialog box (see *Figure 4.4-20*) enter the name of the component.
2. Click and drag the **Bus Interface** symbol,  onto the component name in the black box editor. In the resulting **Bus Interface Editor** dialog box shown in *Figure 4.4-21*, specify the **Bus Type** as **ambaAHB Master**.
3. Repeat step 2 twice, and specify **Bus Type** as **ambaAPB Slave** and **Interrupt Slave**. The black box generator interface would now be identical to *Figure 4.4-22*.
4. Select **File > Generate…** and navigate to the PlatformExpress2.1h directory in the dialog box that appears. Select **pxLibraries** and click **Generate** as illustrated in *Figure 4.4-23*. The bbcLib library will be placed inside the pxLibraries directory. The terminal window will show an output similar to *Figure 4.4-24*.

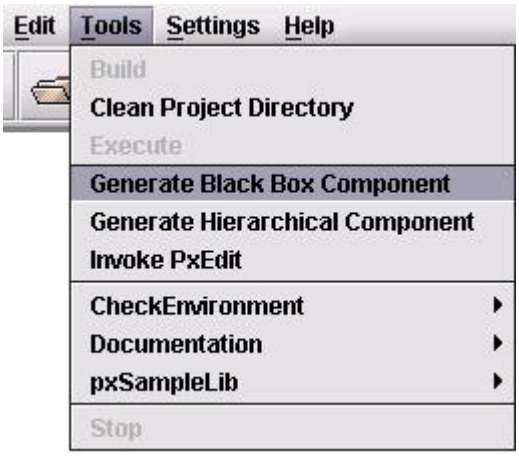


Figure 4.4-19: Invoking the Black Box Generator Interface

Figure 4.4-20: Component Editor Dialog Box

Figure 4.4-21: Bus Interface Dialog Box

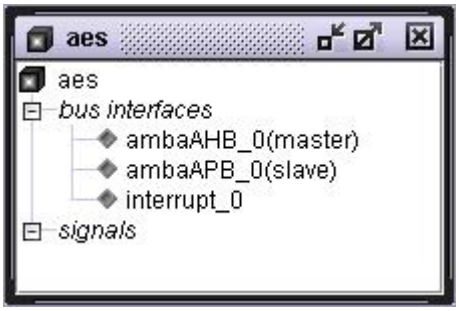


Figure 4.4-22: BBC Generator Interface

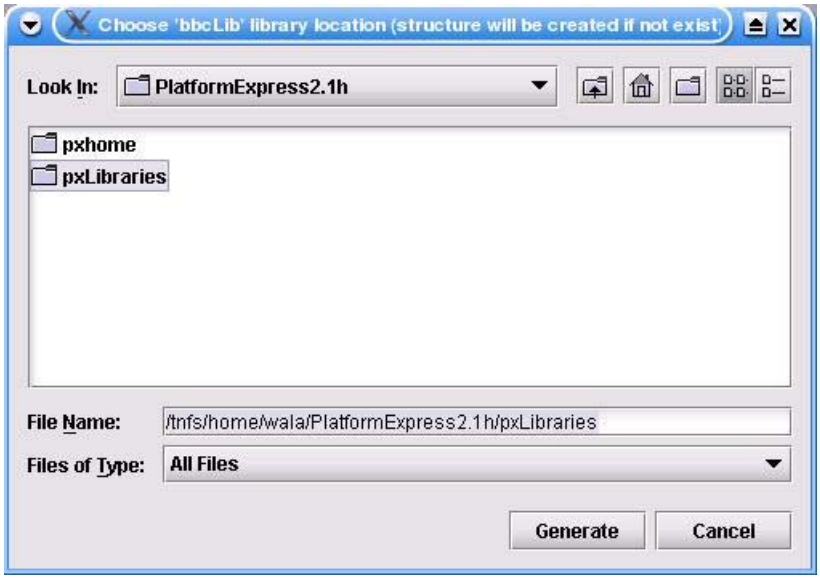


Figure 4.4-23: bbcLib Creation Directory

```
Created /tnfs/home/wala/PlatformExpress2.1h/pxLibraries/bbcLib/Pxkey
-----------------------------------------
Expiration Date: Jan. 15, 2007
Library ID:   4121681882
-----------------------------------------
```

Figure 4.4-24: bbcLib Creation Message

4.5 GENERATING A TEST DESIGN

With the component XML file validated and all the filesets packaged in the proper component libraries (VOLIPository and bbcLib), we can now verify the behavior of the integrated IP core in the *Platform Express* environment.

1. From the command prompt, invoke *PX* by issuing the following command:

```
$PXHOME/bin/px -refresh &
```

   Alternately, if *PX* is already running, select **File > Refresh Libraries** to reload newly added libraries without exiting.

2. Notice that the installed component libraries are not visible in the Component Browser yet. This is because initially, *PX* only shows components (CPU cores) that could be dragged onto the Design Editor Pane. Perform a drag-n-drop on the **Leon2 Processor**. Go through the resulting **Configure mcore_ahb** window (shown in *Figure 4.5-1*) and click **OK**. Leon2 CPU core will now appear in the Design Editor; simultaneously *PX* will update the Component Browser as shown in *Figure 4.5-2*.

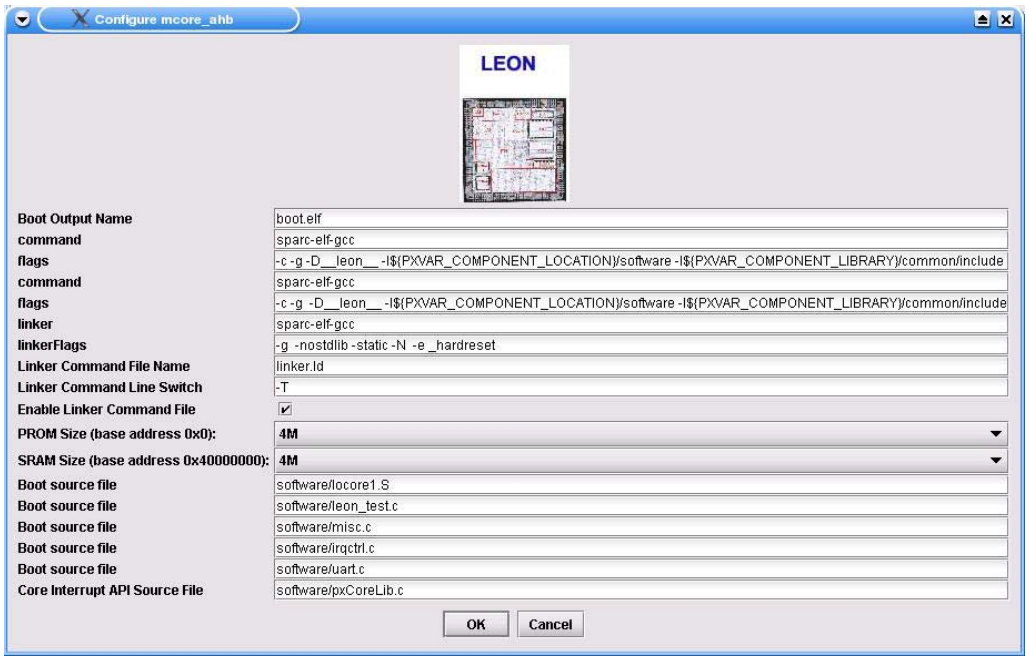


Figure 4.5-1: mcore_ahb Configuration Window

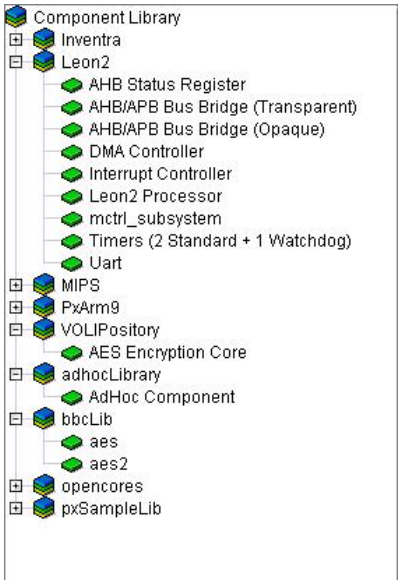


Figure 4.5-2: Updated Component Browser

3. Select **aes2** (aes, in your case) under **bbcLib** and drag it onto the **ambaAHB_1** master bus of the Leon2 CPU. As shown in *Figure 4.5-3*, a **Bus Bridge Required** window will come up. This is to bridge the connection between the AHB master of Leon2 with APB slave of AES. Select the opaque bus bridge, **Leon2 – apbmst_obb** for this purpose.
4. Leave the values unchanged in the **Configure apbmst_obb** dialog box shown in *Figure 4.5-4* and click **OK**.
5. Do the same for the **singlePinInterruptBus_1** configuration dialog box shown in *Figure 4.5-5*.
6. At this point the AES core will appear in the Design Editor as shown in *Figure 4.5-6*. Attach the **ambaAHB_2** master bus on the AES to the **ambaAHB_1** master bus of Leon2 by performing a drag-n-drop operation. This way a non-processor component like AES can have access to the master bus. The resulting change is shown in *Figure 4.5-7*
7. Click  to save the design under <savedProject> directory and then click  to build it. Click **OK** on the resulting **Required Configuration** window and watch the Output Pane for

