BUS CONTROLLED AMBA 2.0 AHB2APB BRIDGE FOR SOC APPLICATION

A DISSERTATION
SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

IN

VLSI DESIGN AND EMBEDDED SYSTEMS

Submitted by:
PULKIT MITTAL
2K20/VLS/14

Under the supervision of
DR. SONAM REWARI

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042
MAY 2022

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I PULKIT MITTAL student of MTech (VLSI Design and Embedded Systems), hereby declare that the project Dissertation titled "BUS CONTROLLED AMBA 2.0 AHB2APB BRIDGE FOR SOC APPLICATION" which is submitted by me to the Department of Electronics and Communication Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi **PULKIT MITTAL**

Date: May 16, 2022

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled "BUS CONTROLLED AMBA 2.0 AHB2APB BRIDGE FOR SOC APPLICATION" which is submitted by **PULKIT MITTAL**, **2K20/VLS/14** of Electronics and Communication Department, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi **DR. SONAM REWARI**

Date: May 16, 2022 SUPERVISOR

ABSTRACT

The AMBA (Advanced Microcontroller Bus Architecture) is an open SOC (System-on-Chip) bus protocol to strengthen the reusability of IP core, for high-performance buses to communicate with low-power devices by communication through the connection of different functional blocks (or IP), and using multiple controllers and peripherals, we can develop multiprocessor unit. This Research paper explains the implementation of AHB to APB Bridge. The bridge provides a communication interface between AMBA AHB v2.0 Masters and APB v2.0 slaves and parameterized data bus for AHB master and APB slaves. It supports transfers even when AHB transfer size and APB data bus width is not equal and AHB and APB interfaces work in separate independent clock domains. It has inbuilt cross-domain synchronization with parameterized number of synchronization stages. The design supports multi master and multi slave configuration. To perform functional and timing simulation, we are using System Verilog on Xilinx VIVADO Tool.

ACKNOWLEDGEMENT

A successful project can never be prepared by the efforts of the person to whom the project is assigned, but it also demands the help and guardianship of people who helped in completion of the project.

I would like to thank all those people who have helped me in this research and inspired me during my study.

With profound sense of gratitude, I thank Dr. Sonam Rewari, my Research Supervisor, for her encouragement, support, patience, and her guidance in this research work.

I owe my deepest thanks to my family, who always stood by me and guided me through my career and have pulled me through against impossible odds at times. Words cannot express the gratitude I owe them.

PULKIT MITTAL

Table Of Content

List of Tables

List of Figures

Chapter 1: INTRODUCTION

1.1 Overview

AMBA which abbreviates to Advanced Microcontroller Bus Architecture protocol was basically introduced by ARM in the year 1996. It was mainly used to connect several blocks of SOC or ASIC to build a complex circuitry. Using AMBA built of multiprocessor and microprocessor designs with large number of peripherals and the controllers has become possible. The major use of AMBA can be seen in complex SOC designs which includes the SOC used in your mobile phone or laptop. The functioning of SOC depends not only on the building blocks but also on the interconnection of these blocks and to make this functioning possible AMBA protocol was introduced. The initial release i.e., the first version had two buses namely the APB (Advanced Peripheral Bus) and the ASB (Advanced System Bus).

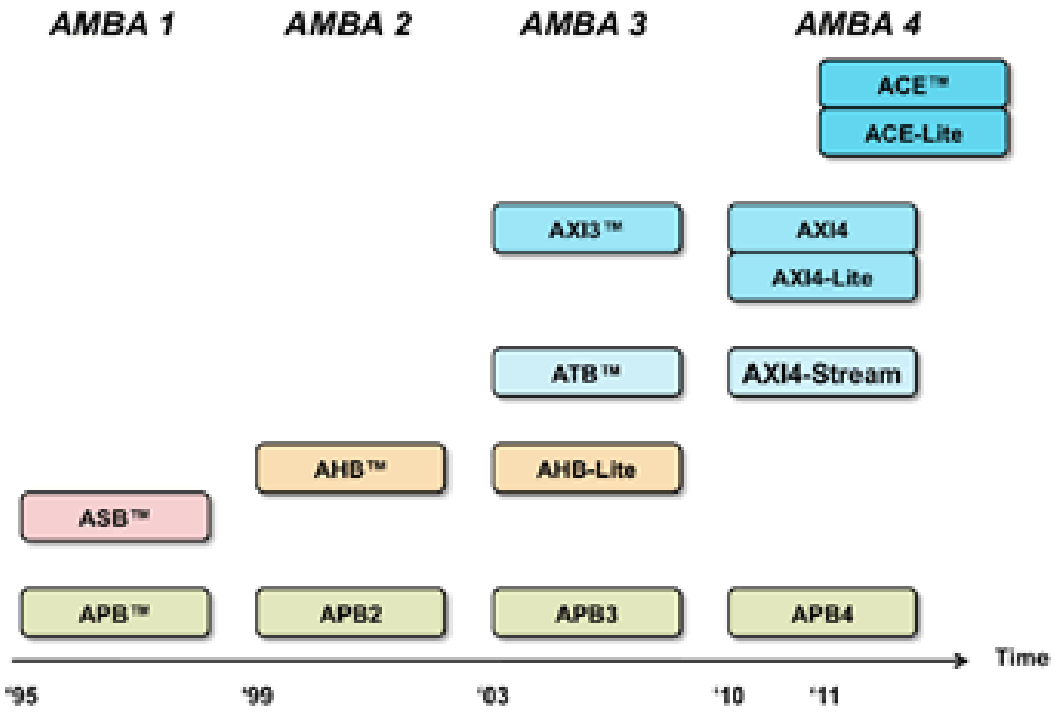


Figure 1.1. Evolution of AMBA Standards

An addition of AHB (AMBA High Performance Bus) was made in its second version in the year 1999. This protocol works on a single clock edge and has a wide application on ARM 7 and ARM 9 designs and even today it is used on ARM Cortex-M based designs. AMBA 2 protocol made communication between high speed and low speed peripherals possible and till date it is widely used. Later in the year 2003, 3rd Generation of the AMBA protocol i.e., AMBA 3 was introduced. It included AXI (Advanced Extensible Interface). In year

2010 AMBA 4 was introduced which included AXI4 and then in year 2011 ACE (AXI Coherency Extensions). And at last in the year 2013 AMBA included CHI (Coherent Hub Interface) it was designed for scalability so that performance can be maintained as the number of components and traffic grows. This includes forcing masters to respond to coherent snoop transactions, implying that forward mobility for specified masters can be more easily guaranteed in a busy system.

