FRONTEND DESIGN FLOW OF DIGITAL SUBSYSTEM

A Thesis Submitted in Partial Fulfillment of the Requirements for the Award of the Degree of

Master of Technology

In VLSI Design

Submitted By

PIYUSH KUMAR
601762013

Under Supervision of

Dr. Mohit Aggarwal
Assistant Professor

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT
THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY
(A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB
JULY, 2019

Date: June 28th, 2019

CERTIFICATE

This is to certify that Piyush Kumar (601762013), a student of M.Tech (VLSI), Thapar Institute of Engineering & Technology, Patiala, has successfully completed one year (June 2018 – June 2019) internship program in STMicroelectronics Pvt. Ltd., Greater Noida. He has done his work on the "**Frontend Design of the Hardware Security Module (HSM) subsystem**" from June 2018 to June 2019.

Charul Jain

Charul Jain

Staff engineer

STMicroelectronics Pvt. Ltd.

Greater Noida, India

DECLARATION

I, **Piyush Kumar** hereby declare that the work presented in this thesis entitled "**Frontend Design Flow Digital Subsystem**" in partial fulfillment of the requirement for the award of degree of **Master of Technology (VLSI Design)** submitted at **ECED** , Thapar Institute of Engineering & Technology (Deemed to be University), Patiala is an authentic record of work carried out under supervision of **Dr. Mohit Aggarwal (Assistant Professor, ECED, Thapar institute of Engineering and Technology)** from **2017 to 2019**. The matter presented in this has not been submitted either in part or full to any other university or institute for the award of any other degree.

Date: 15/7/19

Piyush Kumar
601762013

It is certified that the above statement made by student is correct to the best of my knowledge and belief.

Date: 15/7/19

Dr. Mohit Aggarwal
Assistant Professor
Department of Electronics and Communication Engineering
Thapar Institute of Engineering & Technology
Patiala, Punjab

iii

ACKNOWLEDGMENT

The work for this thesis was carried out at STMicroelectronics, Greater Noida, India, during the year 2018-2019.

Firstly, I would like to thank my guide **Dr. Mohit Aggarwal** and my mentor **Mrs. Charul Jain** for providing excellent guidance and encouragement throughout the journey of this work. Without their guidance, this work would never have been a successful one. I also take this opportunity to express a deep sense of gratitude towards my Industrial colleagues Mr. Nirav Prashantkumar Trivedi from Digital IP team at STMicroelectronics for his support and encouragement for conquering every hurdle that I have encountered throughout the process. I also like to thank Principle Engineers Mr. G.N Chaudhary and Mr. Asif Rashid Zargar from Digital IP team, ST Microelectronics for the technical discussion whenever I was in need of any. My regards to all my friends at TIET, Patiala who made this journey a wonderful one. Last but not the least; I would like to thank my Parents for supporting me spiritually and emotionally.

Piyush Kumar

ABSTRACT

Complexities in the chip designs are continuously increasing with the increase in the size of the designs. The integration of IP blocks into a design and managing the design is becoming more and more challenging. Thus the semiconductor industry has began to embrace new design and reuse methodologies to keep in pace with increased complexities of circuits. The predesigned and pre verified blocks known as intellectual property are obtained from internal sources or third parties and are integrated together into a single chip. This traditional process of integration is time consuming, error prone and requires a lot of efforts from the designer. A large percentage of design errors come from connectivity issues. Component reuse, complex buses and input/output connections add challenges to the SoC integration process. Intellectual Property reuse improves system on chip design productivity, helps to meet design quality and time to market goals.

In this thesis a design methodology has been used which enables the IP reuse by using the standard bus architectures such as AMBA bus protocol and IP-XACT standards which allow the extensive reuse of designs and ease the process of design integration. The IPs are packaged into IP-XACT standard .XML files. The packaging extracts the metadata of designs and makes the design easy to use in different tools flow. The IPs are then integrated together to make a subsystem. A subsystem, Hardware Security Module has been implemented by using this methodology in the design flow. This use of these standards significantly reduced the design time..

Table of Contents

Table of Figures

Abbreviations Used

AMBA	Advanced Micro Controller Bus Architecture
ASIC	Application Specific Integrated Circuit
CDC	Clock Domain Crossing
DMA	Direct Memory Access
DSP	Digital Signal Processing
FPGA	Field Programmable Gate Array
HSM	Hardware Security Module
HDL	Hardware Description Language
IC	Integrated circuit
IP	Intellectual Property
MPU	Memory Protection Unit
RMM	Reuse Methodology Manual
SoC	System on Chip
USB	Universal Serial Communication
VHDL	Very Large Scale Hardware Description Language
VLSI	Very Large Scale Integration
XML	Extensible Markup Language

Chapter – 1

Introduction

Over the past few years, the ICs are becoming more complex and smaller in size. The problem of making ICs with billions of gates requires more powerful resources and time. Current design methodologies are not much efficient to meet the challenges of design quality, design productivity and in reducing time-to-market window. Therefore, designers are looking for a more reliable and more efficient methodologies. The frontend subsystem design flow involves many steps, that is from deciding the architecture based upon specification and designing the required blocks and then integrating them into a single working subsystem. The process also includes verifying the design and generating a gate level net-list by synthesizing the design. Predesigned and verified blocks also referred to as (IPs) intellectual property or virtual components can also be obtained from other vendors and integrated on a single chip. Huge gains in productivity can be approached by using this technique. Successful integration of IPs requires different views that provide necessary information for every IP block through the design flow for an integration system. The process of integration of different IP into a system level is challenging and complex task which demands skills along with some standards to be followed for proper functioning of the system. To tackle the design complexity problem, the key elements are IP reuse and to automate the process of designing and verification. By increasing the reusability and interoperability of IPs, the risk of developing IP modules with errors and their costs can be reduced effectively. The IPs are designed in accordance to some standards that ensures the flexible reuse of IPs. Standards such as IP-XACT and AMBA protocols are used to develop IPs and subsystems. The use of EDA tools and extensive automation allows incorporating IP cores for high level specification designs which helps in improving the time-to-market. This is achieved by reducing the multiple iterations on the manual error prone designing and verification processes. For a design which contains multiple clocks, it is important to check any setup and hold time violations at the clock domain crossing boundaries. Alongside design, the process of verification also starts parallel so as to reduce the time to market. Verification is done to check the proper integrity of subsystem. In first stage, it includes the sanity checks, where proper reset and boot up the system is checked. Also basic read/write operations are performed to check the correctness of integration. The main reason for verification of design is to check whether the design is working properly or not in accordance with the design specifications. Verification consumes 40% to 50% of the effort of design cycle and is on the critical path in the design flow. Verification must be well planned and organized. Otherwise it is impossible to say and prove that the verification is done. Features to be verified must be stated clearly and completeness of the verification needs to be tracked as well as implementation errors of the design. If any of the two, verification or design, is not ready, the design cannot be considered as ready for the tape-out.

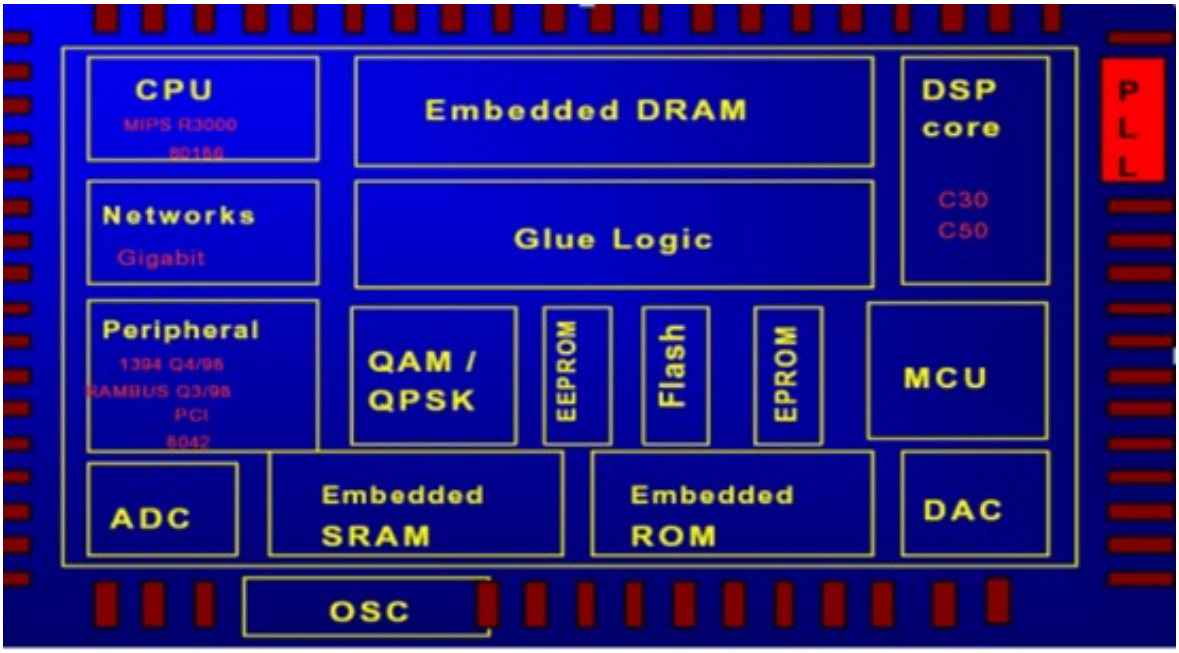

Figure 0-1 A subsystem containing many IP blocks.

Many IPs blocks are integrated into a single subsystem as shown in *Figure 1-1* which then works as a single device to perform specific functions. We will further discuss how these processes of integration and designing works and will develop a working subsystem.

1.1 Overview of Subsystem Design Flow

For designing a subsystem, it is necessary to understand the specifications and then proceed to the implementation of design. The specifications are discussed in multiple iterations to meet the various design constraints. Finally, the specifications are frozen and the implementation of design starts. The process of design contains various stages. Following *Figure 1-2* will illustrate a standard SoC/subsystem design flow which involves various stages form specification to tape-out. In *Figure 1-2,* the whole VLSI chip design process has been shown which includes various stages from specifications to tape out of the chip but in this document, we are only concerned with the frontend design process which includes defining the specifications of the design, making the architecture for the design, RTL coding, and verification of design.

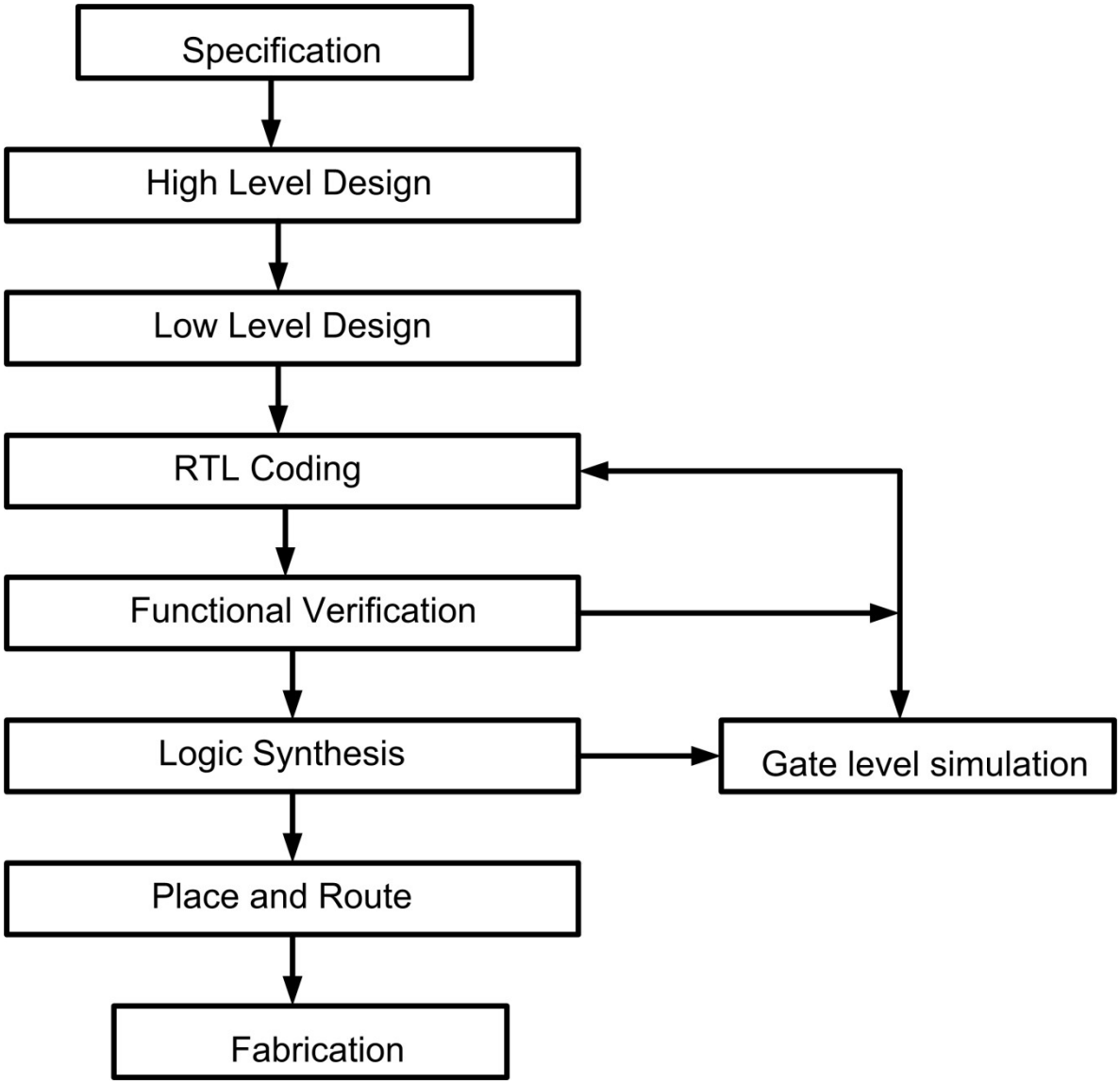

Figure 0-2 Design Flow Of a VLSI Chip.

As depicted in *Figure 1-2, the* Frontend design process contains the stages from specifications to the verification of the design.

- **Specifications** are the first stage of design flow where important parameters or features that are needed in the subsystem are taken into consideration and the design starts.
- In **High Level Design**, the architecture of the whole design is defined. In this stage, the details of different functional blocks which are to be used, interface and communication protocols between them are defined.

- In **Low level design** stage, which is also called micro-architecture design stage, details about each functional block and their implementation is defined. It generally includes modules, state machines, decoders, multiplexers, counters, registers etc.
- In **RTL Coding stage**, the micro-architectural design is implemented by using a Hardware Description Language like VHDL or Verilog. The caution is taken during stage is that while writing the code, only synthesizable verilog language constructs are used so that actual gate level mapping of the RTL model can be done and physical implementation of design is possible. The RTL model must be bug free and should pass the CDC(Clock Domain Crossing) and lint checks.
- **Functional Verification** is the process of checking or verifying the basic connectivity and functional behavior of the design. This is achieved by making a test-bench environment in which design is placed. The proper input stimuli is given as test input and corresponding to the input,,the behavior is checked and verified.

Rest of the stages in the flow which includes logic synthesis, placement and routing, gate level simulation, and fabrication and post silicon validation falls in the backend design process and are beyond the scope of this document.