Figure 4.5-3: AHB-APB Bus Bridge Dialog Box

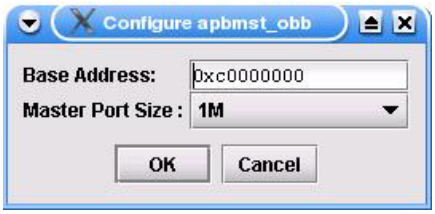


Figure 4.5-4: Bus Bridge Configuration

Figure 4.5-5: Interrupt Bus Configuration

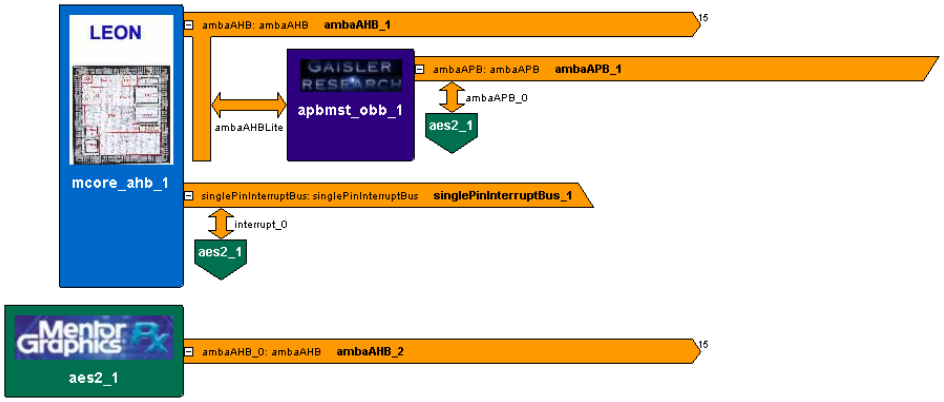


Figure 4.5-6: Leon2-AES Before AHB Bus Attachment

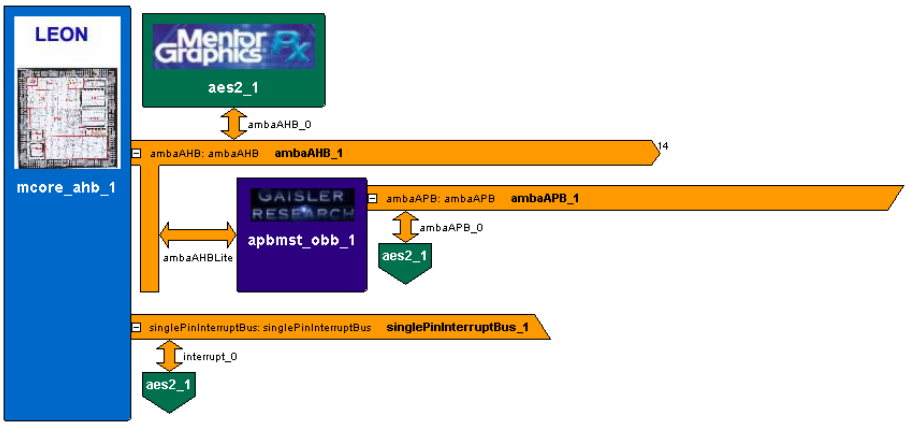


Figure 4.5-7: Leon2-AES After AHB Bus Attachment

build status messages. During this 'build' process, *PX* will generate an HDL system design based on the contents of the Design Editor and the component configurations.

Note: Performing a build is not possible without proper software compilation tools. For instance, Leon2 core requires the sparc-elf-gcc cross-compiler for compiling the boot code (Refer to *Appendix C* for information on building a sparc-elf-gcc cross-compiler for *PX* running on Solaris machines). Also, all configurable settings can be accessed by selecting **Settings > Configure All**. For more information on various configure options available to system designers, please refer to the *Platform Express User's Guide*. [23]

8. The build process generates a savedProject.plx file inside the <savedProject> directory along with other sub-directories to hold the HDL source files of the design, object files, build scripts and configuration files. The directory structure of the saved project is shown in *Figure 4.5-8*. Once the build process is completed, the last few lines of a successful build will be identical to those depicted in *Figure 4.5-9*. Refer to *Appendix D* for the detailed build log.

```
<savedProject>
      +- verificationEnv
      |      |
      |      +- Modelsim
      |            +- designData
      |            +- exec [Generated files for Seamless execution]
      |            +- hdl [Generated HDL files, build scripts and compiled
      |            |      models]
      |            +- software [Contains compiled object files and source
      |            |      code for the main diagnostics file]
      |            +- testbench
      |            |- pxenv.properties
      |            |- pxenv.sh [PX_HDL_BUILD Directory Specifications]
      |- <savedProject>.plx
```

Figure 4.5-8: Directory Structure of Generated Design

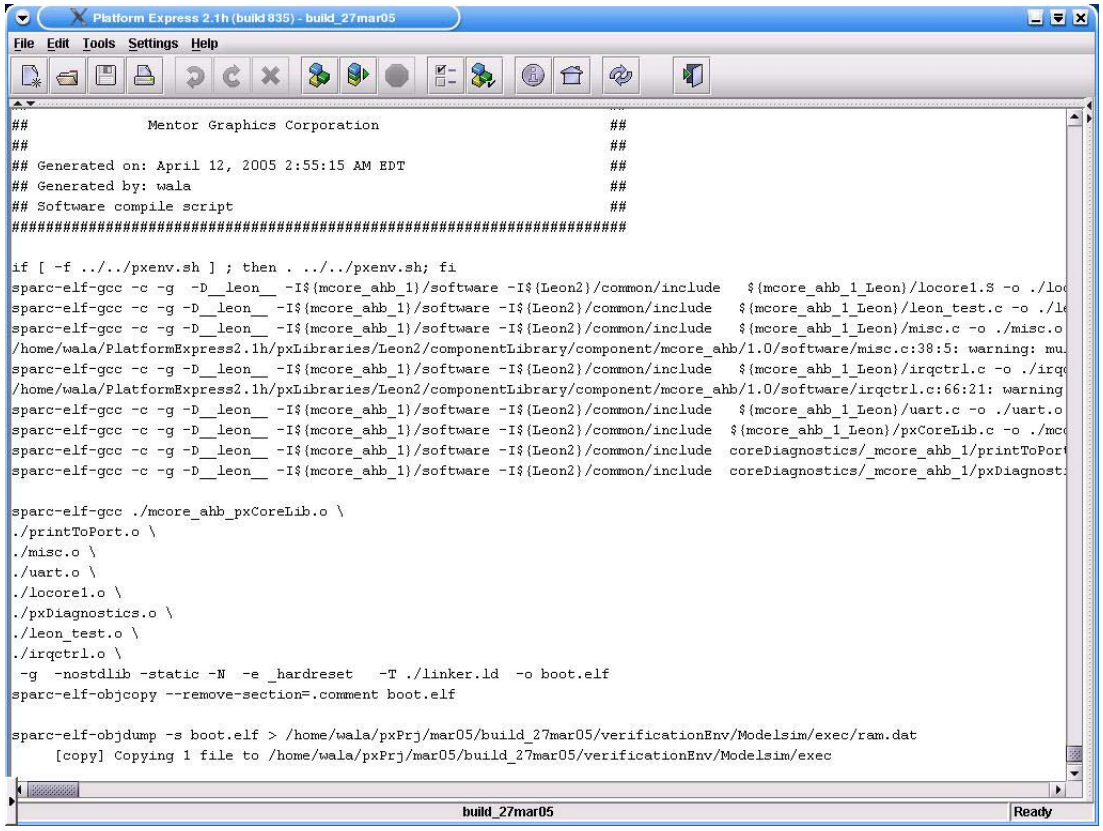


Figure 4.5-9: Output Pane Build Log Messages

9. Click  to execute the build and invoke the *Seamless CVE* session. This will bring up the *ModelSim* application and its *Wave Viewer* interface (shown in *Figure 4.5-10*).

The next section explains the process of generating a build for designs featuring a black box component (see *Section 4.4.6*) and can be skipped if your test design consists of IPs described using a single HDL.

4.5.1 Building Designs Featuring Black Box IPs

Generating a black box component in a manner described in *Section 4.4.6* results in a top-level bbc_top.vhd file in the hdlSrc directory of the bbcLib component directory. This file is essentially an AMBA-compliant wrapper. Follow the steps below to successfully build a design consisting of IPs described in multiple HDLs.

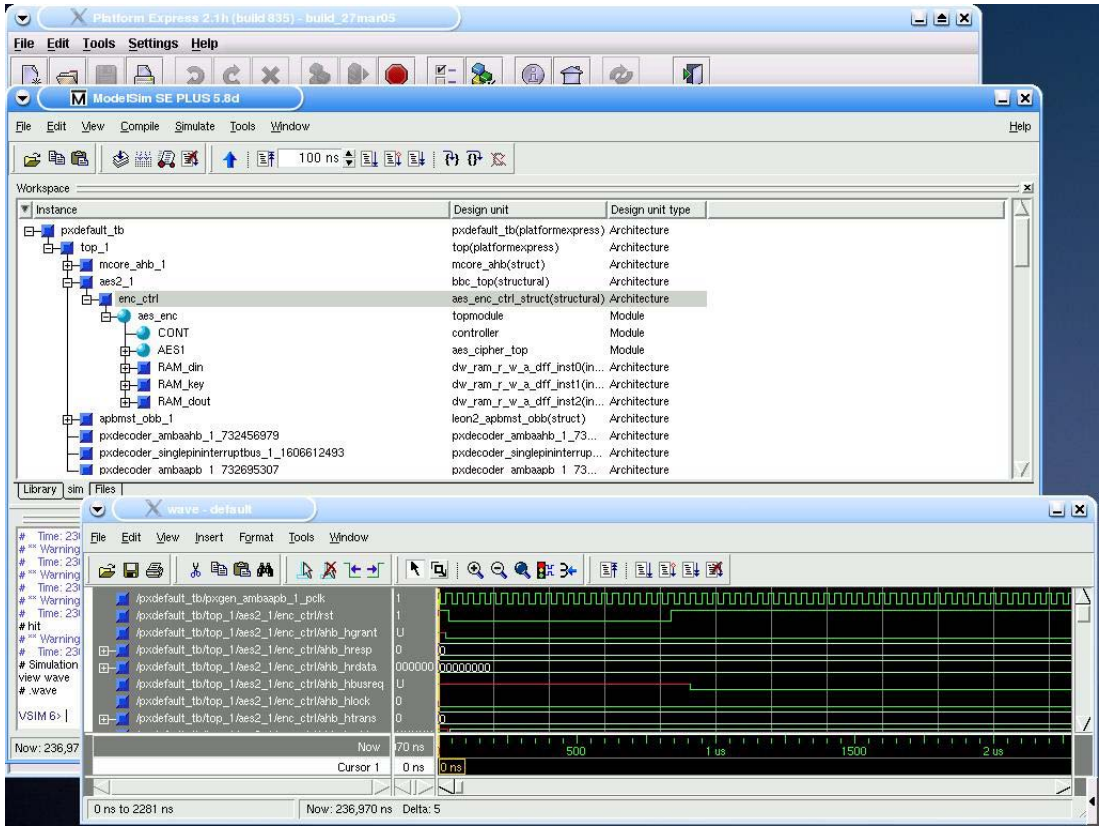


Figure 4.5-10: Seamless Environment – *ModelSim* Application with *Waveform Viewer*

1. Move all the HDL files (VHDL as well as Verilog) into the hdlSrc directory of the bbcLib component library.

2. Modify the architecture declaration in the bbc_top.vhd file to include port-mapping statements to the lower-level aes_enc_ctrl_struct.vhd file. Leave the architecture declaration of the aes_enc_ctrl_struct.vhd file empty. This is to hide the Verilog files instantiation declared inside aes_enc_ctrl_struct.vhd from the *PX* environment.