Advanced System Bus	ASB	Now obsolete, so don't worry about this one!
Advanced Peripheral Bus	APB	Simple, easy, for your peripherals
Advanced High-Performance Bus	AHB	Now used a lot in Cortex-M designs
Advanced eXtensible Interface	AXI	The most widespread, now up to AXI4
Advanced Trace Bus	ATB	For moving trace data around the chip, see CoreSight
AXI Coherency Extensions	ACE	Used in big.LITTLE systems for smartphones, tablets, etc.
Coherent Hub Interface	CHI	The highest performance, used in networks and servers

Table 1.1. Abbreviations used in AMBA protocol.

AMBA (Advanced Microcontroller Bus Architecture) is an open SOC (System-on-Chip) bus protocol that allows high-performance buses to communicate with low-power devices by connecting different functional blocks (or IP), allowing us to create multiprocessor units with multiple controllers and peripherals. This research paper explains how the AHB to APB Bridge was implemented. The bridge connects AMBA AHB v2.0 Masters and APB v2.0 Slaves, as well as providing AHB masters and APB slaves with a parameterized data bus. It facilitates transfers even when the AHB transfer size and APB data bus width are not the same and the AHB and APB interfaces are in different clock domains. It offers built-in cross-domain synchronization with a number of phases that may be customized. The architecture allows for several master and slave configurations. We use System Verilog on the Xilinx VIVADO Tool to do functional and timing simulation.

The major objectives of AMBA are to permit right-first-time development, to be innovation-free, to energize a measured framework outline, and to effectively lower the silicon foundation. AMBA forms can help with configuration reuse as well as changing framework execution by allowing for swift transit and high transfer speeds.

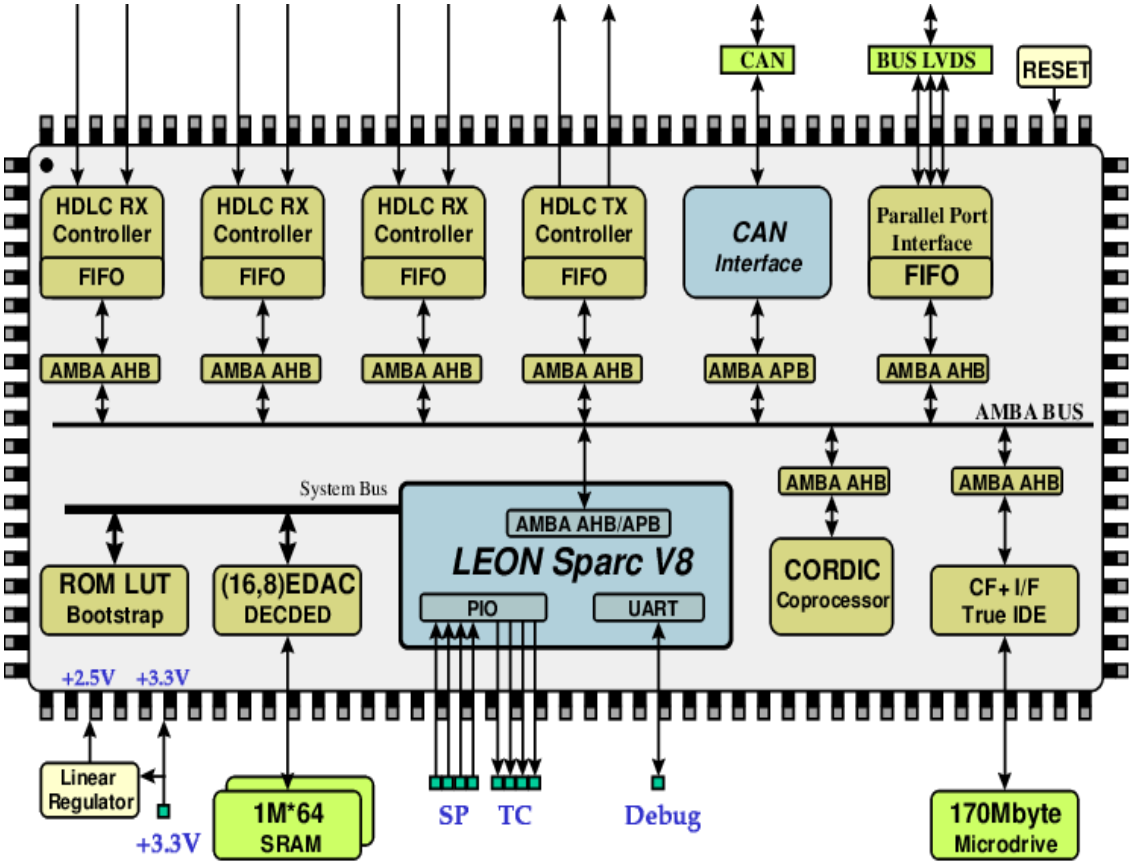


Figure 1.2. SOC System Block Diagram

1.2 Benefits of AMBA:

- **Compatibility**- A standard connection point detail, as AMBA, permits similarity between IP parts from various plan groups or merchants.
- **Efficient IP reuse**- If an IP is compatible to be reused, then it directly reduces the SoC design cost and the time to market, it happens because AMBA protocol provides such specifications interface standards.
- **Support**- AMBA is all around upheld. It is broadly carried out and upheld all through the semiconductor business, including support from outsider IP items and apparatuses. Bus interface principles like AMBA, are separated through the high-speed performance that they empower.
- **Bandwidth**- Bandwidth is defined as the maximum speed at which the data can be communicated through the interfaces so for synchronous digital system the bandwidth is basically the product of the width of the data bus and the clock speed
- **Latency**- It is defined as the time elapsed between the initiation and completion of the transaction. So, for a burst-based system latency is basically the time taken to complete the first transaction and not the entire transfer. An interface is typically

regarded as efficient if it acquires the maximum bandwidth in addition with almost zero latency

- **Flexibility-** AMBA offers the versatility to work with an extent of SoCs. IP reuse requires an extra ordinary standard while supporting a wide collection of SoCs with different power, execution, and district essentials. ARM offers an extent of association point specifics that are updated for these different necessities.