1.2 Hardware security Module

With the increase in the communication between the devices, it has become a challenging task to prevent the information from being stolen. It is necessary to have a mechanism which provides security of the data and prevent information from being hacked. A Hardware Security Module is basically a hardware which performs cryptographic functions like cryptographic key generation, storing of digital keys, authentication and securing the information. The hardware is generally a plug-n-play device. HSM find their application in many security services. They provide high level of security in automobile electronics systems and prevent the un-authorized access to the vehicles. They provide a temper detect mechanism in which all the critical data is erased when the device is found to be tempered. They are also employed in card payment industry where the authentication of user is done for making the payments.

1.3 Motivation

This design and implementation of a subsystem is a complex task, which require a lot iteration of design cycles to achieve the desired results. This repetitive task is much error prone and requires a lot of effort to figure out errors in the design in early stages. So, there is a need for a design flow which will take care of error free design in less time. The use of different standards in the design processes allow to overcome design related problems. The benefits of making reusable and configurable blocks make it easy to use them in other designs without much effort. Increase in the reusability decreases the design efforts and time to market.

1.4 Objectives of the work

The Objectives of the work are:

- To implement the **Frontend Design flow** to achieve the maximum productivity in minimum time.
- To implement a subsystem (HSM - Hardware Security Module) by using the subsystem design flow.
- To Perform the CDC (Clock Domain Crossing) check on the Implemented Design.
- To check whether the device will power up properly or not.

1.5 Organization of the Thesis

The work in this thesis has been organized in five chapters. Chapter 1: It includes the introduction to the subsystem design and the complexities in the designs. It gives overview in the design process and various stages involved into it. It also have a small introduction about the Hardware Security Module and its applications. Chapter 2: It contains the literature survey done by various authors in the field of design flow and observations drawn from them. Chapter 3: It includes the introduction to various standards in design flow and the implementation architecture of Hardware Security Module. Chapter 4: This part consists of results obtained in different stages of design implementation of Hardware Security Module. Chapter 5: Conclusion made in this work and future scope has been discussed.

Chapter-2

Literature Survey

In this chapter, the survey of work done by various people in the field of ASIC design has been highlighted. In this chapter, we will discuss the various methodologies and challenges in the design flow. From the analysis of the literature, we shall conclude the proper methodology for the implementation of HSM (Hardware Security Module) sub system.

2.1 Introduction

In [1] **Gordon Moore** has talked about miniaturizing the size of electronics equipments and increase complex electronic functions in minimum limited space. The statement further known to us as famous Moore's law which stating that the number of transistors in an IC (Integrated Circuit) will double in about every two years. This causes rapid increasing complexity in designs [2] and the many problems related to design. The semiconductor industry has made many exciting improvements in achieving very high densities in VLSI (Very Large Scale Integration) circuits [3].Design engineers have to develop new techniques and methodologies to make progress with the increased complexities of the designs.

As mentioned in [4] that to meet the design challenges of SoC design, the traditional design flows are changing, for example instead of traditional waterfall model, spiral model is being used. Designers are not only using top down approach but a combination of both top down and bottom up approaches. [4] Clearly indicate that in spiral model, multiple aspects of designs are worked upon simultaneously. The concept of reusability of IP blocks has been explored in [5]. **Saleh** et al in [5] focuses on reuse and integration issues, the reusability can be carried at block, platform and chip level. He suggests the use of more generic IP blocks which are configurable for a wide range of applications.

Kwanghyun et al in [6] uses an innovative SoC integration methodology to increase the design productivity by employing IP reuse and IPXACT standards which has reduced the integration and SoC verification time by 30%. Out of many emerging ideas, the SPIRIT [8] and the design reuse [7] are most

prominent one. In platform based design which is actually an abstraction which includes lower level of abstractions. **A.S. Vincentelli** et al in [9] focuses on the use of platform based methodology where similar SoC chips which are slightly different from one another in one or more functions and components can use same platform model. The challenge these days is not implementation of reuse methodology but is how it should be implemented. Reuse Methodology manual [10] gives a sight into effective use of this methodology.

As mentioned in [11], **Ajoy Bose** says that "*New level of abstraction is Reusable IP*". **Ron Wilson** [11] is talking about selection of IPs used for the design based upon the specifications. The main focus is on IP centric flow in which IPs are assembled and the only RTL to be written is glue logic. From implementation point of view, the IP centric flow is beneficial for Backend Flow also. To tackle with rising complexity, the level of abstraction shall also increase. From [4] it is deduced that the process of designing a reusable IP is quite different as it requires more design efforts and minimum five times more time to design as compared to traditional process of IP Block design.

ARM AMBA is the standard [12] bus protocol for making connections which supports on-chip communication effectively. The reason for using AMBA is its AHB (Advanced High Performance Bus) high speed protocol[12] and a low bandwidth, low power consuming APB (Advanced Peripheral Bus). **Salita Sombatsiri** [13] has proposed the use of AMBA and has estimated area and time of execution of different architectures. **Rishabh Singh Kurmi** et al. [14] have demonstrated the working of AMBA – AHB bus and coding has been carried out using VHDL. AHB has been implemented using burst mode of transfer and has been verified.

W. Kruijtzer et al [16] have focused on the IP-XACT standards for SoC integration. IP-XACT standards have been used in flows to make the designs easily portable and reusable to increases the productivity. [16] It has been summarized that the IP-XACT is a powerful methodology for the integration of IPs. Increased reusability of IP modules helps in reducing the time to market window of the designs and the risks of IP development.[17] gives the insight in how to use the IP-XACT standards in the tool flows for the packaging and reusing the IPs.

In [18] **Clifford E. Cummings** has described the problem of meta-stability in multiple asynchronous clock domain designs. The problem of CDC is explained to a detailed level and has explained methods to tackle the problem of CDC in the design. In [19], Cummings has described a method to design, analyze and synthesize a FIFO between asynchronous clock domains using grey code pointers. **Don Mills** et al [20] has discussed different coding styles that can lead to the mismatch between gate level simulation and RTL. These design mismatches are very hard to detect if bad coding styles are used.

HU Zhaohui et al in [21] have presented an efficient and practical way of verification of SoC by introducing reusable IP test-bench structures and by using the UVM (Universal verification methodology). Applying this methodology, he has successfully decreased the complexities in the verification of a SoC and also the debugging difficulties. Magillem has been used [28] for designing a System On Chip. The methodology for design and the tool used has been tested.

H. Ju and **J. Kim** [29] has implemented the hardware security module on smart phone platform. Security mechanism against any physical tampering [30] has been implemented by **S. Zamanzadeh** et al on a FPGA. Use of hardware security module as a USB stick has been implemented [31] in a very cost effective way. **D. Hwang** et al [32] has made a design based on the security interface for the protection of memory on RISC processor. It has operated independently without using processor resources and causing any overhead.

Pawankumar B [33] has increased the level of abstraction by using the concept of IP reusability and by using ESL (Electric System Level) design methodology. Test efforts are decreased to a large extent by applying same test cases in multiple designs verification. Stand alone verification and system level verification has been performed.

Hardware and software co-verification [34] has been done based upon a co-verification platform. By implementing platform based co-verification **Xue-Qiu Dai** et al has not only achieved full functional coverage but has also decreased the time for design verification. 3 months of design time on average has been shortened.

2.2 Observations from the literature survey

From the literature survey, following observations can be deduced:

- To achieve the less time to market and ease of integration of IPs, the reusable IPs must be designed which are highly configurable. This can be achieved by using standard Bus architectures such as AMBA.
- It is clear that the process of design require many iterations of the same work which is very much time consuming and requires a lot of design efforts. Therefore for error free design, it is necessary to adopt a method which does not require much efforts and time when a little changes are made in design.
- Use of IP-XACT standard in design flow can significantly reduce the design time by providing customary organization for IP reuse, integration and packaging.

Chapter – 3

Design Flow implementation

In this section, the components, standards and processes required for the implementation of subsystem design flow has been described. Also various steps involved in the design flow are implemented to design a HSM (Hardware Security Module Subsystem) .

3.1 Digital IP

Digital IPs is the basic building blocks required to make an integrated circuit or chip. It is a reusable unit of logic or functionality or a layout design for example ARM bus protocols like AMBA, AHB, design cores like USB, Ethernet etc. Digital IP blocks in the VLSI industry are the most widely worked on and used form of reusable IPs. The design methodologies and the tools for making digital IP are already being applied for implementing the concept of reusability. However, to implement the maximum reusability possible, it is required to do tremendous changes in the design methodologies of IP design. Moreover, there are few more technological considerations which are needed to be pointed out in order to take care of all the applications where the IP block may be used.

The most important stages which are must be taken into consideration of an IP design are :

1. Specifications and documentation of the IP to make the design understanding easy for its users.
2. Implementation of the design by using standard coding practices.
3. Verifying the design functionality in order to check any errors.

The first stage includes making of easily readable and easy to understand documentation for the IP block such as its block guide, creation guide and integration guide. These documents will further lead other teams to understand the IP block and to use it accordingly. The second step includes coding of the design based upon standard coding practices such as by using only synthesizable constructs. However, it will be difficult to believe that the second part of design implementation is only like a tip of iceberg. It is surprisingly a very small part of the design process. Depending upon the behavior, type, size and the functionality of the IP, the third stage that is verification of IP may take up to 50% of the total design time. Since the IP is to be used many times differently in a variety of applications, design errors inside the

block must not be tolerated up-to a certain level. The goal of verification is to ensure the correctness and proper functioning of the IP block.

IPs are generally licensed as either **Soft IP, Hard IP** or **Firm IP:**

• **Soft IP** blocks are generally specified as synthesizable RTL models. They are developed in HDLs (Hardware Description Languages) like Verilog or VHDL. They are more suitable for digital cores because Hardware Description Languages are process-independent and IP can be provided as a generic gate-level net-list after the synthesis of the design. They are more configurable and flexible than Hard IPs. The implementation of Soft IPs in different applications and processes produces different results such as many different applications will have variations in performance. Sometimes, the soft IP in the form of a RTL code is encrypted by the third party vendors for some security reasons. This does not allow the modification of the IP block. This encryption put some constraints on the use of the unanticipated use of IP block, but simultaneously has an advantage which prevents the user from addition of unwanted design errors in the IP block.

• In **Hard IP** blocks it is difficult to make any changes as they are highly optimized for a specific application and have fixed layouts. They are less flexible and configurable to be reused in multiple designs. This greatly limits the areas of application. They are pre-verified by the supplier and have advantages such as reduced time for their integration and no performance issues. They are mapped to a process technology and can be uses as plug or play cores.

• **Firm IPs** is the blocks available as parameterized RTL descriptions so that the users can easily optimize and use the IPs according to the requirements of the design. The configurability and flexibility in IP block allow the designers to make the performance of the design much predictable. These IPs lies between soft IPs and hard IPs in terms of being more configurable, portable and flexible than hard IP but still much predictable than soft IPs.

Figure 3-1 shows difference between the soft, firm and hard IPs based upon the reusability, flexibility, cost, predictability and portability parameters. It is quite clear from the figure that the soft IPs are most flexible but are least predictable in performance. . The main thing required before creating a SoC is a set of reusable IPs which supports plug and play integration. Highest level of building blocks, the reusable IPs blocks with different area, power and timing configuration is the key to design a good SoC because the configurability allows the easy integration of IPs and the designer can do the tradeoff which is best suitable according to the design applications.

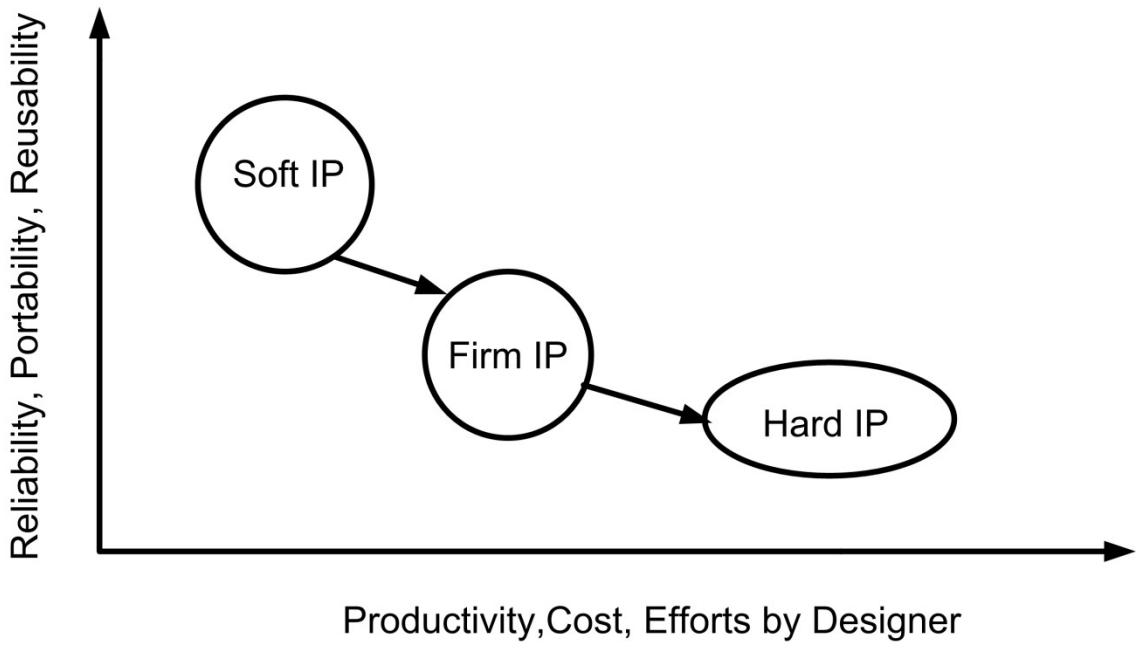

Figure 3-1 Soft vs Firm vs Hard IPs

The design process used for the creation of a reusable IP is quite different from that of traditional design process as it requires more time and design efforts. The RMM (Reuse Methodology Manual)[4] provides proper guidelines and a complete vision for the design process of a reusable IP.

3.2 A subsystem architecture with ARM AMBA bus interface

AMBA (Advanced Microcontroller Bus Architecture) is a product of ARM Corporation and is a open license standard adopted widely as the on-chip bus interface in majority of SoC designs. Basically AMBA is a set of guidelines or a standard protocol which is needed to be followed while designing. It standardizes the on chip communication between various IP blocks in a subsystem for making high performance SoC designs. Initially, the names of the protocol buses were ASB (Advanced System Bus) and APB (Advanced Peripheral Bus). Later on second version that is AMBA 2, a new bus protocol was added by arm named as AHB (High-performance Bus) which supports the pipelining and thus works in a single clock-edge. After 2nd version, the AMBA 3, AXI was included to achieve higher performance interconnect. These protocols today are used as the de-facto standards for many subsystems or IP designs because they can be used freely without loyalties and are well documented. The AMBA based designs typically consists of micro-controllers, processors along with several other peripherals blocks. It aims to

have a standardized and effective way of interconnecting the IP blocks with reuse across the multiple designs.