3. Modify the aes.xml file under pxLibraries/bbcLib/component-Library/component/aes/1.0 using *PxEdit* or a text editor to include aes_enc_ctrl_struct.vhd under hwmodel and filesets. For reference, use the XML file generated in *Section 4.4* or the one included in *Appendix B*.

4. Create and build the design as described in *Section 4.5*, steps 1—8. Do not execute the build yet.

5. From the terminal window, navigate to the <savedProject>/verificationEnv /Modelsim/hdl directory and modify the *PX*-generated build.xml script. Insert "vlog" statements to allow compilation of the previously hidden Verilog files. *Figure 4.5-11* shows the pre- and post-modification snapshots of the build.xml script.

6. Modify the aes_enc_ctrl_struct.vhd by adding statements in the previously empty architecture declaration.

7. Under the same <savedProject>/verificationEnv/Modelsim/hdl directory, type **ant**. 'Ant' [24] is the Java equivalent of the 'Make' command and uses XML-based configuration files to execute tasks. In this case, *ant* will be using the modified build.xml script to compile all the HDL files in our design.

8. After executing all the instructions in the build script, *ant* will generate a message similar to:

```
BUILD SUCCESSFUL
Total time: 31 seconds
```

   Note: Errors in the build script will result in a "Build Failed" message.

9. Go back to the *PX* application and complete step-10 as described in *Section 4.5*.

This completes our one design cycle using *Platform Express*. Users can examine the design, make further changes and enhancements and cycle through the design flow as explained in this chapter.

```
<description>Build Platform Express component library.</description>
<property environment="env" />
<property file="../pxenv.properties" />
- <target name="compile" description="Compile hdl sources into a library.">
  <mkdir dir="work" />
  - <exec executable="vlib">
      <arg value="work/wala_userlibrary_aes_25mar05_UT_bbcLib_aes2_1" />
    </exec>
  <mkdir dir="work" />
  - <exec executable="vlib">
      <arg value="work/wala_userlibrary_aes_25mar05" />
    </exec>
  <mkdir dir="work" />
  - <exec executable="vlib">
      <arg value="work/wala_userlibrary_aes_25mar05_Mentor_Leon2_mcore_ahb_1" />
    </exec>
  <mkdir dir="../designData/libraryData/Mentor_Leon2" />
  - <exec executable="vlib">
      <arg value="../designData/libraryData/Mentor_Leon2/l
    </exec>
  <mkdir dir="work" />
  - <exec executable="vlib">
      <arg value="work/leon2_apbmst_obb" />
    </exec>
  <mkdir dir="work" />
  - <exec executable="vlib">
      <arg value="work/pxdefault_tb" />
    </exec>
  - <exec executable="vcom" failonerror="true">
      <env key="PX_HDL_BUILD" value="${PX_HDL_BUILD}" />
      <arg value="-work" />
      <arg value="work/wala_userlibrary_aes_25mar05_Ment
      <arg value="-93" />
```

a

```
  <exec executable="vcom" failonerror="true">
    <env key="PX_HDL_BUILD" value="${PX_HDL_BUILD}" />
    <env key="PX_HDL_BUILD_2" value="${PX_HDL_BUILD_2}" />
    <env key="PX_HDL_BUILD_3" value="${PX_HDL_BUILD_3}" />
    <env key="PX_HDL_BUILD_1" value="${PX_HDL_BUILD_1}" />
    <arg value="-work" />
    <arg value="work/wala_userlibrary_aes_25mar05_UT_bbcLib_aes2_1" />
    <arg value="${PX_HDL_BUILD_3}/DW_ram_r_w_a_dff_inst.vhd" />
  </exec>
- <exec executable="vlog" failonerror="true">
    <env key="PX_HDL_BUILD" value="${PX_HDL_BUILD}" />
    <env key="PX_HDL_BUILD_2" value="${PX_HDL_BUILD_2}" />
    <env key="PX_HDL_BUILD_3" value="${PX_HDL_BUILD_3}" />
    <env key="PX_HDL_BUILD_1" value="${PX_HDL_BUILD_1}" />
    <arg value="-work" />
    <arg value="work/wala_userlibrary_aes_25mar05_UT_bbcLib_aes2_1" />
    <arg value="${PX_HDL_BUILD_3}/timescale.v" />
    <arg value="${PX_HDL_BUILD_3}/aes_rcon.v" />
    <arg value="${PX_HDL_BUILD_3}/aes_sbox.v" />
    <arg value="${PX_HDL_BUILD_3}/aes_key_expand_128.v" />
    <arg value="${PX_HDL_BUILD_3}/aes_cipher_top.v" />
    <arg value="${PX_HDL_BUILD_3}/controller.v" />
    <arg value="${PX_HDL_BUILD_3}/topmodule.v" />
  </exec>
```

b

Figure 4.5-11: The build.xml file before Modification (a); Modified Build File with Verilog Compile Instructions (b)

4.6 VERIFYING THE DESIGN

Currently, only a few processors are supported by *Seamless* in *PX*. The ARM's Processor Support Package (PSP) and the MIPS PSP are supported but not Leon. Therefore, as of now, due to the unavailability of a *Seamless* model of the Leon processor, simulation/optimization of Leon CPU core is not possible using *Seamless* application. This means that at present, the test design can only be simulated and verified for correctness using a hardware simulator such as the *ModelSim* application.

5

CONCLUSIONS

Being a pioneer is non-trivial.
--Don Bouldin

Results! Why man, I have gotten a lot of results. I know a several thousand things that won't work.
--Thomas A. Edison

The primary objective of the *Volunteer SoC* project is to allow designers to be able to reuse their current design by having it as a starting point for their future work. In this task we explored the possibility of using *Platform Express* to quickly generate a platform for our future SoC designs. The choice of this tool proved to be not just right but also very appropriate. In addition to allowing designers to rapidly create system designs, *PX* also enables IP developers to showcase their components for possible use in that design.

5.1 CONTRIBUTIONS

In our attempt at using the *PX* application for the first time at the University of Tennessee, we overcame some minor as well as a few major issues and were successful in implementing a processor-centric platform subsystem for derivative designs. The complete IP integration and platform design flow, illustrated in *Figure 4.1-1 (Chapter 4)*, was followed while using *Platform Express* for the SoC platform design.

The detailed explanation of IP installation and platform building process is given in this thesis and it is intended for use as an instructional guide for students at our university.

One of the main problems encountered during this project was the unavailability of the Solaris build of the sparc-elf-gcc cross-compiler for Leon2 CPU. This compiler has been built at our university and copies are available for download [25]. Additionally, one copy sent to Gaisler Research is available for download, while the other sent to Mentor Graphics is for their internal use to assist in problems involving the Leon2 CPU core.

5.2 CURRENT STATUS AND FUTURE WORK

At the time of writing this thesis, the platform building process and functional verification of the platform using the *ModeSim* hardware simulator has been completed. With the availability of the Leon2 Seamless model, co-simulation of both the hardware and software components of the design will be possible.

One of the tasks in the near future can be to add more IP cores to the existing *VOLIPository* component library and enhance the existing platform. Another possibility is to use the recently acquired Virtex II™ series FPGAs to prototype the platform-based SoCs designed using *PX*.

REFERENCES

[1] Moore, G., "Cramming More Components onto Electronic Circuits". [Online]. Available: ftp://download.intel.com/research/silicon/moorespaper.pdf

[2] Bouldin, D., "Platform-based System-on-Chip Design," *Proceedings of 2003 NASA Symposium on VLSI Design,* Cour d'Alene, ID, pp.1-4, May 28-29, 2003.

[3] Bouldin, D., Microelectronic Systems Courses, University of Tennessee. http://vlsi1.engr.utk.edu/ece/bouldin_courses/

[4] Bouldin, D., and 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.

[5] Mentor Graphics Corporation. http://www.mentor.com/platform_ex/

[6] Gaisler Research. http://www.gaisler.com/

[7] AMBA 2.0 Specification. http://www.gaisler.com/doc/amba.pdf

[8] Weiss, R., "Advanced Microprocessor Bus Architecture (AMBA) Bus Sysem," *Electronic Design*, March 2001. [Online]. Available: http://www.elecdesign.com/Articles/ArticleID/4165/4165.html

[9] Chandra, R., "IP-Reuse and Platform-based Design," *D&R Industry Articles*, August 2003, [Online] Available: http://www.us.design-reuse.com/articles/article6125.html

[10] Sangiovanni-Vincentelli, A., "The Tides of EDA," *IEEE Design & Test of Computers*, pp. 59-75, November-December 2003

[11] Fritz, D., "Why Platform-based Design works better than discrete IP approach," *Portable Design*, October 2003, [Online] Available: http://pd.pennnet.com/Articles/Article_Display.cfm?Section=Articles&Subsection=Display&ARTICLE_ID=189021

[12] Borel, J., "Design Automation in MEDEA: Present and Future," *IEEE Micro*, Vol. 19, No. 5, pp. 71-79, September 1999.

[13] Billie, A., and S. Hatliff, "Platform approach speeds MIPS-based SoCs," *EEDesign*, October 2001, [Online]. Available: http://www.eetimes.com/story/OEG20011012S0-076

[14] Bouldin, D., "Platform-based System-on-Chip Design," *Proceedings of 2003 NASA Symposium on VLSI Design,* Cour d'Alene, ID, pp.1-4, May 28-29, 2003.

[15] Tarverdians, F., "A Platform-based Design can reduce DSC design time and cost," *Portable Design,* March 2004, [Online] Available: http://pd.pennnet.com/Articles/Article_Display.cfm?Section=Archives&Subsection=Display&ARTICLE_ID=199976&KEYWORD=ARC

[16] Altizer, R., "Platform-based Design: The Next Reuse Frontier," *Embedded Systems Conference,* San Francisco, CA, March 14, 2002.

[17] Zaidi, J., "Different platform types are needed for SoC Design," *EEDesign,* January 2003, [Online] Available: http://eetimes.com/story/OEG20030131S0057

[18] Schirrmeister, F., and G. Martin, "A Design Chain for Embedded Systems," *IEEE Computer,* Embedded Systems Column, pp. 100-103, March 2002.

[19] Kuetzer, K., A. R. Newton, J.M. Rabaey, A. Sangiovanni-Vincentelli, "System Level Design: Orthogonalization of Concerns and Platform based Design," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol.19, No.12, pp. 1523-1543, December 2000.