1.3 Different variety of AMBA-

APB

Advanced Peripheral Bus is used in AMBA-based designs to reduce the complexity of interface and power consumption. APB is used to connect peripherals with low transfer speeds. Framework execution can be improved with APB. The two main elements of this transportation are the APB Bridge and the APB Slave. The vehicle specialist is APB Bridge. Only for single transit, the APB in AMBA-based outlines will have a master. Address hooking, creating a strobe flag PENABLE and a select flag SSELx, putting information on APB for compose exchange, and making APB information available for perused exchange are the main features of APB Bridge. APB Bridge identifies a number of planning factors.

Information and yield parameters are the names of these parameters. The APB slave's interface is touted to be extremely customizable. The APB slave work is controlled by the APB timing parameters. SCLK, SRESETn, SADDR [31:0], SSELx, SENABLE, SWRITE, SRDATA, and SWDATA are the essential flags that govern APB's work. APB3 v1.0 is an APB subset. SREADY and SLVERR, as well as other APB indications, are used in these two extra signals. The most current upgraded adaption of APB is APB4 v2.0. These signals are used to write and read data.

ASB

Advanced System Bus can be used as part of the numerous inserted micro controllers as an elite pipelined transport. It helps to connect several CPUs, external memory ports, and on-chip memories. ASB provides fundamental features such as burst exchange, varied transport master assistance, and superior pipelined activity.

ASB master, ASB slave, ASB Judge, and ASB decoder are all guideline portions of ASB.

Through address and control information, the ASB master initiates the read exercises. Various modes of transportation are available in ASB, but only one will be used. ASB Judge will assist the pro in gaining admission to ASB. The normal ASB slave limit is to reply to scrutinised requests and generate tasks.

The ASB decoder is used to untangle the convey and choose the best possible slaves. The three types of exchanges that can happen through ASB are non-sequential, sequential, and address-only. DSELx, BWRITE, BWAIT, BTRAN [1:0], BPROT [1:0], BSIZE [1:0], BnRES, BLOK, BLAST, BERROR, BD [31:0], BCLK, BA[31:0], AREQx, and AGNTx are the characteristic signals used in this transport.

AHB

Advanced High-Performance Bus is widely employed as an elite mode of transport that may improve data transmission activity. Split exchanges, enhanced information transit setup, burst exchange, single clock edge activity, and so on may all be accomplished with AHB. AHB is supported by ARM7, ARM Cortex-M, and ARM9 processors. AHB master, AHB slave, AHB decoder, and AHB authority are all part of the AHB framework setup.

AHB uses address and control to read and compose tasks. Only one master at a time can make effective use of the conveyance. The slave of AHB reacts to the master of AHB. Slave reacts to the read or compose operation with the help of the address. The slave to the master recognises the status of the information exchange. The status indicates if the information exchange was a success, failure, or halt. AHB arbiter and decoder have the same capabilities as ASB.

RWDATA, RSELx, RRDATA, RREADY, RRESP[1:0], RSPLITx [15:0], RMASTLOCK, RMASTER [3:0], RGRANTx, RLOCKx, and BUSREQx are all AHB signals. AHB-Lite v1.0 has a high-speed exchange motion. In addition to the fundamental AHB signals, this modification employs a variety of banners for improved action.

AXI

ARM introduced Advanced Extensible Interface v1.0, a burst-based convention, in the third period of AMBA. It provides superior task performance, high recurrence, and speed.

There is a distinct address and stage of information. It is used to exchange data between byte strobes. With address issue, burst exchanges are possible. The distinctive classifications of signs introduced in AXI are compose information carrier signals, compose address carrier signals, compose reaction carrier signals, read address carrier signals, read information carrier flags, and low power interface signals. There are five unique channels accessible for reading and writing. AXI's other main features include request exchange fulfilment and the increase of enlistment phases. AXI4 – Lite is a more advanced version of AXI. It alters the signs of the essential AXI. This subset utilizes a settled information transport width and backings compose strobes. AXI-Stream v1.0 is the most recent adaptation of AXI.

ACE

AXI Coherency Extensions is an AXI upgrade that includes third-level reserves, on-chip RAM, peripherals, and external memory. The AXI read and compose channels can be created for a 64-bit or 128-piece interface here. In terms of the CPU clock, it supports 1:1 clock proportions. It can also operate multiple CPU clocks. On ARM Cortex-A processors such as the Cortex-A7 and Cortex-A15, Pro is used. Interconnect, ACE masters, ACE – Lite masters, and ACE Lite/AXI slaves are the various portions of master. The framework level coherency is given structure by experts. The genuine flags of ACE are read data channel signals, read address carrier signals, snoop carrier signals, write address carrier signals, and response signals.

ACE is a subset of Pro Lite. Ace components that don't have equipment intelligible reserves use Pro Lite. They can show whether the issued exchanges may be held in distinct bosses' equipment coherent stores or whether they can assist in obstructing exchanges. It includes additional flags on the read and compose address channels. Snoop channels, snoop indicators, and response signals are not included in Pro Lite.

ATB

The Advanced Trace Bus promotes information sharing inside the Core Sight troubleshooting framework. It supports bite-sized bundles and the control signals for displaying the number of bytes significant in each cycle. The indications used by this transport for activity include ATCLK, ATCLKEN, ATRESETn, ATVALID, ATREADY,

ATID [6:0], ATBYTES [m: 0], ATDATA [n: 0], AFVALID, and AFREADY.

Designs based on AMBA

AMBA is used by SDRAM and Flash memory controllers (DMC-34x), Network Interconnect (NIC-301), store controllers (L2C-310), and DMA controllers (DMA-230), among others. AMBA is also used by non-ARM plans.