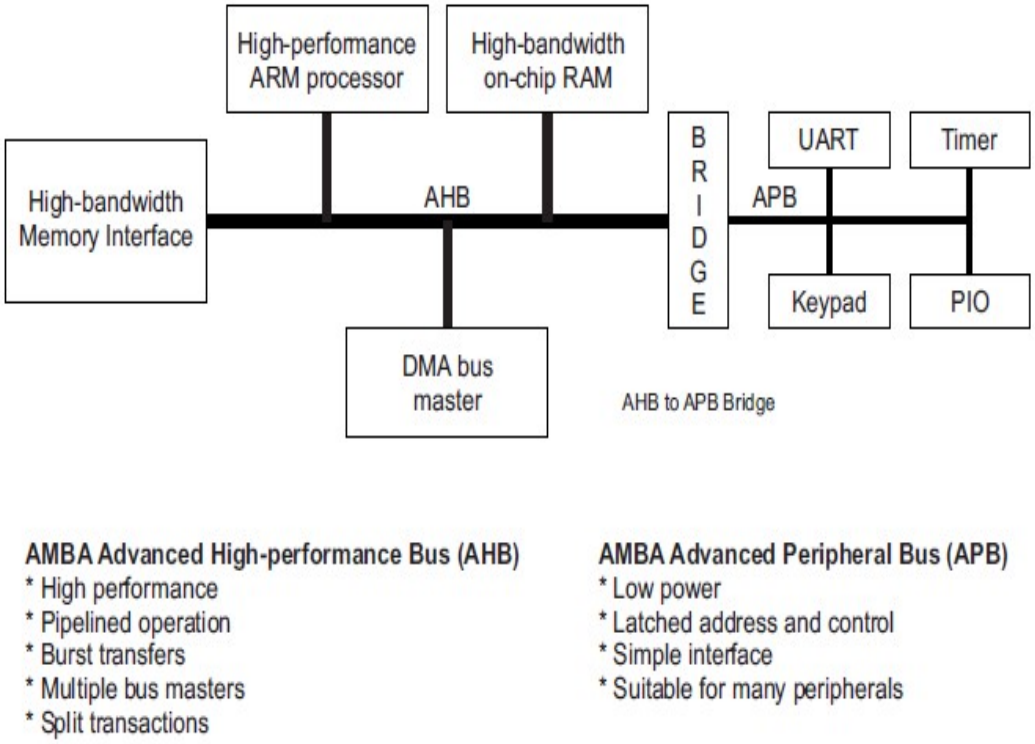

Figure 3-2 A subsystem with AMBA interface

The *Figure 3-2* represents a standard subsystem architecture based upon ARM AMBA protocol. Here, we have a high performance bus AHB on which blocks which require high bandwidth are placed. For example, high performance processors, a DMA controller, memory controllers are placed on this bus interconnect. For connecting the high performance bus with the peripheral low power bus, a bridge is used which maps AHB bus signals to the corresponding APB signals. The APB bus is connected with bridge. The bridge acts both as a master and as a slave, it acts as an AHB slave and as a master for APB. All the peripheral blocks which have low performance requirements are connected on this bus.

3.2.1 Overview of the AMBA specification

The Advanced Microcontroller Bus Architecture (AMBA) as specification defines standards for the on-chip-communication on a chip for designing the high - performance subsystems.

Many different types of bus protocols are there in the AMBA specification

The *Figure 3-3* shows the evolution of bus protocols form simple interfaces such as APB to complex one as CHI protocol within nearly two decades.

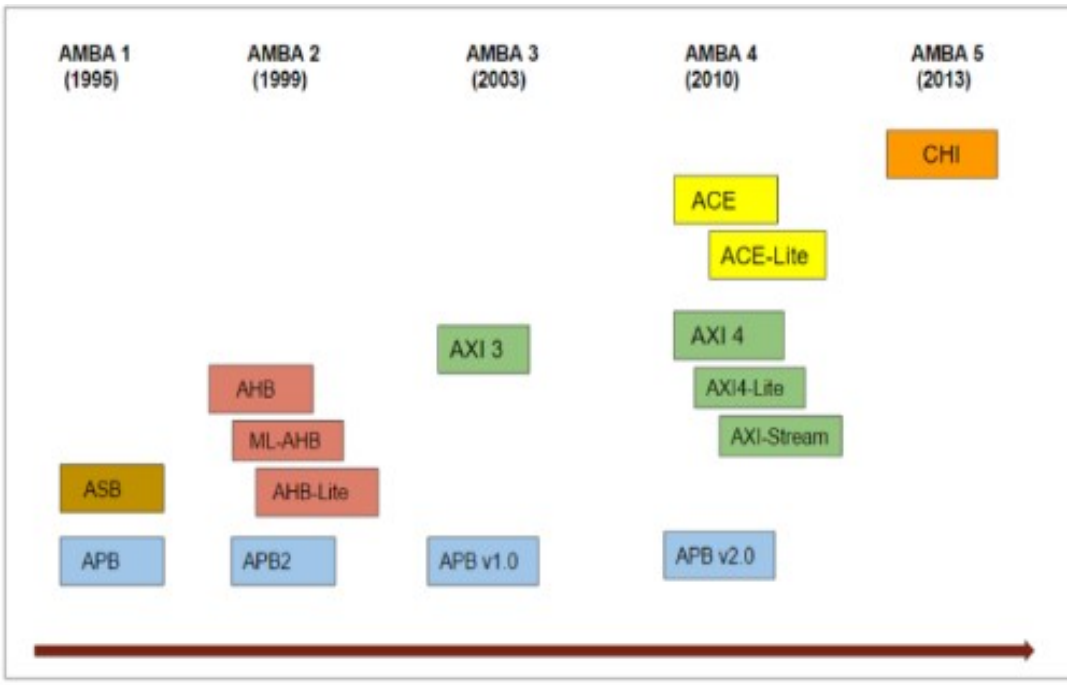

Figure 3-3 Evolution of different buses along with the SOC design trend .

The following two buses will be explained here for making the basic understanding:

• The Advanced High-performance Bus (AHB).

• The Advanced Peripheral Bus (APB).

3.2.2 Terminology Used in Design

Bus cycle A bus cycle is a clock pulse of a definite time period and for the AHB or APB protocols, it is defined from rising edge to rising edge transition.

Bus transfer AMBA AHB bus transfer is a read or write operation on data. This may take one or more bus cycles. The bus transfer is initialized by a master and is terminated by a completion response from the slave. The transfer size supported by AMBA APB includes a byte, half-word and word (32-bit). AMBA AHB supports as large as 64-bit and 128-bit data transfers. APB bus transfer is either a read or a write transaction which always requires two bus cycles. In one cycle, the address is latched and in other transfer takes place.

Burst operation As name suggests, burst operation is a multiple data transactions, started by a bus master and have a constant width of transaction to an incremental region of memory address space. The increment step of the transaction is set by the width of transfer (byte, half-word, word). Burst operation is not supported on the APB.

3.2.3 AMBA AHB

AHB is the 2nd generation of ARM AMBA bus protocol which is used to connect blocks that need high bandwidth on a shared bus. It is designed to take care of the requirements of a high-performing synthesizable design. It is designed for high-performance designs and supports multiple bus masters and slaves and is capable of doing high-bandwidth operation due to burst data transfer. It is used where the implementation of the design requires both high performance and high clock frequency. Other features of AHB include the following:

- Burst transfers.
- Pipelining.
- Split transactions.
- Single clock-edge operation. `
- Non tri-state implementation.
- 64 bit and 128 bit data bus configuration.

To connect the AHB with the APB, a bridge between both buses is implemented to enable the easy integration of the design. A bridge binds the system together and acts as a slave for AHB bus. A design

with AHB protocol can contain one or many masters, generally we can easily find a processor, DMA , memories and test interface in a system. Apart from a processor, DMA (Direct Memory Access) can also be a bus master. The definition of a slave and a master is explained below. Although any IP on AHB can be a slave or a mater. This totally depends upon the type of the peripheral connected on bus. It is general that the low- bandwidth peripherals will reside on APB interface. Some of the terminology or components in a system are:

AHB master is the one which is responsible for initiating any read or write transaction. It provides the control signals and the address signals. In AHB only a single bus master can use the bus actively at any given time which is based upon the arbitration scheme.

AHB slave is one which reacts to the corresponding read/write command given by the master. The bus slave responds back to the active master the status of data transfer such as success, failure or waiting for the data transfer.

AHB arbiter is one which is responsible for the access of the bus to different masters. An arbiter makes sure that only a single master is using the bus at a given time. This is done by implementing different arbitration algorithms such as round robin, where each master is given access sequentially based upon a definite interval of time or a transaction time. Other algorithms for example highest priority and fair access can also be implemented based upon design needs.

AHB decoder is the decoding unit to decode the address where the transaction is needed to be performed. It provides a chip select signal corresponding to the slave on which transaction is to be performed. Normally a centralized decoder is used for the implementation of AHB based designs.

3.2.3.1 Basic AHB Transfer

A transfer is normally done in two different pats or phases which are shown below:

- The address phase, where the address is latched in a single clock cycle.
- After data phase is the data phase, where the transfer of data takes place. It may be single or multiple cycles depending upon the **HREADY** signal.

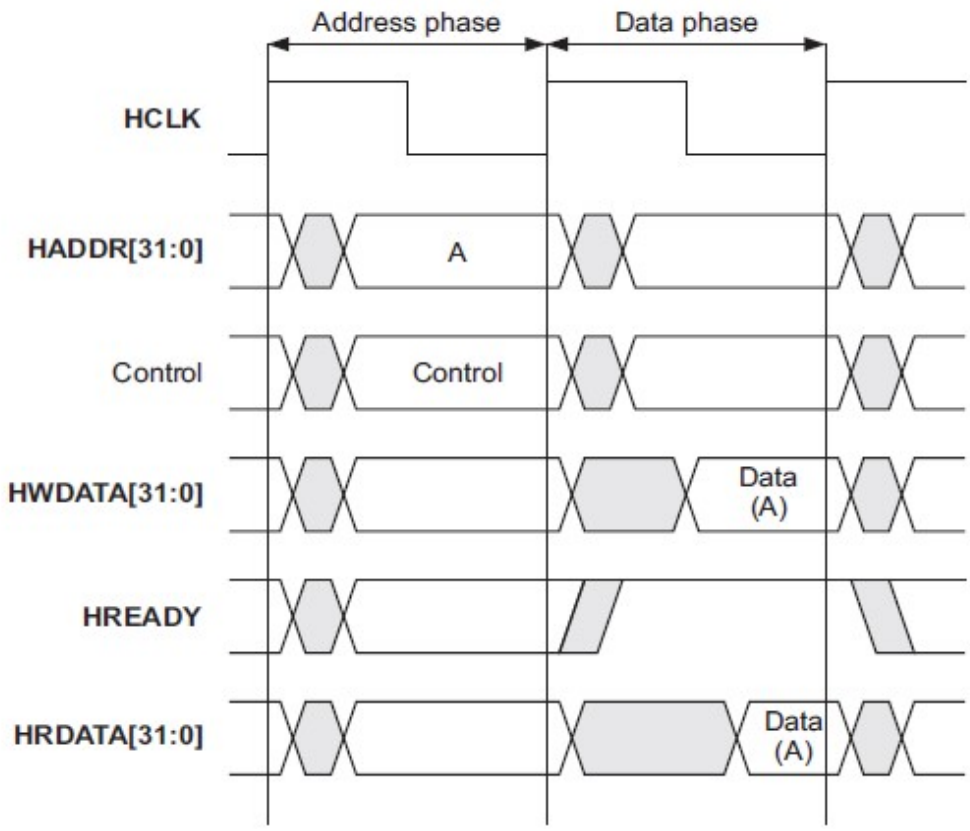

Figure 3-4 Basic AHB Transfer

A simple AHB transaction is shown above in the *Figure 3-4. The transaction is explained as following:*

- Master gives the control and address signal when the rising edge of the clock comes. In AHB, the clock signal is denoted by **HCLK** signal as shown above in figure.
- After the clock, the control and address signals are sampled by the when the next rising edge of the clock occurs.
- When the sampling of the signals is done, the slave gives the response signal which is **HREADY** signal as shown in figure. This signal is high to represent that the slave is ready for transaction while low is used to add a wait state in the transaction. This response signal is then sampled by the master when the next clock edge occurs.
- For a write transaction, the master has to keep the data stable for the next cycles until the transaction is completed. This is done to avoid any unwanted data values which can be caused by the violation of setup and hold time.
- While performing a read transaction, it is not necessary for the slave to give the valid data until the end of the transfer.

3.2.4 AMBA APB

The design of APB is such that it consumes minimum possible power had have a less complex usage. It is recommended to use APB for less complex, low bandwidth slaves. The APB seems to be a local secondary bus, but it actually acts as a single AHB slave devise. APB is used as an extension with low power consumption to the main bus of the system. The bridge which is used to connect AHB to APB is an AHB slave module is a handshake mechanism for the control signals and other signals .APB is suitable for hose peripherals which consumes low bandwidth and does not need pipelined operations and burst modes of transfer. The transactions on APB take place on the rising clock-edge. The prefix **P** is used to represent an APB signal for example **psel, penable** etc. The flexibility and ease of integration makes it best suitable for use in designs with low power consuming peripherals. The main features and advantages and features of the APB are:

- Consumes low power.
- Easy to implement in the designs
- Timing analysis easy as it uses simple data transaction phases
- Have space for test signals insertion.
- Can connect many slaves operating on low frequency.
- Control signals are always valid during access.
- During the un-pipelined operation, it has zero power consumption.

3.2.4.1 Operating states in APB

The transaction in APB takes place as shown by *Figure 3-5*. The figure shows different operation states of a APB transaction. The transaction starts with an IDLE state where the signals psel and penable are low, this state indicates that no APB transfer is taking place. At this state the pready signal is high which indicates that the slave is ready for the transaction. Whenever a transaction starts, the psel signal is set high, but still the penable signal is low. This state is represented as setup state and points to the start of the transaction.

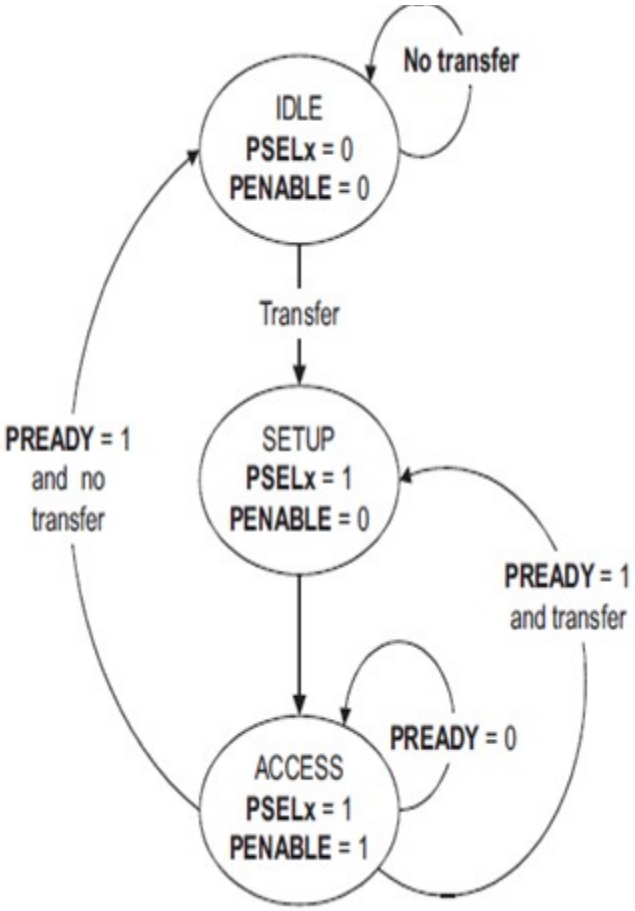

Figure 3-5 FSM for APB operation

After setup, the penable signal is set high and the access state of the starts. In this stage, the transfer of data is done. The pready signal automatically gets low until the transfer is complete. If there is another transfer after the ongoing transfer the pready is set to one and the setup state is approached while in case of no transfer , IDLE state is default state.

Figure 3-6 shows the APB transaction with and without wait states inserted. The wait state are inserted by the slave to tell master that the slave is unable to complete the transaction in given clock cycle and requires more clock cycles to complete the transactions. It is represented by the **pready** signal. High pready means transaction can be completed and low means addition of wait state.