[20] AES Algorithm (Rijndael)Information, *National Institute of Standards and Technology, Computer Security Division*. [Online] Available: http://csrc.nist.gov/CryptoToolkit/aes/rijndael/

[21] SPIRIT Consortium, http://www.spiritconsortium.com/

[22] Mentor Graphics Corporation, "Platform Express Integrator's Guide," July 2004 [Online] Available: http://www.mentor.com/products/embedded_software/platform_baseddesign/platform_express/upload/pxCompIntGuide.pdf

[23] Mentor Graphics Corporation, "Platform Express User's Guide," July 2004 [Online] Available: http://www.mentor.com/products/embedded_software/platform_baseddesign/platform_express/upload/px_help.pdf

[24] The Apache Ant Project, http://ant.apache.org/

[25] Sparc-elf-gcc for Solaris, http://vlsi1.engr.utk.edu/~wala/sparc-elf-gcc.html

APPENDICES

Appendix A

VHDL SOURCE CODE LISTING

AES.VHD

```
--+-----------------------------------------------------------+
--| Module: AES.VHD                                           |
--|         Top level AMBA AHB/APB wrapper                    |
--|                                                           |
--| Modified by: Mardav Wala [mardav.wala@gmail.com]          |
--|                                                           |
--| Project: Using Platform Express for System-on-Chip Design |
--+-----------------------------------------------------------+

---
-- Standard Libraries
---
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_signed.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY aes IS
  PORT(
        HRESETn         :IN STD_LOGIC;
        HCLK            :IN STD_LOGIC;
        HGRANT          :IN STD_ULOGIC;
        HREADY          :IN STD_ULOGIC;
        HRESP           :IN STD_LOGIC_VECTOR(1 downto 0);
        HRDATA          :IN STD_LOGIC_VECTOR(31 downto 0);

        HBUSREQ         :OUT STD_ULOGIC;
        HLOCK           :OUT STD_ULOGIC;
        HTRANS          :OUT STD_LOGIC_VECTOR(1 downto 0);
        HADDR           :OUT STD_LOGIC_VECTOR(31 downto 0);
        HWRITE          :OUT STD_ULOGIC;
        HSIZE           :OUT STD_LOGIC_VECTOR(2 downto 0);
        HBURST          :OUT STD_LOGIC_VECTOR(2 downto 0);
        HPROT           :OUT STD_LOGIC_VECTOR(3 downto 0);
        HWDATA          :OUT STD_LOGIC_VECTOR(31 downto 0);

        PSELx           :IN STD_ULOGIC;
        PENABLE         :IN STD_ULOGIC;
        PADDR           :IN STD_LOGIC_VECTOR(31 downto 0);
        PWRITE          :IN STD_ULOGIC;
        PWDATA          :IN STD_LOGIC_VECTOR(31 downto 0);

        PRDATA          :OUT STD_LOGIC_VECTOR(31 downto 0);
        irq             :OUT STD_LOGIC
        );
END aes;

ARCHITECTURE rtl OF aes IS

  COMPONENT aes_enc_ctrl
  PORT(
        RST             :IN STD_LOGIC;
        CLK             :IN STD_LOGIC;
        AHB_HGRANT      :IN STD_ULOGIC;
        AHB_HREADY      :IN STD_ULOGIC;
        AHB_HRESP       :IN STD_LOGIC_VECTOR(1 downto 0);
        AHB_HRDATA      :IN STD_LOGIC_VECTOR(31 downto 0);
```

```
        AHB_HBUSREQ    :OUT STD_ULOGIC;
        AHB_HLOCK      :OUT STD_ULOGIC;
        AHB_HTRANS     :OUT STD_LOGIC_VECTOR(1 downto 0);
        AHB_HADDR      :OUT STD_LOGIC_VECTOR(31 downto 0);
        AHB_HWRITE     :OUT STD_LOGIC;
        AHB_HSIZE      :OUT STD_LOGIC_VECTOR(2 downto 0);
        AHB_HBURST     :OUT STD_LOGIC_VECTOR(2 downto 0);
        AHB_HPROT      :OUT STD_LOGIC_VECTOR(3 downto 0);
        AHB_HWDATA     :OUT STD_LOGIC_VECTOR(31 downto 0);

        APB_PSEL       :IN STD_ULOGIC;
        APB_PENABLE    :IN STD_ULOGIC;
        APB_PADDR      :IN STD_LOGIC_VECTOR(31 downto 0);
        APB_PWRITE     :IN STD_ULOGIC;
        APB_PWDATA     :IN STD_LOGIC_VECTOR(31 downto 0);

        APB_PRDATA     :OUT STD_LOGIC_VECTOR(31 downto 0);
        irq            :OUT STD_LOGIC
        );
END COMPONENT;

BEGIN
AES: aes_enc_ctrl_struct PORT MAP(
        RST            => HRESETn,
        CLK            => HCLK,
        AHB_HGRANT     => HGRANT,
        AHB_HREADY     => HREADY,
        AHB_HRESP      => HRESP,
        AHB_HRDATA     => HRDATA,

        AHB_HBUSREQ    => HBUSREQ,
        AHB_HLOCK      => HLOCK,
        AHB_HTRANS     => HTRANS,
        AHB_HADDR      => HADDR,
        AHB_HWRITE     => HWRITE,
        AHB_HSIZE      => HSIZE,
        AHB_HBURST     => HBURST,
        AHB_HPROT      => HPROT,
        AHB_HWDATA     => HWDATA,

        APB_PSEL       => PSELx,
        APB_PENABLE    => PENABLE,
        APB_PADDR      => PADDR,
        APB_PWRITE     => PWRITE,
        APB_PWDATA     => PWDATA,

        APB_PRDATA     => PRDATA,
        irq            => irq
        );
END;
```