1.4 Goals

1. It will configure less potential interfaces.
2. It will overhaul the reusability of periphery and IPs and simplify plan
3. It will redesign the embedded limited scope regulator things with central taking care of unit on SOC.

1.5 Thesis Organization

The following is the format of the thesis report:

The establishment of this idea is summarized in Chapter 1. It includes a brief overview of the Advanced Microcontroller Bus Architecture (AMBA), advanced peripheral bus (APB), advanced system bus (ASB), and advanced high-performance bus (AHB). Encourage it has discussed the AMBA assurance objectives.

The Advanced Microcontroller Bus Architecture (AMBA) advanced peripheral bus (APB) configurations that have existed up to this point are discussed in Chapter 2. It also highlights the shortcomings of the present structures.

The APB is reviewed in Chapter 3, which serves as the foundation for the suggested storyline. It covers the APB connecting segments and flags' places of interest.

The experimental setup and simulation results are presented in Chapter 4.

Chapter 5 summarizes the findings of our research and suggests possible directions for additional investigation.

Chapter 2- LITERATURE SURVEY

ARM introduces AMBA in the year 1999, which is an open-source interface tradition usually characterized as a Bus Protocol. The broad survey is done on Bleeding edge Microcontroller Transport Building and outline of AMBA interface module.

The creator introduced an AMBA 2.0 that recognizes three methods of transportation: ASB, APB, AHB. It researches all methods of transportation's trying frameworks. The connection point module is a piece of programming that permits you to as far as its limited state machine, AMBA AHB may obviously be used as an ordinary joining module since it can peruse and compose information. As it speaks with the fringe contraptions, the association module AMBA APB is melodic, showing a low rehash.

Kiran Rawat proposed a multifaceted connection point between AMBA ASB as well as APB is the objective of the mix. Verilog HDL with limited state machine functioning planned in Model Sim Variant 10.3 and Xilinx-ISE frame suite were utilized to plan the usage overview and power reports. The utilization of APB Extension requires the work of an official and a decoder. In the AMBA ASB and the APB module, the expert connects with APB protocol. The appointed authority starts speaking with the vehicle subsequent to deciding on the expert's status. The decoder chooses a vehicle slave utilizing the particular location lines, and the slave replies to the vehicle ace with an assertion. As the layout intricacy of the designs rises, the power utilization of SoC structures turns out to be progressively fundamental. The power reports separate the different power parts that add to drive use.

Kiran Rawat et al. presented AMBA APB, a variation of AMBA that gives the most reduced power utilization and bandwidth. For this, a Verilog HDL APB Extension with Reset Controller arrangement was utilized. BnRES and Power-on Reset (PO Reset) conditions are introduced by the reset regulator so Meta stable attributes might be spread and errors can be maintained at an essential separation. Power report exhibits that the different power sections contribute to the power use by APB associate plan. As the result, augmentation under PO Reset conditions, On-chip amount to control use is 9.52%, Chain of order control use is 29.12% and dynamic stock control usage is 28.89% not however much Scaffold under no PO Reset conditions. Accordingly, when Scaffold is planned under Power-on Reset conditions, it might take full advantage of force. This is finished

utilizing the Verilog vernacular, which is utilized to make limited state machine models and test seats. Model Sim Rendition 10.3 is utilized to show and repeat the APB Extension and Reset Controller. For mix and power announcing, the Xilinx-ISE frame suite, variant 13.4, is utilized.

Kiran Rawat et al. proposed the focal test for building a game plan to outline as well as to facilitated and mix RTL code to plan importance and streamlined in control utilization. The's maker will probably make the AMBA APB expansion a reality by fittingly figuring out framework assets. For this, a replication and synthetization of the reach point of association, not entirely set in stone to achieve the least power use and data move limit possible between AMBA fast ASB and low speed APB transports. Right when a capability is made between the passage timings of clock flags, clock incline appears. Limit clock inclination can be overseen either a swell counter or a three-piece up or down counter method. The article uses Verilog HDL to complete an APB Scaffold with a clock incline decline methodology.

Jasmine Chhikara et al. proposed the unit which includes more modest valuable squares called subsystems or module. These modules ought to be in a condition of amicability with one another and give resources for the construction to work fittingly. The issue arises when one subsystem takes on comparative guidelines as others. Each module uses another piece rate or baud rate for data exchange, which might be non-synchronous or composed. The maker moreover advises the most effective way to give information by starting to one show and thereafter progressing forward to the accompanying. It takes advantage of I2C's adaptable shows, making it ideal for use with the APB AMBA show. The proposed planning is a framework between the I2C Master and the APB ointment, allowing information to move from an I2C-enabled module to an APB-engaged module. The data is moved in a state of congruity with the area clock, from successive (I2C) to look like (APB) to consecutive (I2C). This makes a bidirectional association point between I2C-enabled and APB-engaged modules.

Ashutosh Gupta gave an on-chip depiction foundation for better implanted microcontrollers. This figure depicts the genuine execution of the AMBA advanced structure transport (ASB) and undeniable level periphery transport (APB) affiliation modules. To stay aware of the clock inclination to a base, a three-piece extend counter was utilized. For showing up and reenactment, Model Sim Form 10.3 is utilized, and test

seats are made thus. For the blend and experience, the RTL Compiler is used, while the Xilinx-ISE plan suite is used to discard the affiliation and use rundown. A refined execution framework is used for the genuine course of action.

Kanishka Lahiri and Anand Raghunathan et al. proposed the confounded System on-chips (SoCs), the design level on-chip correspondence setup is extending as a monster wellspring of force use. The board and redoing are the vital segments of SoC control which requires the attributes of the cutoff use. While persuading, they essentially address a constrained piece of correspondence setup control use. A cutting edge correspondence planning, incorporates very few parts, for example, transport interfaces, center individuals, stages, decoders, and multiplexers, in spite of the general vehicle lines as well as quantifiably supporting the perspective that on-chip correspondence is a fundamental focus for framework level power streamlining, their work shows (I) meaning of totally considering correspondence planning, and (ii) the entryways for control decline that exist through mindful correspondence planning plan

Ge Zhiwei et al. inspected a sharp picture on-chip and CMOS sensor, which is applied to the APB transport. The suggested plan shows hiding picture organizing difficulties and highlights the separations between the proposed structure and the normal picture, which considers pipeline a piece of cutting edge still cameras. This work joins two great vehicles white adjust techniques to change the three self-regulating hiding channels, thinking about the stuff use and power requirements. The suggested development, which is associated with FPGA, can truly restore the picture thought of brutal data.