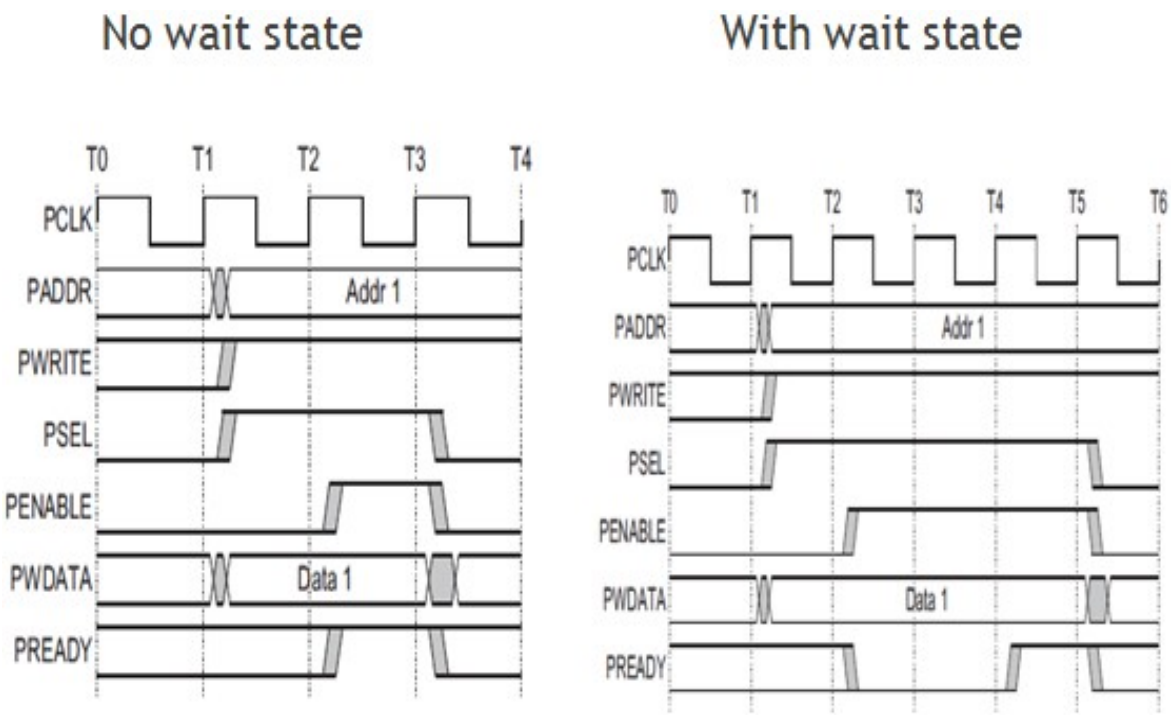

Figure 3-6 AHB transaction with or without wait states.

3.2.5 AMBA Bus interconnection

The AMBA AHB bus interconnections are done based upon a selection scheme which is provided by using multiplexers. These multiplexers are used to select the address and control signals corresponding to the transfer to be performed. *Figure 3-7* shows the implementation of the bus interconnection with central multiplexer interconnection scheme. In figure, the arbiter is used to drive the select signals of the multiplexers which correspondingly select the address, data or control signals. Thus the transfer of data takes place between masters and slaves based upon the given address and data signals.

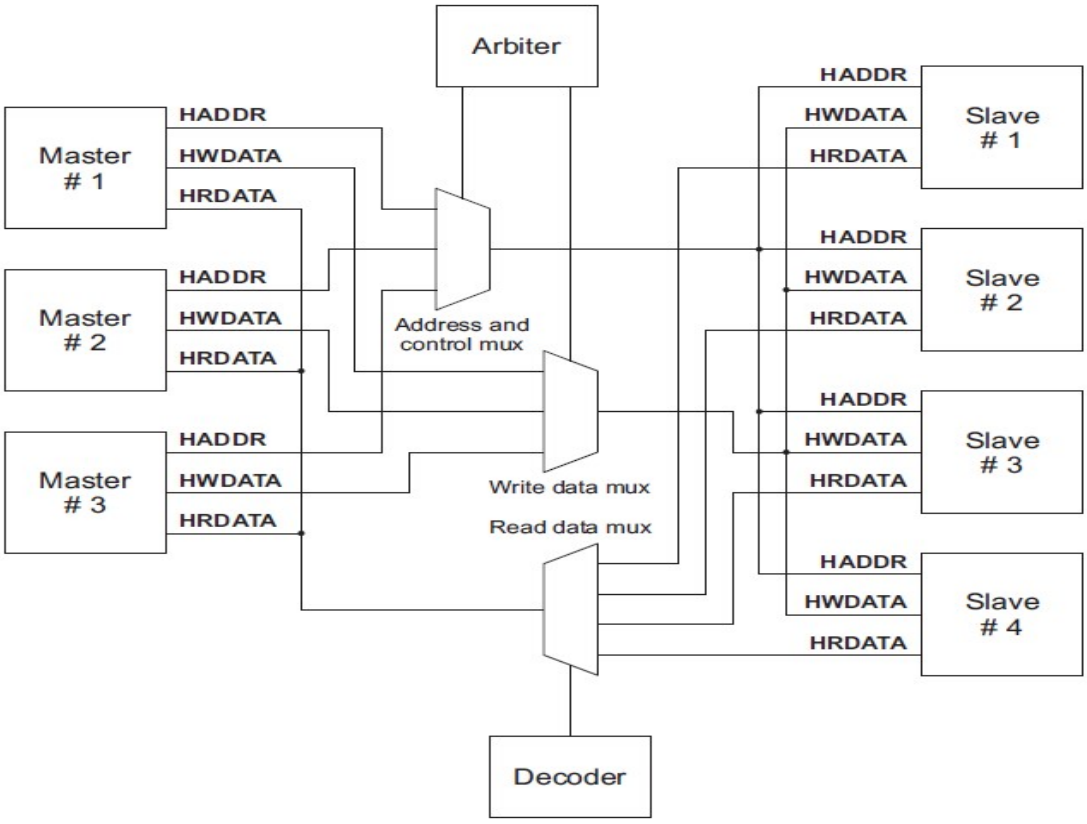

Figure 3-7 AMBA AHB design with three masters and four slaves.

3.3 Generic Architecture used for HSM Subsystem

Figure 3.8 shows the architecture used for the implementation of Hardware Security Module in this thesis. The components or IPs shown in the architecture are reusable IPs and are highly configurable. They are packaged according to the IP-XACT standards. They consist of AMBA Bus Interface. We can add or remove more components according to the specifications required.

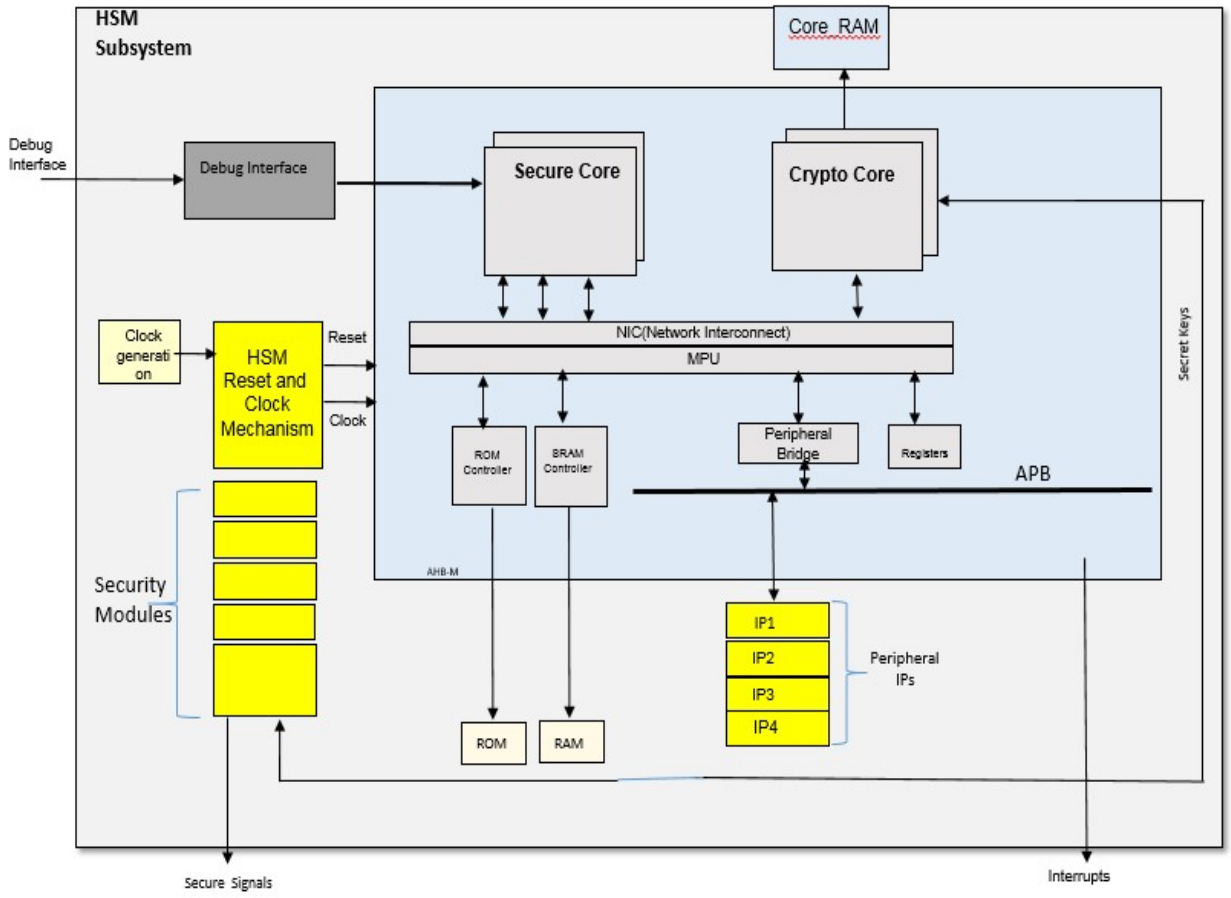

Figure 3-8 Generic architecture for a Subsystem Design.

The architecture consists of various security IPs required for (HSM) Hardware security module on a standard bus architecture. Here ARM AMBA bus is used for connecting different IPs. All the IPs used in HSM are either developed in house or are acquired from third party vendor. The subsystem is divided into small blocks to ease the complexity of the design. Secure core is used for normal operations. It is a master and initiates the transaction. It can communicate with different IPs. The crypto core has been used to perform the secure operations. Since, only single secure core can also do the secure operations or they can be achieved by the software, the need for a second crypto core is because a dedicated core can divide the system load and can enhance the system performance. HSM Reset and Clock mechanism provide the reset and clock signals based upon the requirements. IRC and PLL are also used to generate and convert the clock to required frequency. Network Interconnect is basically a decoder and is used to provide the bus arbitration mechanism for accessing different IPs by the cores. RAM and ROM are used to store the volatile and nonvolatile data respectively. A corresponding RAM controller and ROM controller is used to access these modules. Memory Protection Unit is placed between the core and the memories to prevent

the invalid access to the memories. Other IPs such as security IPs are accountable for storage, comparison and access of secure data or secret keys. Debug interface is to debug the subsystem in case of failure.

3.4 Integration Flows based on IP-XACT™ Standards

3.4.1 IP-XACT

In this section basics of IP-XACT which are necessary foe packaging are has been discussed. The IP-XACT has a standard VLNV format to name the component. Vendor, Library, Name and Version are used to identify the component. As it is designed by SPIRT, a namespace spirit is used which resolves to an address. There is need to configure the components in accordance with the IP-XACT standards. IP-XACT is a standard which is implemented for the packaging, integration and reusing IPs in an automated tool flow. It describes the metadata of sub-system and IP designs. The description of all the meta-data is presented in a standard XML (Extensible Markup Language). The advantage of using XML is that it is readable both by humans and machines. It is a part of SPIRIT consortium and is used as a standard for automating the configurations of IPs in tools flows.

```
Generated with Magillem 5.11.2.1 ( build id: 201706130950_5.11.2.1 ) at 11/2/18 2:27:
->
spirit:component xsi:schemaLocation="http://www.spiritconsortium.org/XMLSchema/SPIRIT/
l/component.xsd" xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4" x
   <spirit:vendor>adg.st.com</spirit:vendor>
   <spirit:library>ip.dig.com</spirit:library>
   <spirit:name>sscm2_top</spirit:name>
   <spirit:version>v00.00</spirit:version>
   <spirit:busInterfaces>
       <spirit:busInterface>
           <spirit:name>IPS</spirit:name>
           <spirit:description>IPS interface</spirit:description>
           <spirit:busType spirit:library="busdef.ips" spirit:name="IPS" spirit:vendor
           <spirit:abstractionType spirit:library="busdef.ips" spirit:name="IPS_rtl" s
'/>
           <spirit:slave/>
```

Figure 3-9 An IP-XACT file

3.4.2 IP-XACT based flow

The IP-XACT standard can be applied in various parts of a typical SoC design flow as depicted in *Figure 3-9*

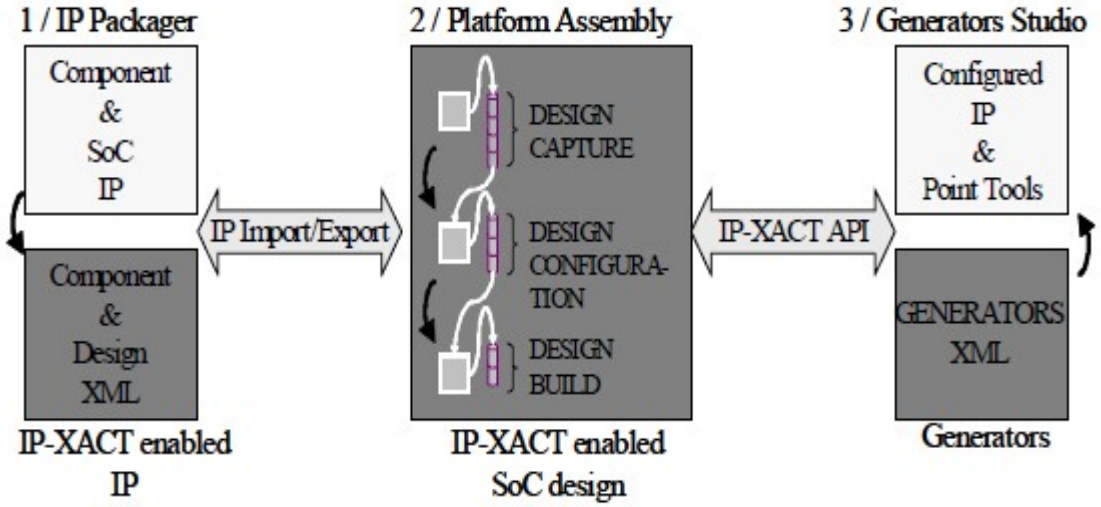

Figure 3-10 IP-XACT flow

From above figure, it can be seen that the flow consists of three stages which are:

- IP-XACT Packaging.
- Platform Assembly
- Generators

IP Packaging: The main motive behind the packaging is to convert the RTL design into a XML file which is easy to integrate in designs and is highly portable. The IPs packaged according to IP-XACT standards which are described in [17]. The packaged file consists of a model of design in a well versed format and have different views such as simulation, synthesis etc. This later helps in extracting the information related to the specific view. The steps involved for the packaging of IPs are given in the APPENDIX.