AES_ENC_CTRL_STRUCT.VHD

```
--+-----------------------------------------------------------------------
--| File: AES_ENC_CTRL_STRUCT.VHD
--|  Describes the AMBA AHB/APB bus-compatible controller module for the
--|  AES Rijndael encryption IP core available from OpenCores.org
--|  Based on the MDCT.VHD module described in the Ogg-on-a-Chip Project
--|  by Luis Azuara. [http://oggonachip.sourceforge.net]
--|
--| Modified by: Rishi Srivastava
--|
--| Revised by: MARDAV WALA [mardav@gmail.com]
--|
--| Project: Using Platform Express for System-on-Chip Design
--|  [MS Thesis. University of Tennessee, May 2005]
--|
--| Sub-Files: AES_RCON.V, AES_SBOX.V, AES_KEY_EXPAND_128.V,
--|  AES_CIPHER_TOP.V, DW_RAM.VHD, CONTROLLER.V, TOPMODULE.V
--+-----------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_signed.ALL;
USE IEEE.std_logic_arith.ALL;
--USE mywork.iface.ALL;

----------------------
ENTITY aes_enc_ctrl_struct IS
----------------------
  PORT(
      RST         :IN STD_LOGIC;
      CLK         :IN STD_LOGIC;

      -- AHB Bus Signals
      AHB_HGRANT  :IN STD_ULOGIC;
      AHB_HREADY  :IN STD_ULOGIC;
      AHB_HRESP   :IN STD_LOGIC_VECTOR(1 downto 0);
      AHB_HRDATA  :IN STD_LOGIC_VECTOR(31 downto 0);

      AHB_HBUSREQ :OUT STD_ULOGIC;
      AHB_HLOCK   :OUT STD_ULOGIC;
      AHB_HTRANS  :OUT STD_LOGIC_VECTOR(1 downto 0);
      AHB_HADDR   :OUT STD_LOGIC_VECTOR(31 downto 0);
      AHB_HWRITE  :OUT STD_LOGIC;
      AHB_HSIZE   :OUT STD_LOGIC_VECTOR(2 downto 0);
      AHB_HBURST  :OUT STD_LOGIC_VECTOR(2 downto 0);
      AHB_HPROT   :OUT STD_LOGIC_VECTOR(3 downto 0);
      AHB_HWDATA  :OUT STD_LOGIC_VECTOR(31 downto 0);

      -- APB Bus Signals
      APB_PSEL    :IN STD_ULOGIC;
      APB_PENABLE :IN STD_ULOGIC;
      APB_PADDR   :IN STD_LOGIC_VECTOR(31 downto 0);
      APB_PWRITE  :IN STD_ULOGIC;
      APB_PWDATA  :IN STD_LOGIC_VECTOR(31 downto 0);

      APB_PRDATA  :OUT STD_LOGIC_VECTOR(31 downto 0);

      -- Single Pin Interrupt Bus Signal
      irq         :OUT STD_LOGIC
      );
END aes_enc_ctrl_struct;
```

```
---------------------------------------------
ARCHITECTURE structural of aes_enc_ctrl_struct IS
---------------------------------------------
-- Memory Map
-- ADDRESS        NAME                                DESCRIPTION
-- 0x80000300     Control Register                    Bit 0: AES Core On/Off
--                                                     Bit 2: IRQ enabled/disabled
--                                                     Bit 3: IRQ request
-- 0x80000304     Key/Plain Text Read Start Address  32 bits: 0x40000000
-- 0x80000308     Cipher Text Write Start Address    32 bits: 0x40001000
-- 0x8000030C     Status Register                     Bit 0: Ready/Busy
--                                                     Bit 1: Reading/Writing
--                                                     (READ ONLY)
-- 0x80000310     Current Memory Register            32 bits: Actual DMA Address
--                                                     (READ ONLY)

  ---
  -- TOPMODULE.V
  ---
  COMPONENT topmodule
  PORT(
    inst_test_clk :IN STD_LOGIC;
    rst           :IN STD_LOGIC;
    inst_rst_n    :IN STD_LOGIC;
    in_cs_n       :IN STD_LOGIC;
    in_cs_n1      :IN STD_LOGIC;
    Go            :IN STD_LOGIC;
    inst_wr_n     :IN STD_LOGIC;
    inst_wr_addr  :IN STD_LOGIC_VECTOR(3 downto 0);
    inst_rd_addr1 :IN STD_LOGIC_VECTOR(3 downto 0);
    inst_data_in  :IN STD_LOGIC_VECTOR(31 downto 0);
    inst_key_in   :IN STD_LOGIC_VECTOR(31 downto 0);

    data_out_inst1  :OUT STD_LOGIC_VECTOR(31 downto 0)
    );
  END COMPONENT;

  ---
  -- AES Record Signals
  ---
  TYPE aes_regs IS RECORD
  -- Memory Mapped Registers
  -- Control Register: 0x80000300
  aes_en_req  :STD_LOGIC; -- Bit 0: AES core enabled if '1', disabled if '0'
  irq_en      :STD_LOGIC; -- Bit 2: IRQ enabled if '1', disabled if '0'
  irq         :STD_LOGIC; -- Bit 3: IRQ request generated if '1', not if '0'

  -- Read Memory Transfer Address: 32 bit at 0x80000304
  rd_start_addr :STD_LOGIC_VECTOR(31 downto 0);

  -- Write Memory Transfer Address: 32 bit at 0x80000308
  wr_start_addr :STD_LOGIC_VECTOR(31 downto 0);

  -- Status Register: 32 bit at 0x8000030C
  ready       :STD_LOGIC; -- Bit 0: Function done if '1', busy if '0' /Read Only
  mem_wr      :STD_LOGIC; -- Bit 1: Writing if '1', reading if '0' / Read Only

  -- Current Memory Register: 32 bit at 0x80000310
  mem_addr    :STD_LOGIC_VECTOR(31 downto 0);
  -- End Memory Mapped Registers

  -- Internal Registers
  wr_n          :STD_LOGIC;
  dma_xfer_req  :STD_LOGIC;
  aes_en        :STD_LOGIC;
  n             :STD_LOGIC_VECTOR(2 downto 0);
  data_in   :STD_LOGIC_VECTOR(31 downto 0);
  key_in    :STD_LOGIC_VECTOR(31 downto 0);
  aes_addr  :STD_LOGIC_VECTOR(3 downto 0);
```

```
  -- AMBA Status Registers
  bus_active  :STD_LOGIC;
  bus_own     :STD_LOGIC;
  bus_grant   :STD_LOGIC;
  END RECORD;
  ---
  -- Constants
  ---
  -- HTRANS (Transfer Type  | Output from the AHB MASTER)
  CONSTANT HTRANS_IDLE    :STD_LOGIC_VECTOR(1 downto 0) := "00";
  CONSTANT HTRANS_BUSY    :STD_LOGIC_VECTOR(1 downto 0) := "01";
  CONSTANT HTRANS_NONSEQ  :STD_LOGIC_VECTOR(1 downto 0) := "10";
  CONSTANT HTRANS_SEQ     :STD_LOGIC_VECTOR(1 downto 0) := "11";

  -- HBURST (Address Increments | Output from the AHB MASTER)
  CONSTANT HBURST_SINGLE  :STD_LOGIC_VECTOR(2 downto 0) := "000";
  CONSTANT HBURST_INCR    :STD_LOGIC_VECTOR(2 downto 0) := "001";
  CONSTANT HBURST_WRAP4   :STD_LOGIC_VECTOR(2 downto 0) := "010";
  CONSTANT HBURST_INCR4   :STD_LOGIC_VECTOR(2 downto 0) := "011";
  CONSTANT HBURST_WRAP8   :STD_LOGIC_VECTOR(2 downto 0) := "100";
  CONSTANT HBURST_INCR8   :STD_LOGIC_VECTOR(2 downto 0) := "101";
  CONSTANT HBURST_WRAP16  :STD_LOGIC_VECTOR(2 downto 0) := "110";
  CONSTANT HBURST_INCR16  :STD_LOGIC_VECTOR(2 downto 0) := "111";

  -- HSIZE (Transfer Size | Output from the AHB MASTER)
  CONSTANT HSIZE_BYTE   :STD_LOGIC_VECTOR(2 downto 0) := "000";
  CONSTANT HSIZE_HWORD  :STD_LOGIC_VECTOR(2 downto 0) := "001";
  CONSTANT HSIZE_WORD   :STD_LOGIC_VECTOR(2 downto 0) := "010";
  CONSTANT HSIZE_DWORD  :STD_LOGIC_VECTOR(2 downto 0) := "011";
  CONSTANT HSIZE_4WORD  :STD_LOGIC_VECTOR(2 downto 0) := "100";
  CONSTANT HSIZE_8WORD  :STD_LOGIC_VECTOR(2 downto 0) := "101";
  CONSTANT HSIZE_16WORD :STD_LOGIC_VECTOR(2 downto 0) := "110";
  CONSTANT HSIZE_32WORD :STD_LOGIC_VECTOR(2 downto 0) := "111";

  -- HRESP (Transfer Response | Output from the AHB SLAVE)
  CONSTANT HRESP_OKAY   :STD_LOGIC_VECTOR(1 downto 0) := "00";
  CONSTANT HRESP_ERROR  :STD_LOGIC_VECTOR(1 downto 0) := "01";
  CONSTANT HRESP_RETRY  :STD_LOGIC_VECTOR(1 downto 0) := "10";
  CONSTANT HRESP_SPLIT  :STD_LOGIC_VECTOR(1 downto 0) := "11";

  ---
  -- Signals / Registers
  ---
  SIGNAL r, tmp   :aes_regs;
  SIGNAL sig_dataRdy  :STD_LOGIC;
  SIGNAL sig_finish   :STD_LOGIC;
  SIGNAL sig_mem_wr   :STD_LOGIC;
  SIGNAL sig_rst_n    :STD_LOGIC;
  SIGNAL sig_cs_n     :STD_LOGIC;
  SIGNAL sig_cs_n1    :STD_LOGIC;
  SIGNAL sig_wr_n     :STD_LOGIC;
  SIGNAL sig_Go       :STD_LOGIC;
  SIGNAL sig_key_in   :STD_LOGIC_VECTOR(31 downto 0);
  SIGNAL sig_data_in  :STD_LOGIC_VECTOR(31 downto 0);
  SIGNAL sig_data_out :STD_LOGIC_VECTOR(31 downto 0);
  SIGNAL sig_addr_in  :STD_LOGIC_VECTOR(3 downto 0);
  SIGNAL sig_addr_out :STD_LOGIC_VECTOR(3 downto 0);

  SIGNAL sig_HADDR    :STD_LOGIC_VECTOR(31 downto 0);
  SIGNAL sig_HTRANS   :STD_LOGIC_VECTOR(1 downto 0);
  SIGNAL sig_HWRITE   :STD_LOGIC;
  SIGNAL sig_HWDATA   :STD_LOGIC_VECTOR(31 downto 0);
  SIGNAL sig_HBUSREQ  :STD_LOGIC;

  SIGNAL sig_PRDATA   :STD_LOGIC_VECTOR(31 downto 0);

  TYPE state_type IS (idle, bus_req, bus_grant, bus_own, load_key, load_text,
xfer_end);
```

```
  SIGNAL cstate, nstate  :state_type;
  ---
  --
  ---
  BEGIN
  AES_ENC:  topmodule PORT MAP(
    data_out_inst1  => sig_data_out,
    inst_rst_n      => sig_rst_n,
    inst_wr_n       => sig_wr_n,
    inst_test_clk   => CLK,
    inst_rd_addr1   => sig_addr_out,
    inst_wr_addr    => sig_addr_in,
    inst_data_in    => sig_data_in,
    inst_key_in     => sig_key_in,
    Go              => sig_Go,
    rst             => RST,
    in_cs_n         => sig_cs_n,
    in_cs_n1        => sig_cs_n1
    );

  state_reg: process(CLK, RST, nstate)
  BEGIN
    IF (RST = '0') THEN
      cstate  <= idle;
    ELSIF (CLK'event and CLK = '1') THEN
      cstate  <= nstate;
    END IF;
  END PROCESS; -- state_reg

  the_process:  PROCESS(CLK, RST, APB_PSEL, APB_PENABLE, APB_PADDR, APB_PWRITE,
APB_PWDATA, cstate, sig_dataRdy, AHB_HREADY, AHB_HGRANT, AHB_HRDATA, AHB_HRESP)
  BEGIN
  IF (CLK'event and CLK = '1') THEN
    tmp <= r;

    IF (RST = '0') THEN     -- Asynchronous Reset
      sig_finish    <= '0';
      sig_mem_wr    <= '0';
      sig_rst_n     <= '0';
      sig_cs_n      <= '0';
      sig_cs_n1     <= '0';
      sig_wr_n      <= '0';
      sig_Go        <= '0';
      sig_key_in    <= (others => '0');
      sig_data_in   <= (others => '0');
      sig_addr_in   <= (others => '0');
      sig_addr_out  <= (others => '0');
      sig_HADDR     <= (others => '0');
      sig_HTRANS    <= (others => '0');
      sig_HWRITE    <= '0';
      sig_HWDATA    <= (others => '0');
      sig_dataRdy   <= '1';
    ELSE
      sig_rst_n     <= '1';
      sig_cs_n      <= '0';
      sig_cs_n1     <= '0';
      sig_wr_n      <= tmp.wr_n;
      sig_PRDATA    <= (others => '0');

 ---------------------
 -- APB Bus Conditions
 ---------------------

      IF (APB_PSEL and APB_PENABLE and APB_PWRITE) = '1' THEN
      -- Write the PWDATA to the registers depending on the PADDR bus contents
        CASE APB_PADDR(4 downto 2) IS