L. Benini, A. Macii et al. proposed the encoding and deciphering counts that will impart the way to deal with imagining that will confine the common number of changes on enthusiastically stacked for the most part transport lines at no cost in correspondence throughput (i.e., single word is sent at each cycle). Given data on word-level measures, the seeing part methodology grows low-change improvement codes and stuff execution of encoders and decoders without relying upon organizer's sense. A right circumstance that is sensible to low-width transports, what's more finished up frameworks that scale well with transport width. Also, show a versatile construction that commonly changes encoding to diminish advance enhancement for transports whose word-level evaluations are not known from the before.

J. Y. Chen et al. proposed a strategy that competently reduces the exchanged capacitance

of the vehicle. The power ate up by the vehicle can, in this way, diminished. The vital of the vehicle division is to apportion transport into several vehicle sections withdrew by pass semiconductors. Especially transmission contraptions are organized to adjoin transport pieces, thusly, most information Correspondence can be accomplished by exchanging a little part of the vehicle fragments. As necessary, control use and delay are both decreased. Exploratory outcomes got by emulating a yield show and a power show that the proposed divided transport structure reduces transport control by around 60%-70% and overhauls central vehicle delay by around 10%-30%.

Late movements in SoC advancement have enabled the blend of various devices on a singular chip, making room for extra limited contraptions. The various gadgets from various vendors, each filling a particular job, are completely combined on a lone chip. These parts can be associated using transport-based shows to enable strong correspondence. The AHB show is the most comprehensively utilized particular technique. The AHB show grants devices to give at fast speeds. Because of their unwieldiness, processors are consistently growing quicker, but memory are ending up being all the more sluggish. This transforms into an issue since the data recuperation speed doesn't match the dealing with speed. Likewise, a fast memory controller is fundamental, one that can match the CPU speed to the memory speed to ensure strong correspondence. It’s trying to associate SDRAM with AHB since SDRAM's dormancy isn't confined to one cycle, obliging AHB to sit inert all through that period. In this way, transport resources are being used inefficiently.

K. Shaikh and accomplices made a SDRAM-express memory unit. Right when an inquiry is made worried actually consumed data, for example, the controller inside memory is glanced through first preceding getting it to the memory. Since the AHB transport designing obliges extended execution, extended clock repeat structure modules, this controller was expected to work with the AHB (AMBA) transport plan. It's the construction block for extended execution systems.

The coming AMBA transport hardware IP presented by Acasandrei et al. is a modularized, entirely versatile, diminished power, and development free focus written in the HDL language. The Viola-Jones methodology, which is maybe the most frequently used face ID technique, is accelerated by the IP place. The gear gas pedal IP is used in an embedded face acknowledgment structure considering the LEON3 Sparc V8 CPU. The

makers portray their strategies, challenges, and execution revelations for programming, gear, and structure level arrangement.

Present day SoCs integrate multi-focus bundles and refined peripherals, for which the continuous AHB show, which supports low-complexity shared transports, can't compare the suppositions of the current world high speed SoCs, due to different show obstacles, including the way that the AHB has recently a solitary uncommon trade, no full-duplex mode, and simply a solitary channel-shared transport that is facilitated to the agent AXI transport. By recognizing how the five-redirects in the vehicle work independently, as well as the handshaking thought, a more reasonable assessment with the significant level AXI transport is drawn.

The AMBA AXI4 transport, which is the most unimaginable concerning throughput, lethargy, and world class execution/repeat, is used with single or various channels, and the affiliation block epitomizes the power, decoder, and multiplexers. The official screens the need to get to/release the vehicle to one of various managers who start trades all the while using an intercession technique. The decoder disentangles the master's area and control and passes the trade on to one of 16 slaves for major and burst read/form exercises.

The arrangement complexity of SOC is extending bit by bit as the result it extended customer demands. Accordingly, there is for the most part a productivity opening, and the reasonable show is picked for each application. To additionally foster organization execution, QoS, and reduce wiring blockage, a migration from AXI4 to AXI4-Lite is required, which allows the processor or supervisors to get to the registers (pretty much nothing and more modest than regular peripherals). We pick AXI4-Lite. Using the recently referenced references and keeping an eye out for the situation, assuming the Master expert regularly holds onto any longing to get to information that may be in little registers for which it uses fundamental registers like store to impart, a reasonable and authentic vehicle point of interaction ought to be chosen to resolve the issue. This actuated the headway of the Verilog HDL work "Plan and place of connection of AXI4-

Chapter 3: ARCHITECTURE AND DESIGN OF AMBA 2.0

In embedded systems, the designer faces a lot of issues, to integrate different designs and meet certain requirements, such as low power consumption, pipelining of data, operating low bandwidth peripherals, high performance devices, etc. on a single system of chips. To solve this problem and meet all the design requirements of such a system with one more CPU or single processor an AHB to APB bridge is implemented.