Platform Assembly: Assembly is the process of concentrating the design from different sources into one. Packaging has made easier to extract the components, configurations and use them in different tools. In this thesis, the assembly has been performed by using a tool called Magillem Assembler. The steps are given in the APPENDIX. The assembly consists of three sub-parts as shown in *Figure 3-9.*

- **Design Capture:** In design capture, all the blocks or IPs which are needed to be integrated are given as a file-set to the assembly tool in the form of XML files. Thus a metadata and catalog of the design is created. This step captures the default parameters associated with the IPs.
- **Design Configuration:** This part consists of passing the parameters to the design through a parameter file which will override the default configuration parameters. The parameters are changes the configuration of the design. These are the elementary key in the concept of reuse of IPs in the designs.
- **Design Build:** In this part, all the IPs are connected together by running the assembly. The connections are mentioned by the user in connection file which generally present in .csv format. In this part, the assembler stitches the design and makes design libraries which will later be used for compilation and verification of design.

Generators*:* This part is generally the scripts which will generate the final XML of the design. These are also made according to IP-XACT standards and are compatible with tool flows.

The *Figure 3-11* as shown below is the proposed flow and is used for the integration of HSM Subsystem in the thesis.

In *Figure 3-11,* it is shown that the IP blocks are available in form of RTL . To integrate these to form a working subsystem Packaging is done by Magillem tool. After Packaging, each IP is available in its IP-XACT representation. The design is then assembled or integrated by the assembler. After assembly, the RTL of top design is generated. It is then again packaged to form a top-design XML file. Along with XML a catalog is created which have the location of all the RTL files used in design. The integration flow includes the usage of IP-XACT standard to generate a XML file of different IPs. The XML files generated by the process of packaging which will be later used for many purposes like integration, creating document etc. The XML file contains information of design, the components used in design and other descriptions. The main advantage of the XML is that it can be reused across different platforms. A sample XML of an IP consists of information of VLNV (Vendor, Library, Name and Version). It helps in identifying the IPs.

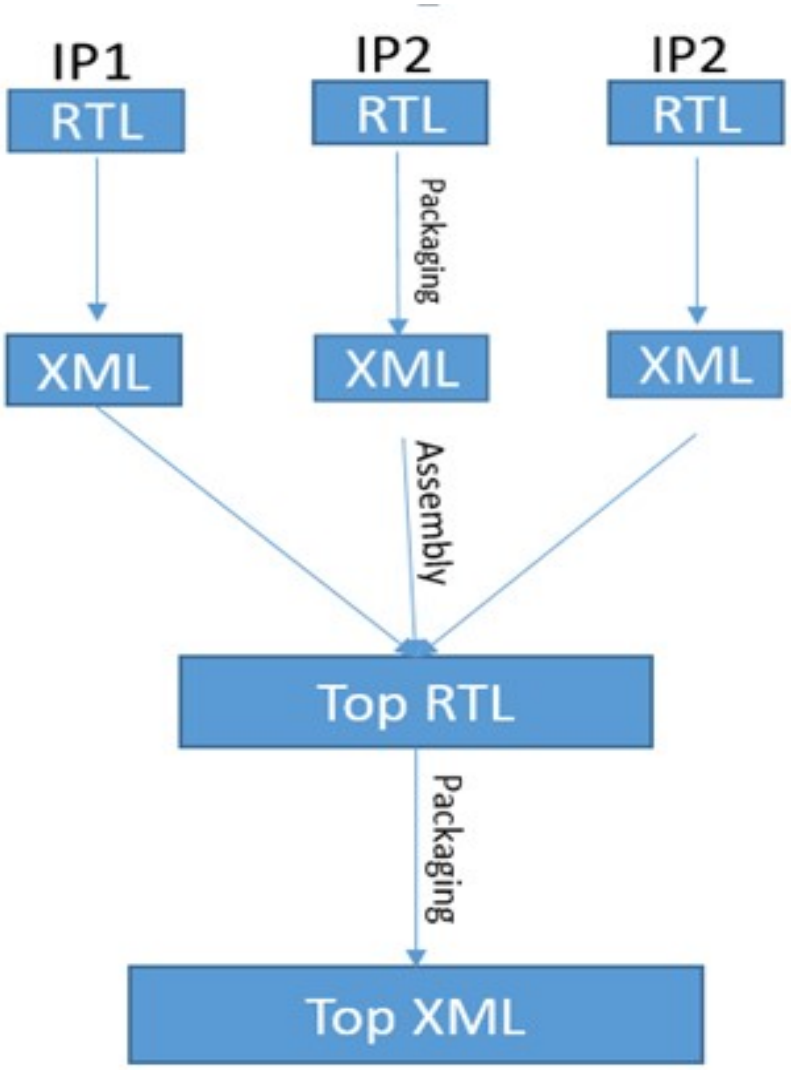

Figure 3-10 Flow used for Integration.

3.5 Lint and CDC (Clock Domain Crossing)

3.5.1 Lint

After the completion of writing the code, lint is the method to check the correctness of code. Linting is the standard for syntactically correctness of code. It consists of some set of rules which must be followed in order to get a synthesizable code. Spyglass is used for lint checks. The RTL is checked based upon the

design methodology of the company. The design methodology consists of some of the rules or guidelines which are necessary to follow in order to make the RTL compatible with other design processes, for example by following or doing the lint on RTL the user will never get issues while using the RTL on later stages such as synthesis, DFT insertion etc.

3.5.2 CDC (Clock Domain Crossing)

A CDC (Clock Domain Crossing) design is one which consists of multiple clock domains which are asynchronous or have a different phase relation with respect to one another. Transfer of data from a clock domain to another domain may lead to the setup and hold time violations in the flip flops which further cause problem in the functionality of the design. The functionality validation of the designs is very complex and expensive task. The traditional simulation techniques cannot model the effect of meta-stability caused by the setup or hold time violations.

3.5.2.1 Metastability

A clock domain crossing is a condition which occurs whenever data is transferred from one flop to another flop and both the flops are driven on different clocks. In big designs which contain multiple clock domains, where signals cross frequently asynchronously, the problem of CDC comes into picture. A clock domain is a part of a digital system or design which operates on a particular frequency.

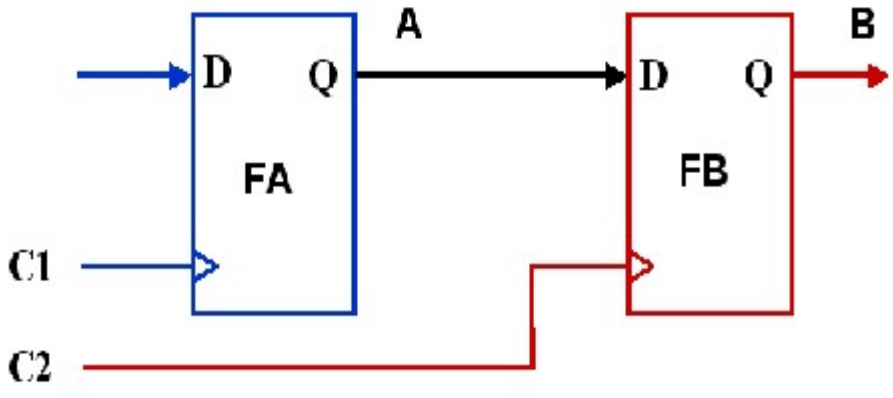

Figure 3-11 Asynchronous flip-flops and CDC condition.

The above *Figure 3-12* can illustrate the condition of CDC. Consider both the flops FA and FB are working on different clocks say C1 and C2, D and Q are the output pins of the flops respectively. Let A be the output of FA and B is the output of FB. Now, if A changes its value near the active edge of the second clock, this may cause setup or hold violation at the second flop or destination flop FB. Due to this, B does not get a correct value. ie, it oscillates for an indefinite period of time and then settles to an logic value which cannot be determined. Therefore, the output is uncertain and may or may not have a stable value before the next rising edge of the second clock C2. This condition is called meta-stability, where the value of signal is uncertain. The meta-stability cannot be avoided in multi clock designs but its effect can be neutralized.

For example, in *Figure 3-13*, the input signal A changes its value near the clock edge of clock C2, this causes the violation of setup time, further leading to the condition of meta-stability. As a result, it settles to an value either 0 or 1 as shown by signals B1 and B2.This may cause problems such as high current flow or different values at different nodes in a design leading to malfunctioning of design.

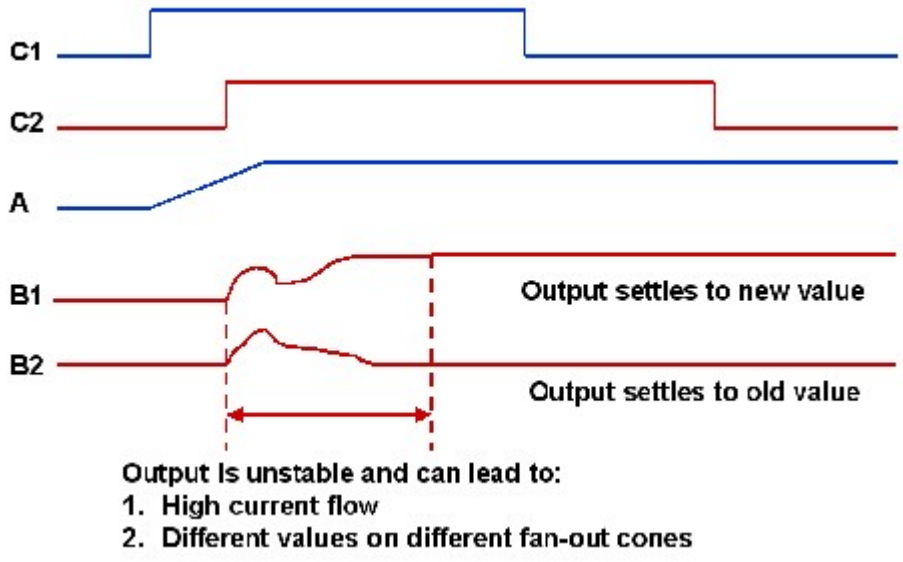

Figure 3-12 Metastability caused in asynchronous clock domains

3.5.2.2 Data Loss

The data transferring from one clock domain to another can easily be lost in multi-clock domain designs. When a source flip-flop generates a new data, it may not be received/captured by the flip-flop in destination clock domain in the first clock cycle due to the meta-stability. There is no data loss as long as every data/transaction is captured in the destination domain. This can only be achieved if the source data remains stable near the rising edge of destination flop and there is no violation of setup and hold time.

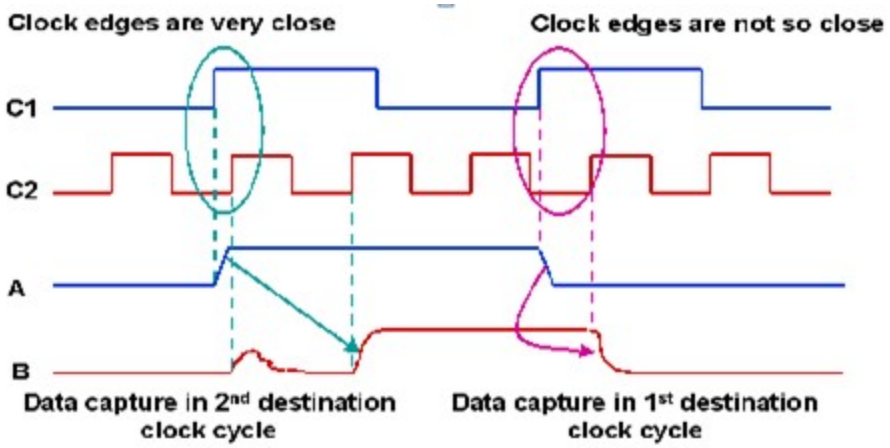

Figure 3-13 Data capture in different clock cycles

In *Figure 3-14*, if the rising edges of clocks C1 and C2 are very close, then the data A will not be captured in the first cycle of clock C2. It will require another cycle of clock C2 to capture the data properly as shown in the figure. However, the data is properly captured in the first clock cycle of C2 after the transition of data A as depicted by the figure. It occurs because there is sufficient time when data is changing and the rising clock edge of clock C2 is coming.

The condition of loss of data can generally be seen in the designs where there are multiple clock domains and the data is transferred from a fast clock domain to a slow clock domain. In this case the data is not captured properly and caused the misbehavior of the design. Such problems must need to be avoided in the designs and can be solved by implementing certain design techniques one of which is mentioned below.

3.5.2.3 Multi-Flop Synchronizer

The problem of meta-stability can be avoided by adding some special flip-flops known as synchronizers in the destination clock domain. The use of synchronizers is to neutralize the impact of meta-stability.

When the meta-stability occurs, the synchronizers provide the sufficient time for oscillations to settle down to a stable output value. A commonly used two flop synchronizer is shown in figure[].

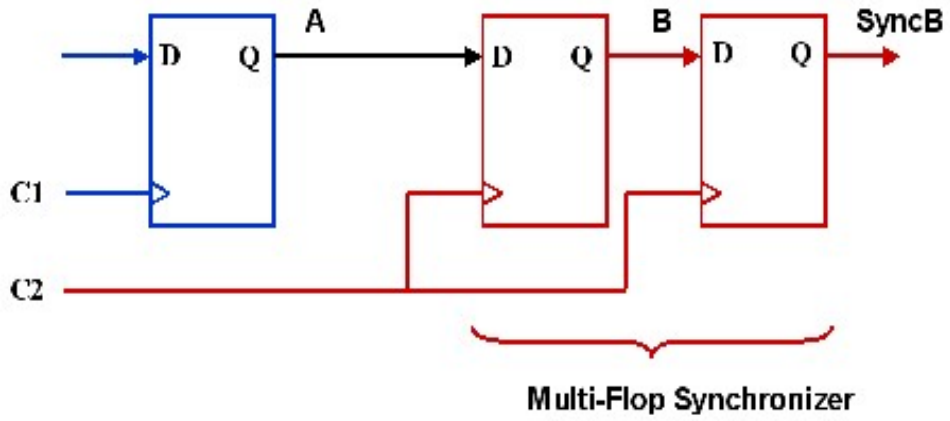

Figure 3-14 A two flop synchronizer.

This type of synchronizer is mainly used for control signals which are singe or multi-bit. There are many other techniques which are used for multiple bit data signals such as handshake, MUX recirculation and FIFO.

3.6 Integration Verification

This part of the SoC design flow requires most of the time of the design. Before passing the design to the verification team, it is necessary for the SoC designer to carry a small verification at his end. This verification includes the basic read, write checks on different IPs. This is done in order to check whether the proper integration of IPs has done or not. The rest of the functionality checks are done by the verification engineer. Normally, the verification is done by creating the test-bench for the design and checking the waveforms according to the stimulus applied. The verification of a SoC is very complex task and require a lot of efforts. Therefore it is generally done in parts. the IPs are verified separately so that their behavior and functionality is known previously. In the SoC verification that is done here, the UVM methodology has been used. A test-bench environment is created by the verification team, the design is placed parallel to the test-bench environment. The main task in this methodology is the test-case design. To check the design correctness, a test case should be such that it will give maximum code and functional

coverage. The basic test-bench structure consists of a test-bench wrapper over the design under test. The structure consists of generators, drivers, monitors and scoreboard.

The generators are the input stimuli generating functions, a generator can be a sequence generator, or a random number generator, the drivers are used to drive the inputs of design under test with the stimuli generated by the generators. A driver can be any logic placed to provide the input. The monitors and score board are for monitoring the output and comparing them with the desired one. Thus the design verification becomes easy in such environment.