          WHEN "000" =>
          -- PADDR = 0x80000300
            tmp.aes_en_req  <= APB_PWDATA(0);
```

```
            tmp.irq_en      <= APB_PWDATA(2);
            IF (APB_PWDATA(3) = '0') THEN
              tmp.irq <= '0';                -- Allow IRQ Reset only
            END IF;
            IF (tmp.aes_en_req = '1' and r.aes_en_req = '0' and r.ready = '1') THEN
            -- Initialize AES transaction when ENABLED and READY
              tmp.aes_en    <= '1';          -- Enable AES core
              tmp.mem_addr  <= X"40000000";  -- Initial memory read address is
0x40000000
              tmp.mem_wr    <= '0';          -- Start Read cycle
              tmp.ready     <= '0';
              tmp.wr_n      <= '0';
              tmp.aes_addr  <= (others => '0');
              sig_finish    <= '0';
              sig_HTRANS    <= HTRANS_NONSEQ; -- First transaction is ALWAYS non-
sequential
              --cstate        <= bus_req;       -- Request bus for transaction
            END IF;
          WHEN "001" =>
          -- PADDR = 0x80000304
            tmp.rd_start_addr <= APB_PWDATA;
          WHEN "010" =>
          -- PADDR = 0x80000308
            tmp.wr_start_addr <= APB_PWDATA;
          WHEN others => null;
        END CASE;
      ELSIF (APB_PSEL = '1'and APB_PENABLE = '1' and APB_PWRITE = '0') THEN
      -- Read the register contents on PRDATA depending on the PADDR bus contents
        CASE APB_PADDR(4 downto 2) IS
          WHEN "000" =>
          -- PADDR = 0x80000300
            sig_PRDATA(0)  <= r.aes_en or r.aes_en_req;
            sig_PRDATA(2)  <= r.irq_en;
            sig_PRDATA(3)  <= r.irq;
          WHEN "001" =>
          -- PADDR = 0x80000304
            sig_PRDATA <= r.rd_start_addr;
          WHEN "010" =>
          -- PADDR = 0x80000308
            sig_PRDATA <= r.wr_start_addr;
          WHEN "011" =>
          -- PADDR = 0x8000030C
            sig_PRDATA(0)  <= r.ready;
            sig_PRDATA(1)  <= r.mem_wr;
          WHEN "100" =>
          -- PADDR = 0x80000310
            sig_PRDATA <= r.mem_addr;
          WHEN others => null;
        END CASE;
      END IF;

 ---------------------
 -- AHB Bus Conditions
 ---------------------

   CASE cstate IS
     WHEN idle =>
     -- Initialize all registers/bus contents
       sig_finish    <= '0';
       sig_mem_wr    <= '0';
       sig_rst_n     <= '0';
       sig_cs_n      <= '0';
       sig_cs_n1     <= '0';
       sig_wr_n      <= '0';
       sig_Go        <= '0';
       sig_key_in    <= (others => '0');
       sig_data_in   <= (others => '0');
       sig_addr_in   <= (others => '0');
       sig_addr_out  <= (others => '0');
```

```
        sig_HADDR     <= (others => '0');
        sig_HTRANS    <= (others => '0');
        sig_HWRITE    <= '0';
        sig_HWDATA    <= (others => '0');
        sig_dataRdy   <= '1';
        nstate        <= bus_req;

      WHEN bus_req =>
      -- Request bus for transaction
        IF (r.dma_xfer_req = '1') THEN
          sig_HBUSREQ   <= '1';
          sig_dataRdy   <= '0';
        ELSE
          sig_HBUSREQ   <= '0';
        END IF;
         IF (sig_dataRdy and r.aes_en) = '1' THEN
        -- Backup register signals...
          tmp.n             <= "111";  -- Initially number of WORDS (32-bit data) is
set to 7 (for counting eight 32-bit data 7...6...5...4)
          tmp.dma_xfer_req  <= '1';
          tmp.mem_addr      <= X"40000000";
        ELSIF (r.aes_en = '0') THEN
        END IF;
        -- ...and check for bus ownership
        tmp.bus_grant <= AHB_HGRANT;
        IF (tmp.bus_grant and r.dma_xfer_req) = '1' THEN
        -- Bus granted upon request
          tmp.bus_active  <= '1';
          nstate          <= bus_grant;
        ELSIF (tmp.bus_grant = '1' and r.bus_grant = '0' and r.dma_xfer_req = '0')
THEN
        -- Bus granted without request
          tmp.bus_active  <= '0';
          sig_HTRANS      <= HTRANS_IDLE;  -- Do nothing!
          sig_HBUSREQ     <= '0';
        END IF;

        WHEN bus_grant =>
        -- Skip first bus_own after granted
          IF (r.bus_active = '1' and AHB_HREADY = '1' and sig_dataRdy = '0') THEN
          -- Own bus at next clock
            nstate <= bus_own;
          END IF;

        WHEN bus_own =>
        -- Get ready for transaction
         IF (r.bus_active = '1' and AHB_HREADY = '1' and sig_dataRdy = '0') THEN
           tmp.bus_own   <= '1';
           sig_HTRANS    <= HTRANS_SEQ;      -- Subsequent bus transfers are ALWAYS
sequential
           nstate        <= load_key;
         END IF;

        WHEN load_key =>
        -- Load 128-bit KEY for encryption
          sig_key_in    <= tmp.key_in;
          sig_addr_in   <= tmp.aes_addr - 1;
          tmp.aes_addr  <= r.aes_addr + 1;
          CASE tmp.n(2 downto 0) IS
            WHEN "011" =>
            -- The 128-bit KEY comprising of four WORDS has been written
              nstate   <= load_text;
            WHEN others =>
              IF (sig_mem_wr = '0') THEN
              -- Write the KEY onto the internal DW_RAM
                tmp.key_in <= AHB_HRDATA;
                tmp.wr_n    <= '0';
              END IF;
            tmp.mem_addr  <= r.mem_addr + 4;  -- Update next Read address
```

```
          tmp.n         <= r.n - 1;         -- One WORD has already been written
          nstate        <= load_key;        -- Continue until 4 WORDS (128-bits)
have been written
          END CASE; -- tmp.n

      WHEN load_text =>
      -- Load 128-bit PLAIN TEXT for encryption
        sig_data_in   <= tmp.data_in;
        sig_addr_in   <= tmp.aes_addr - 1;
        tmp.aes_addr  <= r.aes_addr + 1;
        CASE tmp.n(2 downto 0) IS
          WHEN "000" =>
          -- The 128-bit Plain Text comprising of four WORDS have been read
            sig_dataRdy <= '1';  -- All data is present for encryption
            IF (r.aes_en = '1') THEN
            -- End transaction and start outputting data onto PRDATA bus
              sig_HTRANS  <= HTRANS_NONSEQ;
              sig_HWRITE  <= '0';
            ELSE
            -- End transaction and switch to idle state
              sig_HTRANS        <= HTRANS_IDLE;
              tmp.dma_xfer_req  <= '0';
            END IF;
            sig_Go  <= '1';
            nstate  <= xfer_end;
          WHEN others =>
            IF (sig_mem_wr = '0') THEN
            -- Write the PLAIN TEXT onto the internal DW_RAM
              tmp.data_in  <= AHB_HRDATA;
              tmp.wr_n    <= '0';
            END IF;
          tmp.mem_addr  <= r.mem_addr + 4;  -- Update next Read address
          tmp.n         <= r.n - 1;         -- One WORD has already been written
          nstate        <= load_text;       -- Continue until all 4 WORDS (128-bits)
have been written
          END CASE; -- tmp.n

      WHEN xfer_end =>
        IF (sig_finish = '1') THEN
          tmp.ready         <= '1';
          tmp.aes_en        <= '0';
          tmp.aes_en_req    <= '0';
          tmp.irq           <= r.irq_en;
          tmp.dma_xfer_req  <= '0';
          sig_mem_wr        <= '1';
          nstate            <= idle;
        ELSE
          sig_mem_wr        <= '0';
        END IF;

      WHEN others =>
        nstate <= idle;
      END CASE; -- cstate
    END IF;
  END IF;
 END process; -- the_process


  -- Encryption output on the AMBA bus
  irq   <= r.irq;
  APB_PRDATA  <= sig_PRDATA;
  AHB_HADDR   <= r.mem_addr;
  AHB_HTRANS  <= sig_HTRANS;
  AHB_HBUSREQ <= sig_HBUSREQ;
  AHB_HWDATA  <= sig_HWDATA;
  AHB_HLOCK   <= '0';
  AHB_HWRITE  <= sig_HWRITE;
  AHB_HSIZE   <= HSIZE_WORD;
  AHB_HBURST  <= HBURST_INCR8;
```

```
  AHB_HPROT   <= (others => '0');

  -- Synchronize data with CLK, RST signals
  sync: PROCESS(CLK, RST)
  BEGIN
  IF rst='0' THEN
    r.rd_start_addr <= (others => '0');
    r.n             <= (others => '0');
    r.wr_start_addr <= (others => '0');
    r.mem_addr      <= (others => '0');
    r.aes_en        <= '0';
    r.aes_en_req    <= '0';
    r.dma_xfer_req  <= '0';
    r.ready         <= '1';
    r.mem_wr        <= '0';
    r.irq_en        <= '0';
    r.irq           <= '0';
    r.bus_own       <= '0';
    r.bus_grant     <= '0';
    r.bus_active    <= '0';
    r.wr_n          <= '0';
    r.aes_addr      <= "1010";
    r.data_in       <= (others => '0');
    r.key_in        <= (others => '0');
  ELSIF RISING_EDGE(CLK) THEN
    r <=  tmp;
  END IF;
  END PROCESS; -- sync

END structural;

---
library ieee;
use     ieee.std_logic_1164.all;
use    IEEE.std_logic_arith.all;
package aes_enc_ctrl_struct_pkg is
  component aes_enc_ctrl_struct
  port (
      RST         :IN STD_LOGIC;
      CLK         :IN STD_LOGIC;

      -- AHB Bus Signals
      AHB_HGRANT  :IN STD_ULOGIC;
      AHB_HREADY  :IN STD_ULOGIC;
      AHB_HRESP   :IN STD_LOGIC_VECTOR(1 downto 0);
      AHB_HRDATA  :IN STD_LOGIC_VECTOR(31 downto 0);
      AHB_HBUSREQ :OUT STD_ULOGIC;
      AHB_HLOCK   :OUT STD_ULOGIC;
      AHB_HTRANS  :OUT STD_LOGIC_VECTOR(1 downto 0);
      AHB_HADDR   :OUT STD_LOGIC_VECTOR(31 downto 0);
      AHB_HWRITE  :OUT STD_LOGIC;
      AHB_HSIZE   :OUT STD_LOGIC_VECTOR(2 downto 0);
      AHB_HBURST  :OUT STD_LOGIC_VECTOR(2 downto 0);
      AHB_HPROT   :OUT STD_LOGIC_VECTOR(3 downto 0);
      AHB_HWDATA  :OUT STD_LOGIC_VECTOR(31 downto 0);

      -- APB Bus Signals
      APB_PSEL    :IN STD_ULOGIC;
      APB_PENABLE :IN STD_ULOGIC;
      APB_PADDR   :IN STD_LOGIC_VECTOR(31 downto 0);
      APB_PWRITE  :IN STD_ULOGIC;
      APB_PWDATA  :IN STD_LOGIC_VECTOR(31 downto 0);
      APB_PRDATA  :OUT STD_LOGIC_VECTOR(31 downto 0);

      -- Single Pin Interrupt Bus Signal
      irq         :OUT STD_LOGIC
     );
  end component;
end aes_enc_ctrl_struct_pkg;
```