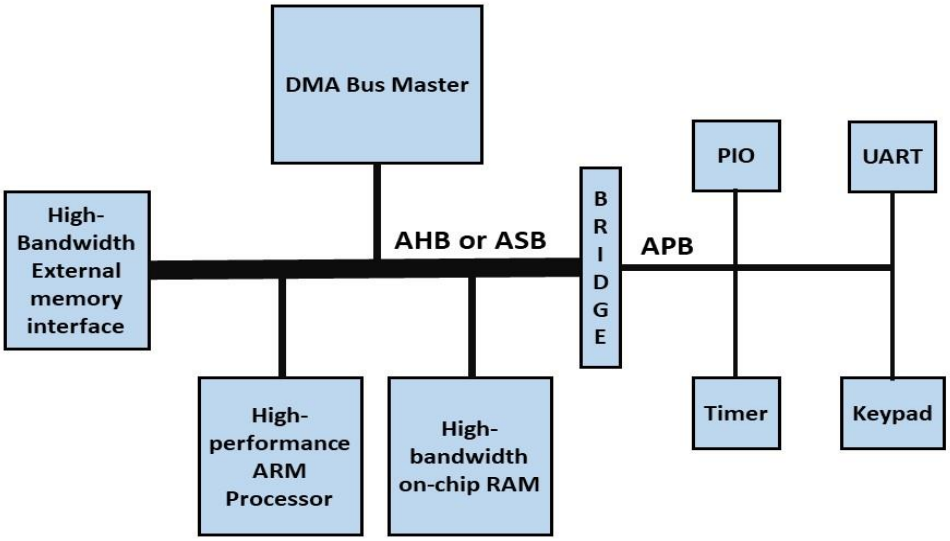


Figure 3.1. AMBA based AHB to APB or ASB to APB bridge

The AHB to APB bridge functions as an AHB slave, connecting the high-speed AHB and the low-power APB. The AHB's read and write transfers are transformed to the APB's equivalents. The connection of high performance, high speed and high bandwidth peripherals such as on-chip RAM, DMA etc. with their counterparts such UART, timer, keypad is established with the help of AHB2APB bridge

This section outlines the hardware architecture design of the IP. The AHB-APB Bridge is a parameterized IP that acts as an interface between the AHB and APB bus protocol. Multiple APB peripherals may be accessed through the AHB-APB bridge.

3.1 BLOCK DIAGRAM

The AHB Bridge will comprise of the following components:

- AHB Slave Interface
- AHB to APB Translation block
- APB Master interface
- AHB and APB Synchronization blocks

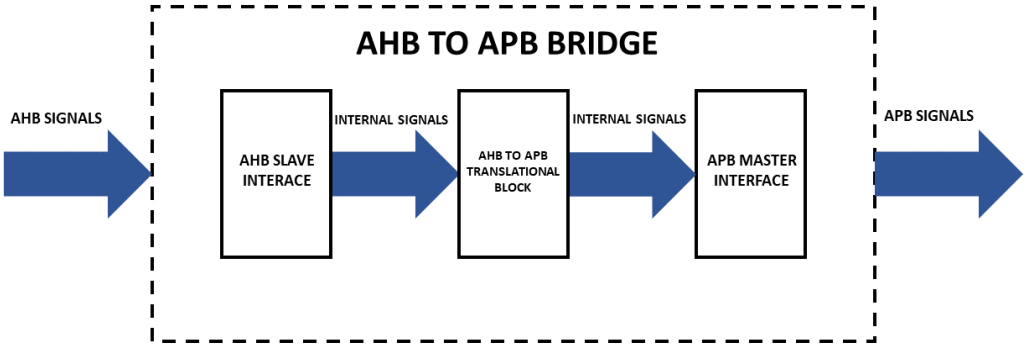


Figure 3.2. AHB2APB Bridge Block Diagram

3.2 Constraints and assumptions made during implementation of the above design

ASSUMPTIONS

- AHB Master input signals are compliant with the AMBA AHB v2.0 protocol.
- APB Slave input signals are in accordance with the AMBA APB v2.0 protocol.
- De-assertion of AHB Reset, hresetn_i is synchronized w.r.t to AHB clock.
- De-assertion of APB Reset, hresetn_i is synchronized w.r.t to APB clock.

CONSTRAINTS

- AHB Clock frequency should be greater than or equal to APB clock frequency.
- AHB HSIZE should be such that the transfer size does not exceed the AHB Data bus width (AHB_DATAWIDTH).
- AHB Data bus width should be greater than or equal to APB data bus (AHB_DATAWIDTH >= APB_DATAWIDTH)
- AHB and APB data bus should be integral multiples of byte (AHB_DATAWIDTH mod 8 = 0 and APB_DATAWIDTH mod 8 = 0).
- The minimum number of synchronization stages (SYNC_STAGES) should be 2.

Taking these constraints and assumptions as the input collaterals for the implementation of AHB2APB bridge has made the design more realistic and efficient so that it can be used for high-speed data transfer without any distortion. The burst transfers, made during the transaction between the high-speed APB and AHB has now become possible for read and write operations.

Parameter	Default value
AHB_DATAWIDTH	32
APB_DATAWIDTH	32
NUM_OF_APB_SLAVES	10
PERIPHERAL_START_ADDR	32'h80000
PERIPHERAL_END_ADDR	32'h8CFFF
SYNC_STAGES	2

Table 3.1. List of Parameters

The data width of AHB is 32 bits as well as the data width of APB is 32 bits. The number of APB slaves supported through this bridge are 10 and it has 2 sync stages. The hexadecimal values of Peripheral start address, and end address are listed in Table 3.1.

There are several pins in the design of the AHB2APB bridge.

The output pins include hready_o which has a width of 1 bit. It basically indicates the completion of present transfer and can also be used to insert wait cycles between the transaction, hresp_o has width of 1 bit and it gives the transfer response which provides information on the status of the transfer, hrdata_o has a width of 32 bit and it reads the data from the slave, psel_o is a pin which is used to select the slaves among the APB, penable_o has a datawidth of 1 bit and it Indicates the second and subsequent cycles of the APB transfer, paddr_o is a 32 bit Address bus which tells the address of the bits, pwrite_o is single bit pin and it Indicates direction of transfer. When HIGH indicates write access and when LOW indicates read access, pwdata_o has same with as the APB_DATAWIDTH and is a APB Write data bus, pprot_o is a 3 bit data bit which is for

APB Protection, pstrb_o has a datawidth of APB_DATAWIDTH/8 and it Indicates the active byte lane during a write transfer.