Chapter - 4

Results and Discussion

This chapter contains the work done in designing of HSM subsystem by using the above design flow.

4.1 Packaging of IPs:

The various components used in the subsystem are needed to be in accordance to IP-XACT standard to ease the integration task. So each component is packaged onto a .xml file which contains the metadata of IP.

Input files:

- File-set of the modules of IP in HDL (Hardware Description Language).
- Bus definition information.
- Generic Parameters.

Tool used:

- Magillem.

Output:

- The xml file of the IP is generated.
- The *Figure 4-1* shows the IPXACT file of a generic IP “Watchdog Timer”

The original file is very large to be displayed in this document so, a small part of file has been displayed.

```
  Generated with Magillem 5.10.1.4_engineering ( build id: 201612221150_5.10.1.4_engineering ) at 1/20/19 5:17:34 PM
-->
<spirit:component xsi:schemaLocation="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4 http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4/componer
    <spirit:vendor>adg.st.com</spirit:vendor>
    <spirit:library>ip.dig.sys</spirit:library>
    <spirit:name>swt_ips</spirit:name>
    <spirit:version>v01.00</spirit:version>
    <spirit:busInterfaces>
        <spirit:busInterface>
            <spirit:name>CLK</spirit:name>
            <spirit:description>IPG Clock</spirit:description>
            <spirit:busType spirit:library="busdef.clock" spirit:name="clock" spirit:vendor="spiritconsortium.org" spirit:version="1.0"/>
            <spirit:abstractionType spirit:library="busdef.clock" spirit:name="clock_rtl" spirit:vendor="spiritconsortium.org" spirit:version="1.0"/>
            <spirit:slave/>
            <spirit:portMaps>
                <spirit:portMap>
                    <spirit:logicalPort>
                        <spirit:name>CLK</spirit:name>
                    </spirit:logicalPort>
                    <spirit:physicalPort>
                        <spirit:name>ipg_clk</spirit:name>
                    </spirit:physicalPort>
                </spirit:portMap>
            </spirit:portMaps>
        </spirit:busInterface>
        <spirit:busInterface>
            <spirit:name>SWT_CLOCK</spirit:name>
            <spirit:description>IPG SWT Clock</spirit:description>
            <spirit:busType spirit:library="busdef.clock" spirit:name="clock" spirit:vendor="spiritconsortium.org" spirit:version="1.0"/>
            <spirit:abstractionType spirit:library="busdef.clock" spirit:name="clock_rtl" spirit:vendor="spiritconsortium.org" spirit:version="1.0"/>
            <spirit:slave/>
            <spirit:portMaps>
                <spirit:portMap>
                    <spirit:logicalPort>
                        <spirit:name>CLK</spirit:name>
                    </spirit:logicalPort>
                    <spirit:physicalPort>
                        <spirit:name>ipg_clk_swt</spirit:name>
                    </spirit:physicalPort>
                </spirit:portMap>
            </spirit:portMaps>
        </spirit:busInterface>
        <spirit:busInterface>
            <spirit:name>RESETn</spirit:name>
            <spirit:description>IPG Reset</spirit:description>
            <spirit:busType spirit:library="busdef.reset" spirit:name="reset" spirit:vendor="spiritconsortium.org" spirit:version="1.0"/>
            <spirit:abstractionType spirit:library="busdef.reset" spirit:name="reset_rtl" spirit:vendor="spiritconsortium.org" spirit:version="1.0"/>
            <spirit:slave/>
            <spirit:portMaps>
```

Figure 4-1 IP-XACT Packaged file in XML format.

Figure 4-1 shows the generated IP-XACT file of a peripheral “Watchdog Timer” in XML file format. The XML file is highly portable and contains the information of the ports, bus definition and file-sets in different views which are simulation and synthesis. Such files have been generated for each IP component in the design and then are integrated together.

4.2 Integrating the Design

After all the components are packaged into the IP-XACT format, they are ready to be integrated into the design. The assembly is also done by magillem tool.

Input files:

- HDL list of all the IPs which are to be integrated.
- Connection file in .csv format defining the ports connections between different modules.
- Parameters of generic IPs which are to be passed in top file.
- Bus definitions which were used in design.

Tool used:

- Magillem.

Out-put:

- Design top.v file which contains the whole design.

Figure 4-2 and 4-3 shows a small part of the top design file of HSM subsystem. Some of its parameters are hidden because of confidential information.

```
crc #(
      .CRC_CNTX_NUM (1)
      )
hsm_crc (
      .ipg_clk(                    hsm_mc_cgl_ipg_clk_hsm_crc_sig           ),
      .ipg_hard_async_reset_b(     hsm_mc_rsl_ipg_hard_async_reset_b_sig[0] ),
      .ips_module_en(              ips_read_mux_0_ips_module_en_out_sig[10] ),
      .ips_addr(                 { hsm_aips_lite_ips_addr_sig[13:10],
                                   hsm_aips_lite_ips_addr_sig[9:8],
                                   hsm_aips_lite_ips_addr_sig[7:6],
                                   hsm_aips_lite_ips_addr_sig[5:2] }        ),
      .ips_seq_access(             hsm_aips_lite_ips_seq_access_sig         ),
      .ips_spv_access(             hsm_aips_lite_ips_supervisor_access_sig  ),
      .ips_test_access(            DEFAULT_TIED_MAG_hsm_crc_ips_test_access ),
      .ips_rwb(                    hsm_aips_lite_ips_rwb_sig                ),
      .ips_byte_en(              { hsm_aips_lite_ips_byte_31_24_sig,
                                   hsm_aips_lite_ips_byte_23_16_sig,
                                   hsm_aips_lite_ips_byte_15_8_sig,
                                   hsm_aips_lite_ips_byte_7_0_sig }         ),
      .ips_xfr_wait(               hsm_crc_ips_xfr_wait_sig                 ),
      .ips_xfr_err(                hsm_crc_ips_xfr_err_sig                  ),
      .ips_wdata(                  hsm_aips_lite_ips_wdata_sig              ),
      .ips_rdata(                  hsm_crc_ips_rdata_sig                    )
      );
```

Figure 4-2 Top design file after integration.

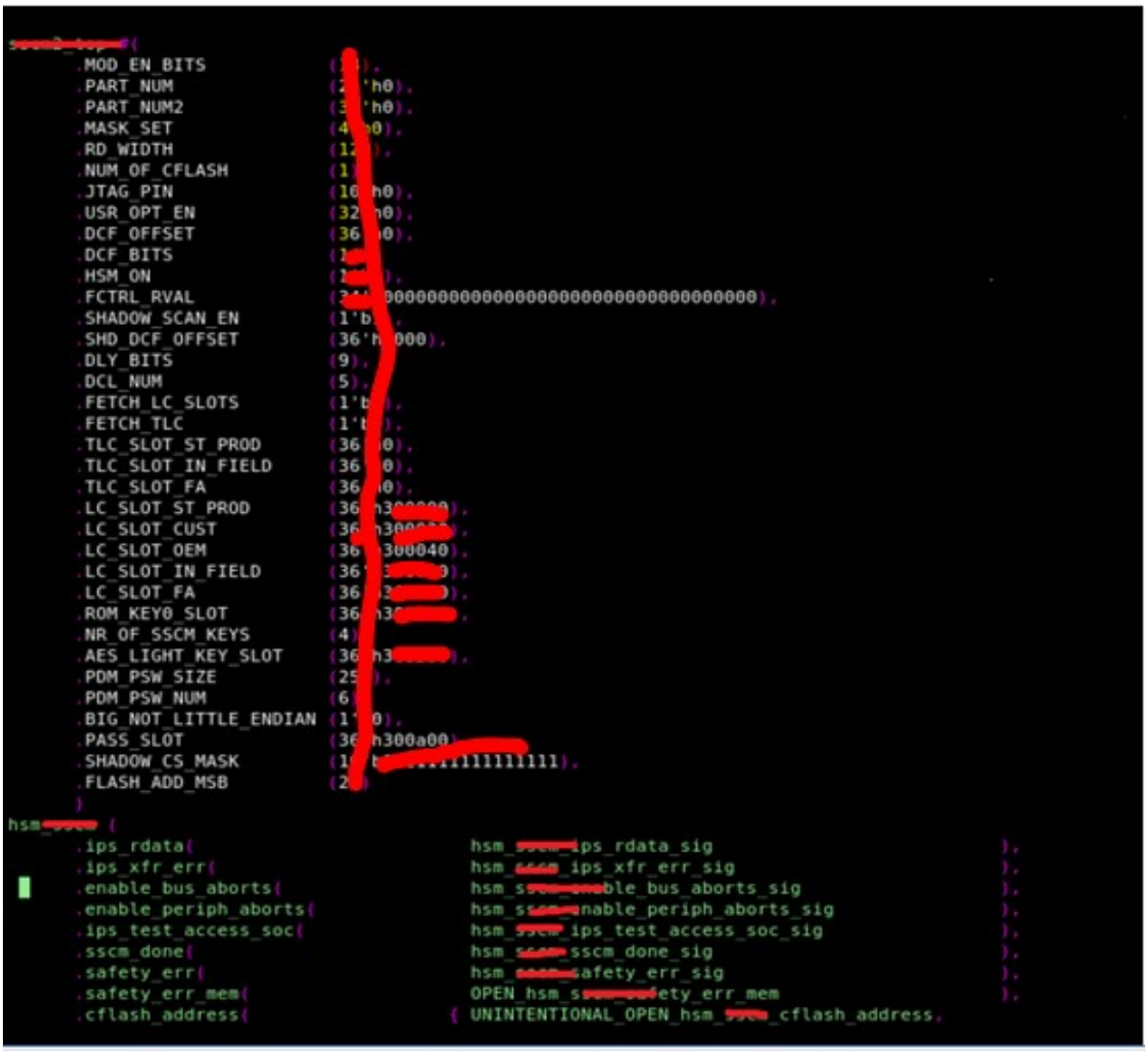

Figure 4-3 Top design file after integration.

The part of file shown here in the *Figure4-3* contains an IP instantiated in HSM along with parameters used in the design. Top file included all the components instantiated in the design.

This file is again packaged into IPXACT format so the design can be reused easily in cdc checks, verification and synthesis.

4.3 Lint and CDC

This step is performed to check if there is any violation of setup and hold times in the design. The CDC and Lint Checks are done on the design.

Input:

- All the design files bundle which is extracted from top IPXACT file of the design
- Clock and Reset Constraints.
- Any parameters which are needed to be passed into design.

Tools used:

- Recital: To extract the required fileset from IPXACT file and make a compile script which is compatible with other tools.
- Spyglass: To perform CDC and Lint checks.

Output:

- CDC and Lint check reports.

In GUI mode of spyglass, the errors and warnings can be seen and debugged. The report of the CDC verification is clean as shown in *Figure 4-4*. There are some warnings which are generated by the tool according to the set of rules defined. These warnings can be waived as there is no severe impact of these issues in the design.

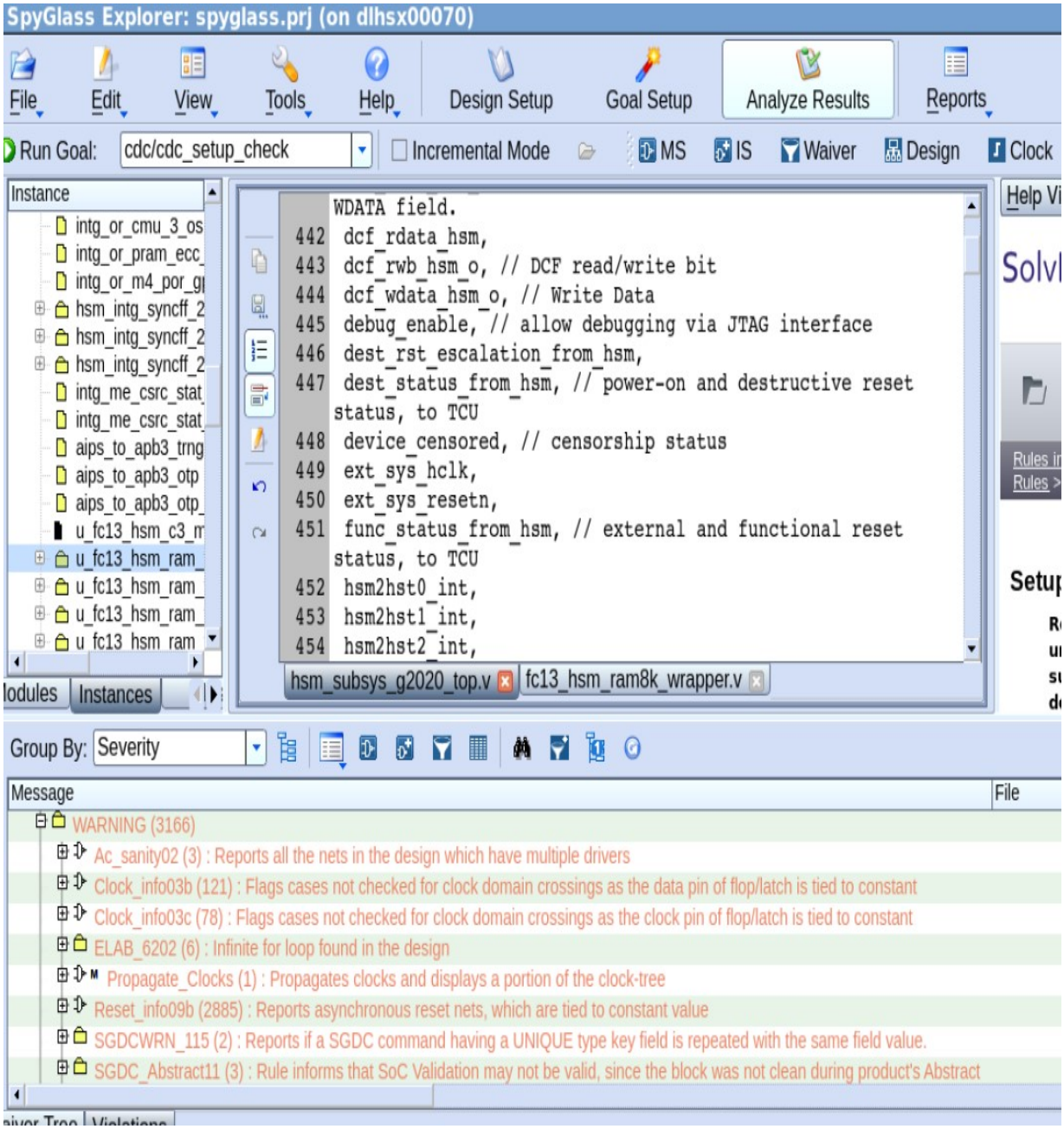

Figure 4-4 Figure shows CDC check run on the design

The results of CDC checks are in accordance with the design and there is no error which needs any attention.

4.4 Verification of subsystem (read/write checks) using UVM

After integration, the formal read/ writes check is done on SoC .After that ,the integrated design is send to the verification team for the complete functional verification of the design.

Input:

- Created Test-case file in C which contains the test-cases .
- Design files.
- Test-bench Environment.

Figure 4-5 shows the input test by designed in C with a test case to verify a feature of the design.