Appendix B

COMPONENT XML FILE

AES.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<ip:component xmlns:ip="http://www.mentor.com/platform_ex/Namespace/IP"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.mentor.com/platform_ex/Namespace/IP
http://www.mentor.com/platform_ex/XMLSchema/3.5/component.xsd">
    <ip:vendor>UT</ip:vendor>
    <ip:library>VOLIPository</ip:library>
    <ip:name>aes</ip:name>
    <ip:version>1.0</ip:version>

    <ip:busInterfaces>
        <ip:busInterface ip:id="busInterface_0">
            <ip:name>AHB_mst</ip:name>
            <ip:busType ip:library="AMBA" ip:name="ambaAHB" ip:vendor="Mentor"/>

            <ip:master>
                <ip:addressSpaceRef ip:addressSpaceRef="ExternalMemory"/>
                <ip:presentation/>
            </ip:master>

        <ip:signalMap>
                <ip:signalName ip:busSignal="HRESETN">hresetn</ip:signalName>
                <ip:signalName ip:busSignal="HCLK">hclk</ip:signalName>
                <ip:signalName ip:busSignal="HGRANTx">hgrant</ip:signalName>
                <ip:signalName ip:busSignal="HREADYin">hready</ip:signalName>
                <ip:signalName ip:busSignal="HRESP">hresp</ip:signalName>
                <ip:signalName ip:busSignal="HRDATA">hrdata</ip:signalName>
                <ip:signalName ip:busSignal="HBUSREQx">hbusreq</ip:signalName>
                <ip:signalName ip:busSignal="HLOCKx">hlock</ip:signalName>
                <ip:signalName ip:busSignal="HTRANS">htrans</ip:signalName>
                <ip:signalName ip:busSignal="HADDR">haddr</ip:signalName>
                <ip:signalName ip:busSignal="HWRITE">hwrite</ip:signalName>
                <ip:signalName ip:busSignal="HSIZE">hsize</ip:signalName>
                <ip:signalName ip:busSignal="HBURST">hburst</ip:signalName>
                <ip:signalName ip:busSignal="HPROT">hprot</ip:signalName>
                <ip:signalName ip:busSignal="HWDATA">hwdata</ip:signalName>
            </ip:signalMap>
        </ip:busInterface>
        <ip:busInterface ip:id="busInterface_1">
            <ip:name>APB_slv</ip:name>
            <ip:busType ip:library="AMBA" ip:name="ambaAPB" ip:vendor="Mentor"/>
            <ip:slave>
                <ip:memoryMap>
                    <ip:addressBlock>
                        <ip:baseAddress ip:configGroups="requiredConfig"
ip:id="baseAddress_0">0x00000000</ip:baseAddress>
                        <ip:bitOffset>0</ip:bitOffset>
                        <ip:range>20</ip:range>
<!--                    <ip:width>32</ip:width>-->
                    </ip:addressBlock>
                </ip:memoryMap>
            </ip:slave>
        <ip:connection>required</ip:connection>
            <ip:signalMap>
                <ip:signalName ip:busSignal="PSELx">pselx</ip:signalName>
                <ip:signalName ip:busSignal="PENABLE">penable</ip:signalName>
                <ip:signalName ip:busSignal="PADDR">paddr</ip:signalName>
                <ip:signalName ip:busSignal="PWRITE">pwrite</ip:signalName>
```

```
                <ip:signalName ip:busSignal="PWDATA">pwdata</ip:signalName>
                <ip:signalName ip:busSignal="PRDATA">prdata</ip:signalName>
            </ip:signalMap>
        </ip:busInterface>
        <ip:busInterface ip:id="busInterface_2">
            <ip:name>IRQ_slv</ip:name>
            <ip:busType ip:library="Utility" ip:name="singlePinInterrupt"
ip:vendor="Mentor"/>
            <ip:slave>
                <ip:memoryMap/>
            </ip:slave>
        <ip:connection>required</ip:connection>
            <ip:signalMap>
                <ip:signalName ip:busSignal="interruptAH">irq</ip:signalName>
            </ip:signalMap>
        </ip:busInterface>
    </ip:busInterfaces>
    <ip:addressSpaces>
        <ip:addressSpace>
            <ip:name>ExternalMemory</ip:name>
            <ip:range>4G</ip:range>
        </ip:addressSpace>
    </ip:addressSpaces>

    <ip:registerBanks>
        <ip:registerBank>
            <ip:name>registers</ip:name>
            <ip:register>
                <ip:name>controlReg</ip:name>
                <ip:addressOffset>0x0</ip:addressOffset>
                <ip:size>32</ip:size>
                <ip:access>read-write</ip:access>
                <ip:resetValue>-1</ip:resetValue>
                <ip:field>
                    <ip:name>coreEnable</ip:name>
                    <ip:bitOffset>0</ip:bitOffset>
                    <ip:bitWidth>1</ip:bitWidth>
                    <ip:access>read-write</ip:access>
                    <ip:description>set if AES core selected</ip:description>
                </ip:field>
                <ip:field>
                    <ip:name>Reserved1</ip:name>
                    <ip:bitOffset>1</ip:bitOffset>
                    <ip:bitWidth>1</ip:bitWidth>
                    <ip:access>read-write</ip:access>
                    <ip:description>Reserved</ip:description>
                </ip:field>
                <ip:field>
                    <ip:name>irqEnable</ip:name>
                    <ip:bitOffset>2</ip:bitOffset>
                    <ip:bitWidth>1</ip:bitWidth>
                    <ip:access>read-write</ip:access>
                    <ip:description>set if IRQ enabled</ip:description>
                </ip:field>
                <ip:field>
                    <ip:name>irqRequest</ip:name>
                    <ip:bitOffset>3</ip:bitOffset>
                    <ip:bitWidth>1</ip:bitWidth>
                    <ip:access>read-write</ip:access>
                    <ip:description>set if IRQ requested</ip:description>
                </ip:field>
                <ip:field>
                    <ip:name>Reserved2</ip:name>
                    <ip:bitOffset>4</ip:bitOffset>
                    <ip:bitWidth>28</ip:bitWidth>
                    <ip:access>read-write</ip:access>
                <ip:description>Reserved</ip:description>
                </ip:field>
```

```
                <ip:description>Bits 0, 2, 3 are set/reset depending on selection of
the AES core, interrupt enabling and interrupting requesting,
respectively.</ip:description>
            </ip:register>
            <ip:register>
                <ip:name>dataLoadReg</ip:name>
                <ip:addressOffset>0x4</ip:addressOffset>
                <ip:size>32</ip:size>
                <ip:access>read-write</ip:access>
                <ip:resetValue>-1</ip:resetValue>
                <ip:field>
                    <ip:name>rdStartAddr</ip:name>
                    <ip:bitOffset>0</ip:bitOffset>
                    <ip:bitWidth>32</ip:bitWidth>
                    <ip:access>read-write</ip:access>
                    <ip:description>Input KEY and PLAIN from this
address</ip:description>
                    <ip:values>
                        <ip:value>0x40000000</ip:value>
                        <ip:description>Leon2 Internal RAM location</ip:description>
                        <ip:name>rdAddr</ip:name>
                    </ip:values>
                </ip:field>
                <ip:description>This register holds the address from which the KEY
and PLAIN TEXT are used for encryption</ip:description>
            </ip:register>
            <ip:register>
                <ip:name>cipherOutReg</ip:name>
                <ip:addressOffset>0x8</ip:addressOffset>
                <ip:size>32</ip:size>
                <ip:access>read-write</ip:access>
                <ip:resetValue>-1</ip:resetValue>
                <ip:field>
                    <ip:name>wrStartAddr</ip:name>
                    <ip:bitOffset>0</ip:bitOffset>
                    <ip:bitWidth>32</ip:bitWidth>
                    <ip:access>read-write</ip:access>
                    <ip:description>Cipher output stored to this
address</ip:description>
                    <ip:values>
                        <ip:value>0x40001000</ip:value>
                        <ip:description>Leon2 Internal RAM location</ip:description>
                        <ip:name>wrAddr</ip:name>
                    </ip:values>
                </ip:field>
                <ip:description>This register holds the address where the ciphered
output from the AES core is stored</ip:description>
            </ip:register>
            <ip:register>
                <ip:name>statusReg</ip:name>
                <ip:addressOffset>0xc</ip:addressOffset>
                <ip:size>32</ip:size>
                <ip:access>read-only</ip:access>
                <ip:resetValue>-1</ip:resetValue>
                <ip:field>
                    <ip:name>ready</ip:name>
                    <ip:bitOffset>0</ip:bitOffset>
                    <ip:bitWidth>1</ip:bitWidth>
                    <ip:access>read-only</ip:access>
                    <ip:description>Ready/Busy status indicator</ip:description>
                </ip:field>
                <ip:field>
                    <ip:name>memwr</ip:name>
                    <ip:bitOffset>1</ip:bitOffset>
                    <ip:bitWidth>1</ip:bitWidth>
                    <ip:access>read-only</ip:access>
                    <ip:description>Writing/Reading status indicator</ip:description>
                </ip:field>
                <ip:field>
```

```
                    <ip:name>Reserved3</ip:name>
                    <ip:bitOffset>2</ip:bitOffset>
                    <ip:bitWidth>30</ip:bitWidth>
                    <ip:access>read-write</ip:access>
                    <ip:description>Reserved</ip:description>
                </ip:field>
                <ip:description>Bits 0, 1 are set/reset if the AES core is ready/busy
or if data is being written to/read from the core, respectively</ip:description>
            </ip:register>
            <ip:register>
                <ip:name>memReg</ip:name>
                <ip:addressOffset>0x10</ip:addressOffset>
                <ip:size>32</ip:size>
                <ip:access>read-only</ip:access>
                <ip:resetValue>-1</ip:resetValue>
                <ip:field>
                    <ip:name>currentHADDR</ip:name>
                    <ip:bitOffset>0</ip:bitOffset>
                    <ip:bitWidth>32</ip:bitWidth>
                    <ip:access>read-write</ip:access>
                    <ip:description>Current Address on AHB HADDR</ip:description>
                </ip:field>
                <ip:description>This register holds the actual DMA
address</ip:description>
            </ip:register>
        </ip:registerBank>
    </ip:registerBanks>

    <ip:presentation>

        <ip:displayLabel>AES Encryption Core</ip:displayLabel>
        <ip:icon>images/oc_logo_outlined.gif</ip:icon>
        <ip:document ip:menuDescription="AES Rijndael
Core">http://www.opencores.org</ip:document>
    </ip:presentation>

    <ip:hwModel>
        <ip:name>aes</ip:name>
        <ip:verificationEnvironment ip:id="modelsimVHDL">
            <ip:envIdentifier>ModelsimVhdl</ip:envIdentifier>
            <ip:language>vhdl</ip:language>
            <ip:defaultFileBuilder>
                <ip:fileType>vhdlSource</ip:fileType>
            </ip:defaultFileBuilder>

        <ip:fileSetRef>fs-externalVhdlSource</ip:fileSetRef>
            <ip:parameter ip:name="entityName">aes</ip:parameter>
        </ip:verificationEnvironment>
   </ip:hwModel>