Pin Name	Direction	Width	Description
hready_o	Output	1	Indicates completion of present transfer and can alsos be used to insert wait cycles
hresp_o	Output	1	The transfer response which provides information on the status of the transfer
hrdata_o	Output	AHB_DATAWIDTH	The data read from the slave
psel_o	Output	y*1	APB Slave select
penable_o	Output	1	Indicates the second and subsequent cycles of the APB transfer
paddr_o	Output	32	APB Address bus
pwrite_o	Output	1	Indicates direction of transfer. When HIGH indicates write access and when LOW indicates read access.
pwdata_o	Output	APB_DATAWIDTH	APB Write data bus
pprot_o	Output	3	APB AProtection type
pstrb_o	Output	APB_DATAWIDTH/8	Indicates the active byte lane during a write transfer
hsel_i	Input	1	AHB Select line
haddr_i	Input	32	AHB Address bus
htrans_i	Input	2	Indicates the transfer type of the current transfer
hsize_i	Input	3	Indicates size of AHB transfer
hburst_i	Input	3	Indicates the burst type
hwrite_i	Input	1	Indicates direction of transfer. When HIGH indicates write access and when LOW indicates read access.
hwdata_i	Input	AHB_DATAWIDTH	AHB Write Data Bus
hprot_i	Input	4	AHB Protection type
pready_i	Input	y*1	Used for extending the transfer
pslverr_i	Input	y*1	Idicates a transfer failure
prdata_i	Input	y*APB_DATAWIDTH	Data from the APB slave
hclk	Input	1	AHB Clock
hrstn_i	Input	1	AHB Asynchronous active low reset
pclk	Input	1	APB Clock source
prstn_i	Input	1	APB Asynchronous active low reset

Table 3.2. I/O Pin configurations

The input pins include the hsel_i which has a datawidth of 1 bit and is a AHB Select line , haddr_i is a 32 bit bus which is AHB Address bus, htrans_i has a datawidth of 2 and it Indicates the transfer type of the current transfer hsize_i has a datawidth of 3 bits and it Indicates size of AHB transfer, hburst_i has a datawidth of 3 bits and it Indicates the burst type, hwrite_i is of 1 single bit and it Indicates direction of transfer. When HIGH indicates write access and when LOW indicates read access. hwdata_i has a width

equivalent to AHB_DATAWIDTH and is used to AHB Write Data Bus, hprot_i is of 4 bit and is AHB Protection type pin, pready_i is used for extending the transfer pslverr_i Indicates a transfer failure prdata_i has a width equivalent to APB_DATAWIDTH and tells the Data from the APB slave, hclk 1 bit AHB Clock hrstn_i is a single bit AHB Asynchronous active low reset, pclk is a single bit APB Clock source and prstn_i is a single bit APB Asynchronous active low reset

3.3 Features of AHB2APB bridge

The bridge basically converts system bus transfers into APB transfers and performs functions such as driving the data onto APB bus to perform read and write operations [4], holding the address throughout the transfer, converting system bus transfers into local bus transfers [5] between AHB and APB.

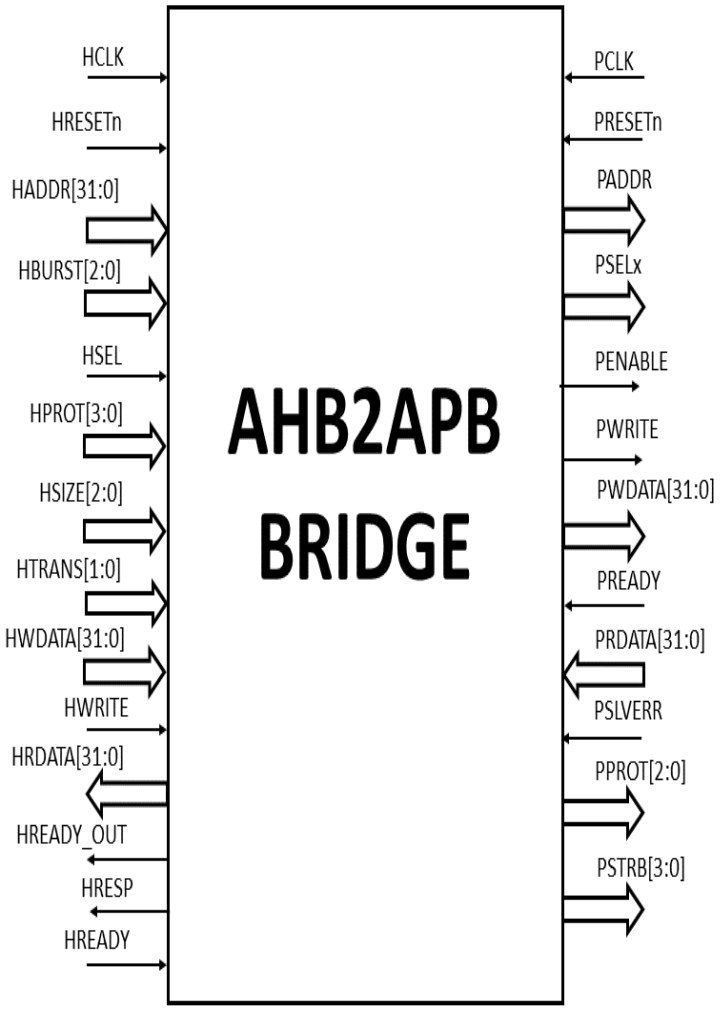


Figure 3.3. Pin Diagram

3.4. Implementation and working using fsm

3.4.1 AHB Slave Interface

The AHB Slave Interface consists of a state machine to implement the functionalities of the AHB slave. The state machines consist of 3 states:

- ST_AHB_IDLE
- ST_AHB_TRANSFER
- ST_AHB_ERROR

When reset is applied, the state machine will be in the ST_AHB_IDLE state. Figure. 4 represents the state transition that occurs when reset is de-asserted. [6]