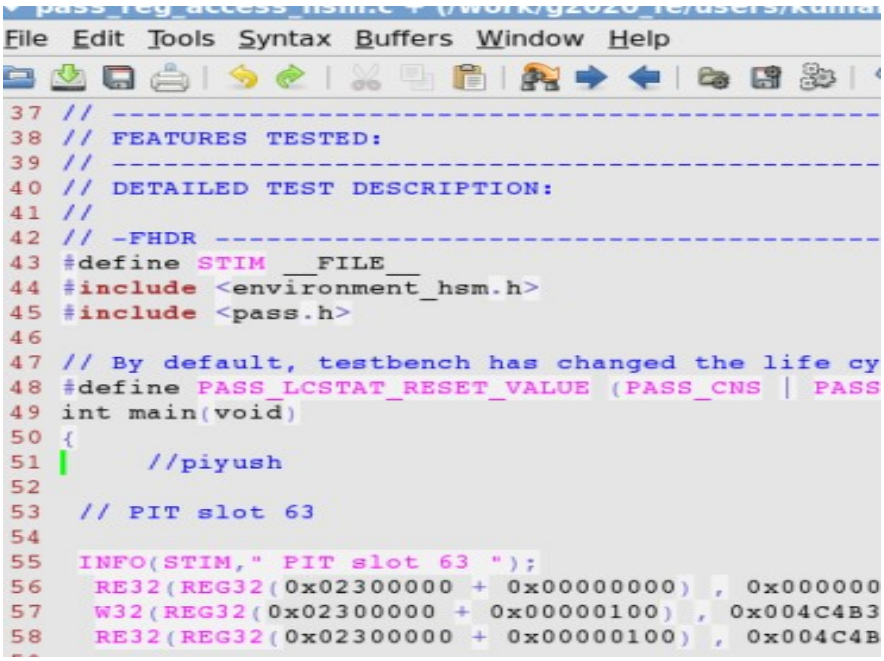

Figure 4-5 Input Test file for verification.

Tools used:

- Cadance Xcelium.
- UVM environment.

Output:

- UVM test report for the test-case.

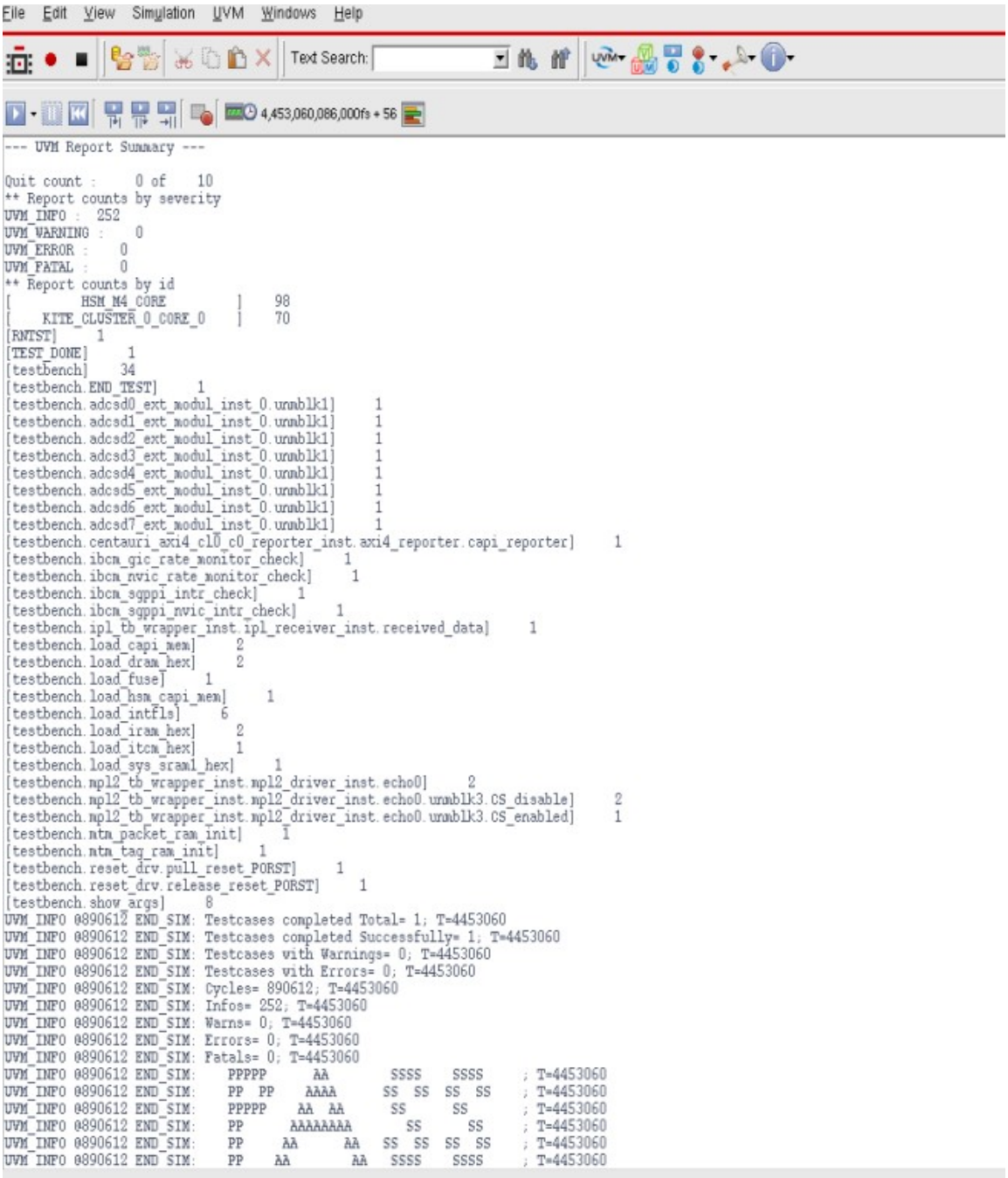

Figure 4-6 Test report for the given test

In Figure 4-6 ,the test case to check the read and write has been successfully done and the status of UVM report indicates no error during the test.

4.5 Reset Lift of HSM

The Figure shows when the HSM is booted up, if the Reset has lift properly or not. Here is the description of the sequence of events that take place during the three reset phases of HSM.

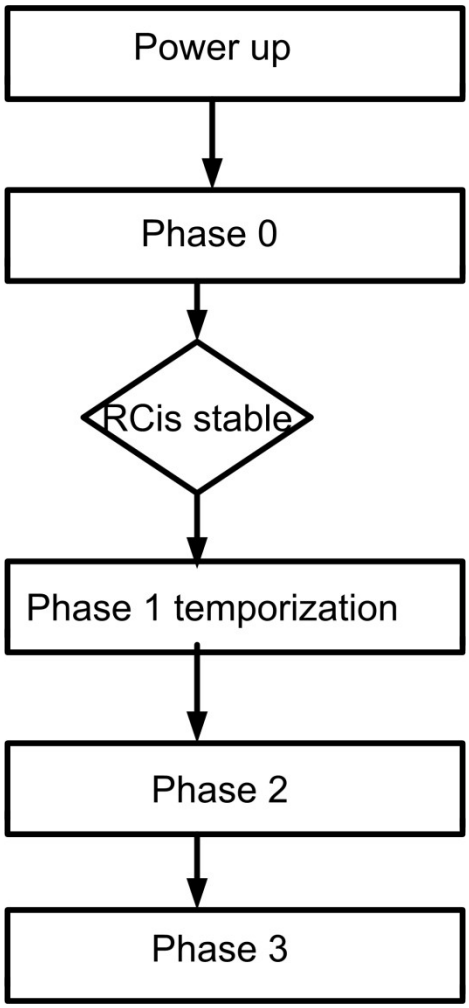

Figure 4-7 Power up scheme for HSM

The results can be seen in the *Figures 4-8* and 4-9 below

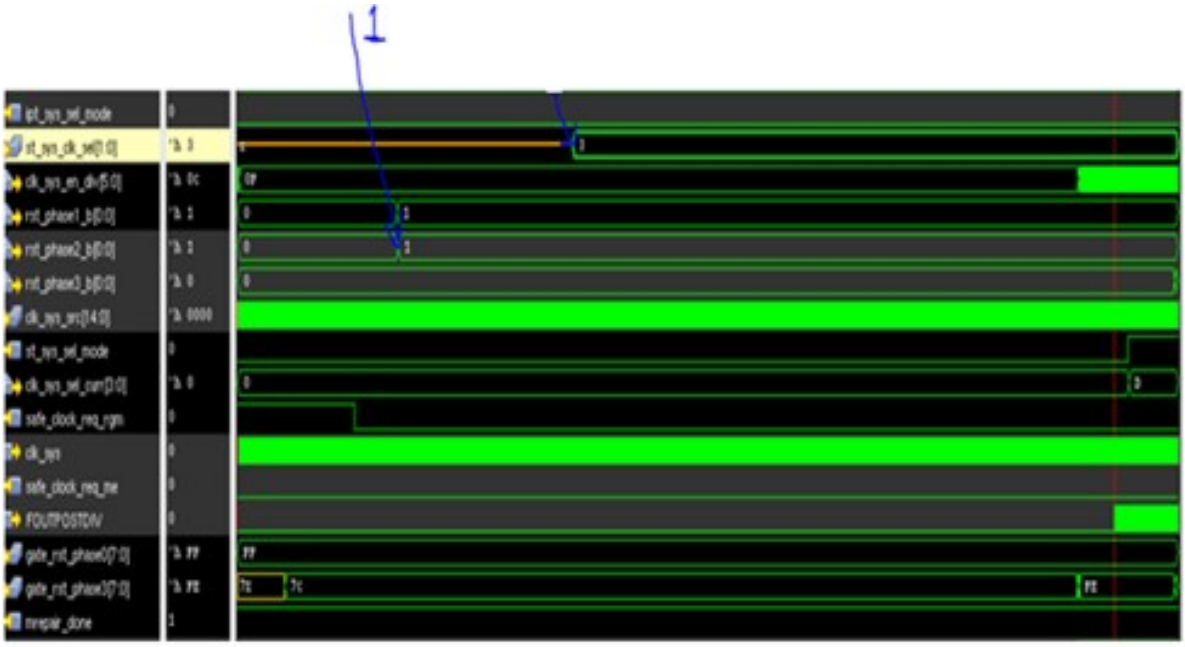

Figure 4-8Reset phase 1 and phase 2 lift

- 1 in the figure shows that the reset phase 1 and 2 has been successfully lifted.
- When the signal sscm_done is asserted, the reset phase 3 has been lifted as shown in *figure 4-9. It* is denoted by numeral 9. This indicates that the subsystem has booted up successfully.

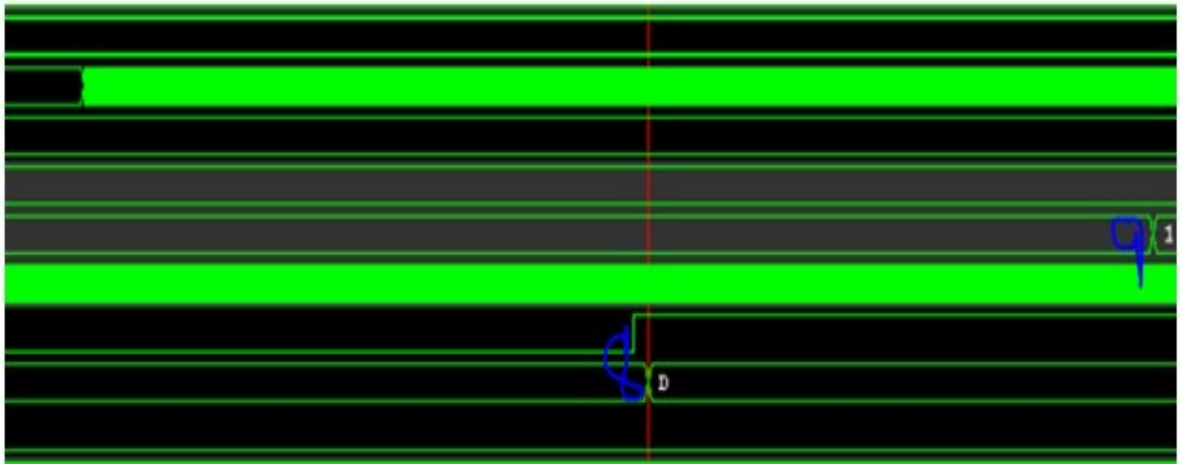

Figure 4-9 Reset phase 3 lift depicted

The results obtained are in accordance with the required design specifications.

Chapter -5

Conclusion and Future Scope

As the designs grows complex, it becomes difficult to achieve the design goals in the required time to market window. The designs are too large to be handled by traditional methods. There is a need for a design flow methodology which can be implemented for and helps in reducing the design time and efforts of the designer. This has achieved by using the IP reuse methodologies and IP-XACT standards in design flow. This has increased the ease of integration of the design. The design is made highly configurable and reusable for a variety of applications. A HSM has been implemented using by this methodology. The optimum results have been achieved by making use of this methodology.

In future extensive automation of design flow can lead to the significant decrease in design time. The IP-XACT standards in future can support highly parameterized structures for Register Banks designs. System RDL can be an alternative for IP-XACT as has more complex feature support than IP-XACT. The highly reusable designs of IPs can support plug and play integration in the designs without using any glue logics in between the designs.

Chapter - 6

References

1. Gordon E. Moore, “Cramming more Components onto Integrated Circuits”, Electronics, April 1965.
2. Haissam El-Aawar, ”Structural and Hardware Complexities Of Microprocessor Design According to Moore’s Law”, International conference on Advanced Computer Science and Information Technology, August 2014
3. “International Techbology Roadmap for Semiconductor”. [Online]. Available: http://www.public.itrs.net
4. M. Keating and P. Bricaud,” Reuse Methodology Manual: For System-on-a-Chip Designs”, 3rd ed. Boston, June 2002.
5. Resve Saleh, Shahriar Mirabbasi, Guy Lemieux and Cristian Grecu,” System-on-Chip: Reuse and Integration”,Proceedings of the IEEE 94, June 2006.
6. Kwanghyun Cho, Jaeboem Kim, Euibong Jung, Sik Kimand Zhenmin Li, "Reusable platform design methodology for SoC integration and verification", International SoC Design Conference, Busan, September 2008.
7. Daniel D. Gajski,V.Chaiyakul, S.Mori, T.Nukiyama, “Essential Issues for IP Reuse”, Proceedings of ASP-DAC, January 2000.
8. “SPIRIT 1.2 Specification”, The SPIRIT Consortium, [Online]. Available www.spiritconsortium.org
9. A.S.Vincentelli and Grant Martin, “Platform-Based Design and Software Design Methodology for Embedded Systems”, Design & Test of Computer IEEE, December 2000.
10. Michael Keating and Pierre Bricaud, “Reuse Methodology Manual for System-on-a-Chip Designs”, Kluwer Academic Publishers, March 2002.
11. Ron Wilson,“SOCs : IP is the new abstraction”, EDN Network, August 2011.
12. “ARM AMBA Specification (Rev 2.0)”, ARM Limited, [Online]. Available: http://www.arm.com
13. S. Sombatsiri, K. Kobashi, K. Sakanushi, Y. Takeuchi and M. Imai, "An AMBA hierarchical shared bus architecture design space exploration method considering pipeline, burst and split transaction," 10th International Conference on Electrical Engineering/Electronics, June 2013

14. Rishabh Singh Kurmi, Shruti Bharwaga, Mahendra Vucha, Sandeep Magarde., " Implementation of an AMBA Advanced High Performance Bus protocol IP block" ,Published in IJECCE, April 2011.
15. F. Demaertelaere, "Hardware Security Modules" , Atos Worldline , May 2015.
16. W. Kruijtzer, Pieter van der Wolf, Erwin de Kock, Jan Stuuyt, "Industrial IP integration flows based on IP-XACT standards," Automation and Test in Europe, Munich, December 2008.
17. "IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows", Sponsor Design Automation Standards Committee of the IEEE Computer Society, June 2014.
18. Clifford E. Cummings," Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog" SNUG , June 2008.
19. Clifford E. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO Design," SNUG, June 2002.
20. Don Mills & Clifford E. Cummings, "RTL Coding Styles That Yield Simulation and Synthesis Mismatches" , SNUG , June 1999.
21. H. Zhaohui et al., "Practical and efficient SOC verification flow by reusing IP testcase and testbench," 2012 International SoC Design Conference (ISOCC), Jeju Island, 2012, pp. 175-178.
22. Universal Verification Methodology(UVM) 1.1 User's Guide , Accellera, 2011.
23. P. Coussy, A. Baganne and E. Martin, "A design methodology for IP integration," IEEE International Symposium on Circuits and Systems, April 2002.
24. Christopher K. Lennard, "Industrially Proving the SPIRIT Consortium Specifications for Design Chain Integration", Proceedings of Design, Automation and Test in Europe, March 2006.
25. Kamil Synek, "Using SPIRIT Cores in Sonics StudioTM", System-on-Chip, International Symposium, November 2006.
26. Vesa Lahtinen, "System Level Design Experiences and the Need for Standardization", System-on-Chip, International Symposium, November 2006.
27. Alistair Bruce and Christopher Lennard, "Maintaining Consistency Between SystemC and RTL System Designs", Design Automation Conference, July 2006.
28. Mattias Carlqvist," System on Chip development based on Magillem 2.3SE" , Europian Space Agency ESTEC, December 2005.
29. H. Ju, Y. Jeon and J. Kim, "A Study on the Hardware-Based Security Solutions for Smart Devices," 2015 International Conference on Computational Science and Computational Intelligence (CSCI), June 2015.