    <ip:fileSets>
        <ip:fileSet ip:fileSetId="fs-externalVhdlSource">
            <ip:file>
                <ip:name>hdlsrc/aes_enc_ctrl_struct.vhd</ip:name>
                <ip:fileType>vhdlSource</ip:fileType>
            </ip:file>
               <ip:file>
                <ip:name>hdlsrc/aes.vhd</ip:name>
                <ip:fileType>vhdlSource</ip:fileType>
            </ip:file>

        </ip:fileSet>
    </ip:fileSets>

    <ip:persistentInstanceData ip:id="persistentData" ip:resolve="user"/>
</ip:component>
```

Appendix C

THE sparc-elf-gcc BUILD FOR SOLARIS

The information provided here is intended mainly for first-time enthusiasts. For those who believe strongly in the 'design reuse theory', a copy of the Solaris build for sparc-elf-gcc is available at http://vlsi1.engr.utk.edu/~wala/sparc-elf-gcc.html.

Many thanks to Jiri Gaisler, for providing the 'how to' on building this cross-compiler and to our System Administrator, Matt Disney, for finding workarounds that suited our system. One of the popular tutorials on this subject is written by William Gatliff and can be obtained at http://www.microcross.com/gnu-arm7t-microcross.pdf.

1. Obtain the Linux Bare-C Cross-compiler (BCC) system for Leon2 from http://www.gaisler.com under Downloads > CCS.
2. Get the gcc-3.2.3 and binutils-2.14 sources from http://www.gnu.org. You need not build newlib as the one in Linux BCC can be used on Solaris.
3. Start by installing the Linux BCC on your Solaris host under the /opt directory. BCC is provided as a bzipped tar-file. To unpack it in the /opt directory:

   ```
   cd /opt
   bunzip2 bcc-linux-<version>.tar.bz2
   tar -xvf bcc-linux-<version>.tar
   ```

   After installation, add /opt/sparc-elf/bin to the PATH variable. ***Note:*** Do not add any other path – /opt/sparc/elf/sparc-elf/bin.
4. Build and install binutils-2.14 and gcc-3.2.3 as explained in Bill Gatliff's tutorial. Configure the build with target=sparc-elf --prefix=/opt/sparc-elf --enable-languages=c,c++

5. Install the libio and mkprom utilities as explained below:

```
cd /opt/sparc-elf/src/libio
make install
cd ../mkprom
make install
```

6. Test the compiler by compiling a test application.

The following are Matt Disney's install notes:

Observe that in our case, Linux BCC was installed under /sw2.

```
I had to create a symbolic link in the binutils source directory to
/dev/null.
    For example:
    cd binutils-2.14
    mkdir dev
    ln -s /dev/null dev/null

Here is my configure command line for binutils (run from the build-binutils
directory):
/sw2/sparc-elf/binutils-2.14/configure  --target=sparc-elf
- --prefix=/sw/sparc-elf/ --disable-nls

After running make in build-binutils, the build would break with errors
from make about bfd/po. I had to cd into bfd/po and then copy Makefile.in
to Makefile (which was blank). Then go back up to the build-binutils
directory and run make again to finish the build.

Here is my configure command line for gcc (run from the build-gcc
directory):
/sw2/sparc-elf/gcc-3.2.3/configure
- --with-gcc-version-trigger=/sw2/sparc-elf/gcc-3.2.3/gcc/version.c
- --host=sparc-sun-solaris2.8 --target=sparc-elf --prefix=/sw/sparc-elf
- --with-newlib --without-headers --with-gnu-as --with-gnu-ld --disable-
shared
- --enable-languages=c --disable-nls --norecursion
```

Appendix D

PARTIAL 'BUILD' LOG

```
Running generator chain: vendor=Mentor library=topLevel name=Build
Generating the SW Builder Scripts
received 5 options
processed: CONFIG_IU_FASTDECODE = yes
processed: CONFIG_IU_FASTJUMP = yes
processed: CONFIG_PERI_LCONF = yes
processed: CONFIG_IU_LDELAY = 1
processed: CONFIG_IU_NWINDOWS = 8
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/verificationEnv/Modelsim/testbe
nch/pxdefault_tb

untar:
    [untar] Expanding:
/tnfs/home/wala/PlatformExpress2.1h/pxLibraries/Leon2/leon2/leon2-1.0.3.gtar.gz into
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/verificationEnv/Modelsim/design
Data/libraryData/Mentor_Leon2

modelsim:
     [echo] Leon2 common build: modelsim

untar:
    [untar] Expanding:
/tnfs/home/wala/PlatformExpress2.1h/pxLibraries/Leon2/leon2/leon2-1.0.3.gtar.gz into
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/verificationEnv/Modelsim/design
Data/libraryData/Mentor_Leon2

compile:
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/verificationEnv/Modelsim/hdl
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
```

```
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0/hdlsrc/leon2
     [copy] Copying 158 files to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2/componentLi
brary/component/mcore_ahb/1.0
     [copy] Copying 236 files to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/exportedFiles/Leon2
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/verificationEnv/Modelsim/exec

compile:
     [exec] ** Warning: (vlib-34) Library already exists at
"work/wala_userlibrary_aes_blckbox_bbcLib_Mentor_bbcLib_aes_1".
     [exec] ** Warning: (vlib-34) Library already exists at
"work/wala_userlibrary_aes_blckbox_bbcLib".
     [exec] ** Warning: (vlib-34) Library already exists at
"work/wala_userlibrary_aes_blckbox_bbcLib_Mentor_Leon2_mcore_ahb_1".
     [exec] ** Warning: (vlib-34) Library already exists at
"../designData/libraryData/Mentor_Leon2/leon2.lib".
     [exec] ** Warning: (vlib-34) Library already exists at "work/leon2_apbmst_obb".
     [exec] ** Warning: (vlib-34) Library already exists at "work/pxdefault_tb".
     [exec] Model Technology ModelSim SE vcom 5.8d Compiler 2004.06 Jun 12 2004
     [exec] -- Loading package standard
     [exec] -- Loading package std_logic_1164
     [exec] -- Compiling package amba
     [exec] -- Compiling package target
     [exec] Model Technology ModelSim SE vcom 5.8d Compiler 2004.06 Jun 12 2004
     [exec] -- Loading package standard
     [exec] -- Loading package std_logic_1164
     [exec] -- Loading package target
     [exec] -- Compiling package device
     [exec] Model Technology ModelSim SE vcom 5.8d Compiler 2004.06 Jun 12 2004
     [exec] -- Loading package standard
     [exec] -- Loading package std_logic_1164
     [exec] -- Compiling entity pxdefault_tb
     [exec] -- Compiling architecture platformexpress of pxdefault_tb
     [exec] -- Loading entity top
     [exec] -- Compiling configuration pxconfig_pxdefault_tb
     [exec] -- Loading entity pxdefault_tb
     [exec] -- Loading architecture platformexpress of pxdefault_tb
     [exec] -- Loading configuration pxconfig_top
#!/bin/sh -ev
########################################################################
##                    Px Generated File                               ##
##              Platform Express, Version 2.1h (build 835)            ##
##              SoC Verification Division                             ##
##              Mentor Graphics Corporation                           ##
##                                                                    ##
## Generated on: March 16, 2005 11:13:24 PM EST                       ##
## Generated by: wala                                                 ##
## Software compile script                                            ##
########################################################################
```

```
if [ -f ../../pxenv.sh ] ; then . ../../pxenv.sh; fi
sparc-elf-gcc -c -g  -D__leon__ -
I../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software -
I../../../../exportedFiles/Leon2/common/include
../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software/loc
ore1.S -o ./locore1.o
sparc-elf-gcc -c -g -D__leon__ -
I../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software -
I../../../../exportedFiles/Leon2/common/include
../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software/leo
n_test.c -o ./leon_test.o
sparc-elf-gcc -c -g -D__leon__ -
I../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software -
I../../../../exportedFiles/Leon2/common/include
../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software/mis
c.c -o ./misc.o
../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software/mis
c.c:38:5: warning: multi-line string literals are deprecated
sparc-elf-gcc -c -g -D__leon__ -
I../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software -
I../../../../exportedFiles/Leon2/common/include
../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software/irq
ctrl.c -o ./irqctrl.o
../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software/irq
ctrl.c:66:21: warning: multi-line string literals are deprecated
sparc-elf-gcc -c -g -D__leon__ -
I../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software -
I../../../../exportedFiles/Leon2/common/include
../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software/uar
t.c -o ./uart.o
sparc-elf-gcc -c -g -D__leon__ -
I../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software -
I../../../../exportedFiles/Leon2/common/include
../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software/pxC
oreLib.c -o ./mcore_ahb_pxCoreLib.o
sparc-elf-gcc -c -g -D__leon__ -
I../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software -
I../../../../exportedFiles/Leon2/common/include
coreDiagnostics/_mcore_ahb_1/printToPort.c -o ./printToPort.o
sparc-elf-gcc -c -g -D__leon__ -
I../../../../exportedFiles/Leon2/componentLibrary/component/mcore_ahb/1.0/software -
I../../../../exportedFiles/Leon2/common/include
coreDiagnostics/_mcore_ahb_1/pxDiagnostics.c -o ./pxDiagnostics.o

sparc-elf-gcc ./mcore_ahb_pxCoreLib.o \
./printToPort.o \
./misc.o \
./uart.o \
./locore1.o \
./pxDiagnostics.o \
./leon_test.o \
./irqctrl.o \
 -g  -nostdlib -static -N  -e _hardreset   -T ./linker.ld  -o boot.elf
sparc-elf-objcopy --remove-section=.comment boot.elf

sparc-elf-objdump -s boot.elf >
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/verificationEnv/Modelsim/exec/r
am.dat
     [copy] Copying 1 file to
/tnfs/home/wala/pxPrj/blackBoxTest/aes_blckbox_bbcLib/verificationEnv/Modelsim/exec
```

VITA

Mardavsinh Wala was born in Ahmedabad, India. He attended Nirma Institute Technology, Ahmedabad from 1997 to 2001 where he graduated with a Bachelor of Engineering degree in Instrumentation and Control Engineering. Mardav came to the United States of America in the Spring of 2002 for his Masters' study in the Department of Electrical and Computer Engineering at University of Tennessee. Since then, he has worked as a Graduate Technology Assistant for the Office of the Vice Chancellor of Student Affairs and the Office of the Dean of Students. He began working for his thesis research under Dr. Don Bouldin in Spring 2004. He will be awarded the Master of Science degree in May 2005.