30. S. Zamanzadeh and A. Jahanian, "Scalable security path methodology: A cost-security trade-off to protect FPGA IPs against active and passive tampers," 2017 Asian Hardware Oriented Security and Trust Symposium,July 2017.
31. A. Asaduzzaman, M. F. Mridh and M. N. Uddin, "An inexpensive plug-and-play hardware security module to restore systems from malware attacks," 2013 International Conference on Informatics, Electronics and Vision (ICIEV), August 2013.
32. H. Oh, J. Park, M. Yang, D. Hwang and Y. Paek, "Design of a Generic Security Interface for RISC-V Processors and its Applications," 2018 International SoC Design Conference , December 2018
33. B. Pawankumar, C. R. Bhargava, B. S. Kariyappa, S. Narayanan and R. Kamalakar, "An approach for reusing test source of an IP to reduce verification effort," International Conference on Emerging Technology Trends in Electronics, Communication & Networking" August 2012.
34. Xue-Qiu Dai, Bu-Min Liu, Chun-Yi Xie and Wan-Cai Wang, "A hardware/software co-verification platform for ASIC design," 2009 International Conference on Apperceiving Computing and Intelligence Analysis, July 2009.

APPENDIX

7.1 IPXACT Packaging flow

MAGILLEM IPXACT setup

##

Input Files :

(1) ip_setup.csh -> Template is availabe in magillem area.

Setup the Top Name, Top File Name, Script Path from Release Area and Prapare the envirionment.

(2) tool_data/ipxact/magillem/hdl.lst -> hdl.lst.template Available

- File to Provide List Of Files for FileSet Creation in XML
- Provide Construct to define Library , Package etc.
- Provide syntax differen features that can be added in XML
- Different Types of RTL File Combinations
- XML Files for Subsystem using Other IP
- Provide views specific construct for fileSets

(3) tool_data/ipxact/magillem/tortoise.tcl -> tortoise.tcl.template Available

- Provide VLNV of IP
- Bus Interface and Port Mapping within bus Interface

(4) tool_data/ipxact/magillem/enhance.tcl -> enhance.tcl.template Available

- Over-riding the generated XML with different Properties.
- This File is used very rarely. Most of the Time It is Empty.
- User Can Over-Ride different properties in XML for

ModelParameters and Other Tags.

(5) tool_data/ipxact/magillem/param_template.csv -> param_template.csv.template Available

- Parameter Values for (Min, Max, Choice)

- Add this to make Assmbly Catch the Error Wrong Value or illegal value of parameter used.

Output Files:

(1) tool_data/../ipxact/{1.4,1685-2009}/<ip_name>.xml

The Generated IPXACT for IP for 1.4 and 1685-2009 Standard.

(2) log files

tool_data/ipxact/magillem/*.log

- checkers.log -> Recital and Magillem Checkers Log file. Always CHECK

- incisive_*.log -> Compilation log file. Always CHECK for parameter related warnings, missing ports, port mis-matches, missing files.

- Please Refer to IP IPXACT Design Flow_v3.0.pdf present in installation directory.

##

IP-XACT STEPS

##

STEP 1: Copy the latest ip_setup.csh file into tool_data/ip_setup.csh and Update the following

parameters in the User Inputs section:

rtldir = ../rtl_v; #change this path relative to the location of ip_setup.csh file

topdir = ../rtl_v; #Path of top file used. Change this path relative to the location of ip_setup.csh file. This is useful when top stub is used for IPXACT generation.

toplang = SV; #Language of top module SV,VLG,VHDL

iptop = sta1_ectu_ips; #change this to module/entity name

iptop_file = sta1_ectu_ips.v; #change this to top module file name. Useful when filename is different from module name.

ipxactdir = ../ipxact; #change this path relative to the location of ip_setup.csh file

****

STEP 2: Source the ip_setup.csh file from the same directory.

If <./tool_data/rundir> does not exist then this will create the <./tool_data/rundir>

If <./tool_data/rundir/csv_data> does not exist then this will also create the <./tool_data/rundir/csv_data>

This will step will always copy the common run.csh and clean.csh and .ucdprod files from central area into ./tool_data/rundir

Review all the environment variables for correctness.

****

STEP 3: If its going to be a new version of the IP then follow the procedure to create the new release directory

This step will result in creation of Magillem Data Dir, which is ./tool_data/<reldir>/ipxact/magillem.

Magillem data dir holds the flow input files, flow output files and flow log files.

****

STEP 4: Assembly XML Flow inputs Preparation.

FLOW-1: Migration from Tortoise to Magillem Flow.

All flow input files should be located in Magillem Data Dir i.e. ./tool_data/<reldir>/ipxact/magillem

a> Copy the tortoise packager TCL file usually located in ./tool_data/tortoise_packager/ip_name.tcl. into ./tool_data/<reldir>/ipxact/magillem/tortoise.tcl

b> Create hdl.lst file based on the template and rules.

c> Create enhance.tcl file based on the template and rules.

This file will have commands to improve the IPXACT.

Initially this file can be empty but should be present.

FLOW-2: IF its a new IP then create the files mentioned in a,b and c above.

FLOW-3: Patching of existing XML with filesets and enhancements

a> copy existing XML file to Magillem Data Dir i.e.

./tool_data/<reldir>/ipxact/magillem/<ip_name>.xml

b> Create hdl.lst file based on the template and rules.

c> Create enhance.tcl file based on the template and rules.

This file will have commands to improve the IPXACT.

Initially this file can be empty but should be present.

****

STEP 5: Register XML Flow inputs Preparation.

FLOW-1: Migration from Tortoise to Magillem Flow.

a> Copy the existing tcl file containing register definitions in appropriate cfgN dir

i.e ./tool_data/<reldir>/<cfgN>/ipxact/magillem/<ip_name>_reg.tcl

b> Update the VLN, defined in this file, to match that of assembly VLNV

b> Make sure that the parametrized registers are as per the cfgN parameter definitions.

****

STEP 6: Assembly IPXACT XML Generation

a> goto ./tool_data/rundir

b> do " run.csh <cfgNum><relNum> ipxact "

example: run.csh 0 v01.00.00.00 ipxact

c> Above step will create the Magillem run area in <./tool_data/rundir/ipxact/magillem> and copy the

required scripts there and then launches the IPXACT flow.

d> The flow will generate various versions of Assembly IPXACT XML and validates them.

****

STEP 7: Assembly IPXACT XML Flow review

In this step review the outputs of IPXACT flow.

a> Check all the log files in Magillem Data Dir ./tool_data/<reldir>/ipxact/magillem/*.log

b> Check the XML files for correct VLNV, and model_parameters correctness

d> Check that IPXACT dir ./tool_data/../ipxact/ has the necessary directories and XML files

****

STEP 8: Assembly IPXACT XML Enhancement

If in Step6 you identify issues in model_parameters or identify enhancements then

update ./tool_data/<reldir>/ipxact/magillem/enhance.tcl file and run steps 5 and 6 again

****

STEP 11: Register IPXACT XML Generation

a> goto ./tool_data/rundir

b> do " run.csh <cfgNum><relNum> ipxact_reg "

example: run.csh 0 v01.00.00.00 ipxact_reg

c> This will generate the register IPXACT XML in the specified MAgillem Reg Data directory

i.e ./tool_data/<reldir>/<cfgN>/ipxact/magillem

****

STEP 12: Register IPXACT XML Flow review

In this step review the outputs of IPXACT Register flow.

a> Check all the log files in Magillem Reg Data Dir ./tool_data/<reldir>/<cfgN>/ipxact/magillem/*.log

b> Check the XML files for correct VLNV

****

STEP 12: Cleanup and Release

In this step clean the setup to remove files not required to be checked in the DB

Make a checkin and release

"source clean.csh ipxact" to remove all the temporary files generated by the ipxact tool.

It will remove the Magillem Run Dir directory

(Note: Run clean.csh before making a release, do not clean files while

de-bugging the IP else all the run files will be lost)

7.2 Integration Assembly running

########## **LAUNCHIN MAGILLEM -Setup for magillem** ##############

Step 1 : Sourcing MAGILLEM Assembly Tool and IPXACT BUS Definitions :

-- $source setup_mag_assembly

Step 2 : Edit "run_magillemAssembly.tcl" to change or do project specific argument settings

or to update IP-XACT list as per Project Block list.

Step 3 : Edit "run_script" for

-- to change the Assembly preferences and the Header file path

-- Workspace directory name and log file name if required

Step 4 : ### LAUNCHING MAGILLEM ###

1. GUI mode : $source run_script gui

2. batch mode : $source run_script batch <argument=flat(if flat run) else no argument>

FLAT Run : $source run_script batch flat

HIERARCHICAL Run : $source run_script batch

######### **Wrapper - run_script** ###############

1. As shown in above example it accepts 2 argument

- Mode : GUI or BATCH, parse argument accordingly as shown in Step 5 example above

- RTL Type : FLAT or HIERARCHICAL , no argument for hierarchical run and "flat" as argument for FLAT run

2. In BATCH mode

- the script creates RTL directory outside "run_assembly" directory as per the run type

- it creates log DIR i.e. log_hier or log_flat, and dumps the log with time and date details

so you donot need to clean up log directory or back up previous log files.

- it takes back up of previous RTL automatically with a timestamp.

3. For choosing a different workspace, you can edit the default workspace names in "bsub" commands inside

the script as per project or run requirement

4. Designer can also set the Magillem Specific Preferences for RTL assembly in the starting of the script.

########## MAGILLEM WRAPPER BRIEFED ##############

Here is the brief about Main Assembly script i.e. "run_magillemAssembly.tcl"

Step 1 : To list down IPXACT paths in a variable - l_XML_DIRECTORY_PATH

Step 2 : To Parse various Generic Assembly level and project specific Arguments to Magillem Tool

- Project Name for magillem database
- Settings table path - i.e. Input table path - Settings.csv
- XML List Variable
- BUSDEFINITION PATH
- CHECKER LOG PATH
- RTL Dunp Language - Verilog/VHDL
- O/p Netlist option deep or Top level
- IPXACT Version used - 1.4 or 1685-2009 - mixed version not supported
- Severity Settings

Step 3 : sub-scripts invoked within the run_magillemAssembly.tcl

-- hierclean.tcl

-- replaceRevertConnections.tcl

-- driver_based_naming.tcl

-- gen_reg_prot_param.tcl -- Register protection Parameter generator

-- magillemAssembly.tcl

DESIGN_FLOW_OF_SUBSYSTEM_DESIGN_HSM___piyush_...

ORIGINALITY REPORT

5%	3%	2%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	documents.mx Internet Source	1%
2	Submitted to Texas A & M University, Kingville Student Paper	<1%
3	Submitted to CSU, San Jose State University Student Paper	<1%
4	www.date-conference.com Internet Source	<1%
5	www.ijmetmr.com Internet Source	<1%
6	Submitted to CSU, San Jose State University Student Paper	<1%
7	ir.nctu.edu.tw Internet Source	<1%
8	Submitted to Higher Education Commission Pakistan Student Paper	<1%
9	theses.ju.edu.jo	

	Internet Source	<1%
10	Sridhar Gangadharan, Sanjay Churiwala. "Constraining Designs for Synthesis and Timing Analysis", Springer Nature, 2013 Publication	<1%
11	Submitted to University of Southampton Student Paper	<1%
12	www.ijfset.org Internet Source	<1%
13	www.ijsres.com Internet Source	<1%
14	Ochoa-Ruiz, Gilberto, Ouassila Labbani-Narsis, El-Bay Bourennane, and Phillipe Soulard. "Model-Driven Approach for Automatic Dynamic Partially Reconfigurable IP Customization", 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, 2012. Publication	<1%
15	Graham Jullien, Magdy Bayoumi. "Special Issue on Systems-on-Chip: Design and Integration", Proceedings of the IEEE, 2006 Publication	<1%
16	Submitted to UC, Boulder Student Paper	<1%

17	Naghmeh Karimi, Krishnendu Chakrabarty. "Detection, Diagnosis, and Recovery From Clock-Domain Crossing Failures in Multiclock SoCs", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2013 Publication	<1%
18	D. Bertozzi, L. Benini. "Chapter 1 Hardware Platforms for Third-Generation Mobile Terminals", Springer Science and Business Media LLC, 2008 Publication	<1%
19	Submitted to Universiti Malaysia Perlis Student Paper	<1%
20	Submitted to Queen's University of Belfast Student Paper	<1%
21	www.armjiemi.com Internet Source	<1%
22	Submitted to Manipal University Student Paper	<1%
23	Submitted to Universiti Teknologi Malaysia Student Paper	<1%
24	Chaturvedi, S.. "Static analysis of asynchronous clock domain crossings", 2011 Design Automation & Test in Europe, 2012. Publication	<1%

25	Himanshu Bhathagar. "Advanced ASIC Chip Synthesis", Springer Nature, 1999 Publication	<1%
26	Submitted to Monash University Sunway Campus Malaysia Sdn Bhd Student Paper	<1%
27	Submitted to Royal Holloway and Bedford New College Student Paper	<1%
28	alpha400.ee.unsw.edu.au Internet Source	<1%
29	Submitted to California State University, Sacramento Student Paper	<1%
30	Jung-Han Lee, Dong-Wook Kim, Won-Il Cho. "Reduced IEEE 1284 soft IP design for system IC implementation", AP-ASIC'99. First IEEE Asia Pacific Conference on ASICs (Cat. No.99EX360), 1999 Publication	<1%
31	Submitted to Cranfield University Student Paper	<1%
32	doras.dcu.ie Internet Source	<1%
33	Submitted to Rajarambapu Institute of	<1%

Technology
Student Paper

34	Katalin Popovici, Frédéric Rousseau, Ahmed A. Jerraya, Marilyn Wolf. "Embedded Software Design and Programming of Multiprocessor System-on-Chip", Springer Nature, 2010 Publication	<1%
35	Submitted to University of Surrey Student Paper	<1%

Exclude quotes	On	Exclude matches	< 8 words
Exclude bibliography	On		