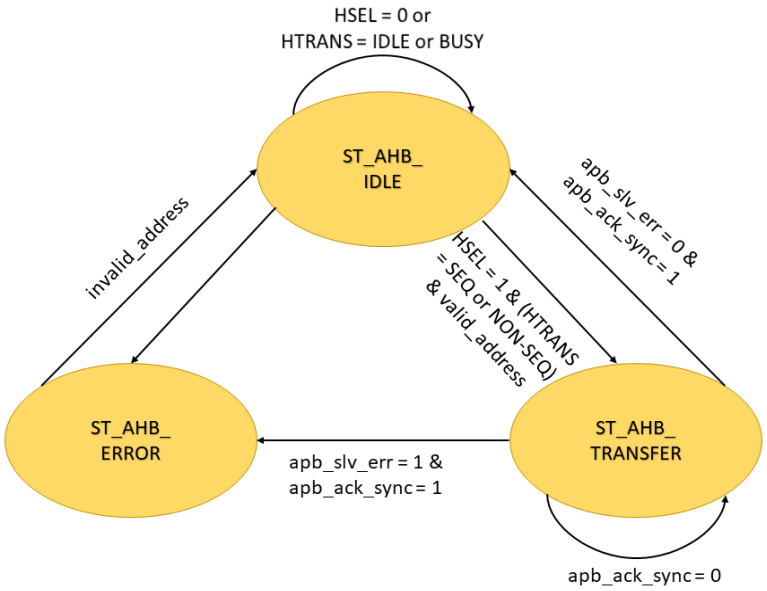


Figure 3.4. AHB State Machine

ST_AHB_IDLE If HSEL is de-asserted or if HTRANS indicates IDLE or BUSY transfer, then all the address and control registers will be reinitialized to their reset value. Once HSEL is asserted and HTRANS indicates SEQ or NON- SEQ transfer, then state transition will take place. If the address lies within the valid range, then the AHB address and control signals will be latched, and the next state will be ST_AHB_TRANSFER. If

the address is invalid, then HRESP will be set to 1, indicating an error response and state transition will occur to ST_AHB_ERROR.

ST_AHB_TRANSFER In this state, a start transfer signal (ahb_start_tr) will be toggled and ‘HREADY’ will be driven LOW, to wait for the APB slave to respond. At the rising edge of the ‘HWRITE’ signal, ‘HWDATA’, will be latched into a local register, for the APB slave to read. Once the acknowledgement is received from the APB slave that the transfer is complete, read data will be taken from the APBslave, if the transfer was a read access, and depending on the slave error, state transition will take place. If the APB slave has asserted the slave error, then ‘HREADY’ will be kept low (2 cycles are re- quired for the AHB master to detect slave error), and ‘HRESP’ will indicate an error. State transition will now occur to the ST_AHB_ERROR state. In case of no error, ‘HRESP’ will indicate an OK transfer and ‘HREADY’ will be driven HIGH to indicate the end of transfer. In this case, state will transit to the ST_AHB_IDLE state.

ST_AHB_ERROR In this state, ‘HREADY’ will be asserted, HRESP will continue to indicate error

response and state transition will occur to the ST_AHB_IDLE state in the next clock cycle.

3.4.2. AHB to APB Translation block

This will consist of the following blocks:

- Address translation
- Generation of PSEL
- PSTRB and PWDATA generation
- Synchronization between AHB and APB domain

Address Translation: In AHB, the address is aligned to the transfer size,

i.e., if it is an 8-bit transfer, the address will increment by 1 and if it is a 64-bit transfer then the address will be incremented by 8. But in the case of APB, the address will be aligned to the data bus width, i.e if the data bus is of 32 bits, then LSB 2 bits of PADDR

will be 0. (Address will increment by 4 for subsequent transfers). The number of LSBs in PADDR to be kept 0 is determined by taking the log of the number of bytes in the data bus.

Example: For 64-bit data bus, number of bytes = 64/8 = 8 Number of LSB 0s = log8 = 3

Generation of PSEL: The APB slave to which access is requested, is indicated by bits [15:12] of HADDR. These bits will represent the slave ID of the APB slave. PSEL of the slave will be set to 1 while that of other slaves will be de-asserted.

PSTRB and PWDATA Generation: To derive PWDATA, it is first required to know which part of HWDATA is valid. For example, if the AHB data bus is of 64 bits, and HSIZE indicates a 16-bit transfer, then it is necessary to know which 16 bits should be considered from the 64-bit data bus. This information can be taken from HADDR. The HADDR bits will represent the byte of the AHB data bus from which the transfer should start and HSIZE will determine how many bytes from the starting byte should be transferred. So the number of HADDR bits to be considered depends on the number of bytes in the AHB data bus (AHB_DATAWIDTH/8). By taking the log of this number, we get the number of bits required to represent the byte in the AHB data bus. Hence for a 64-bit data bus, the last 3 bits will determine the starting byte in HWDATA. The number of bytes to be transferred will depend on HSIZE. Based on this, a strobe(pstrb_full) indicating all active bytes will be generated marking the starting byte as 1 and it will keep marking the subsequent bytes as active till the number of active bytes equals the transfer size (represented by HSIZE). HWDATA will be shifted till we get the desired starting byte. But shifting will happen only in increments of the APB data bus width, i.e., in a 64-bit data bus, if HADDR [2:0] is 3'b110, it indicates that 6th byte is the first valid byte in HWDATA. If APB data bus is of 32 bits, then it can be shifted only 32 bits at a time (i.e. 4 bytes a time). So HWDATA will be right shifted by 32. Similarly, PSTRB will be generated by shifting the 'pstrb_full' signal till the first valid byte is reached. For a 32-bit data bus, the 'pstrb_full' will be shifted only 4 times a time (Number of bytes in the APB data bus) to ensure that it does not cross the APB data bus boundary. If HSIZE indicates 16-bit transfer, then PSTRB will be set as 4'1100.

Synchronization between AHB and APB domain: When AHB has data to send, it

toggles the ‘ahb_start_tr’ signal. This signal is passed through a shift register which is clocked by the APB clock. The size of the shift register depends on the synchronization depth required. MSB 2 bits of the synchronizer are provided as inputs to a EXOR gate to produce the synchronized output, ’start_tr_sync’. Similarly, APB toggles the acknowledgement signal, ‘apb_ack’ when the transfer is complete. This signal is then synchronized in the AHB domain, in the same way as the ‘ahb_start_tr’ signal. [7]

3.4.3 APB Master Interface

APB Master Interface is implemented through a state machine which has the following three states:

- ST_APB_IDLE
- ST_APB_SETUP
- ST_APB_TRANSFER

When reset is applied, the state machine will be in the ST_APB_IDLE state. Figure 5 represents the state transition that occurs when reset is de-asserted.

ST_APB_IDLE Initially the APB state machine will be in the ST_APB_IDLE state. When it receives a start transfer signal from the AHB, state transition will occur to the ST_APB_SETUP state.

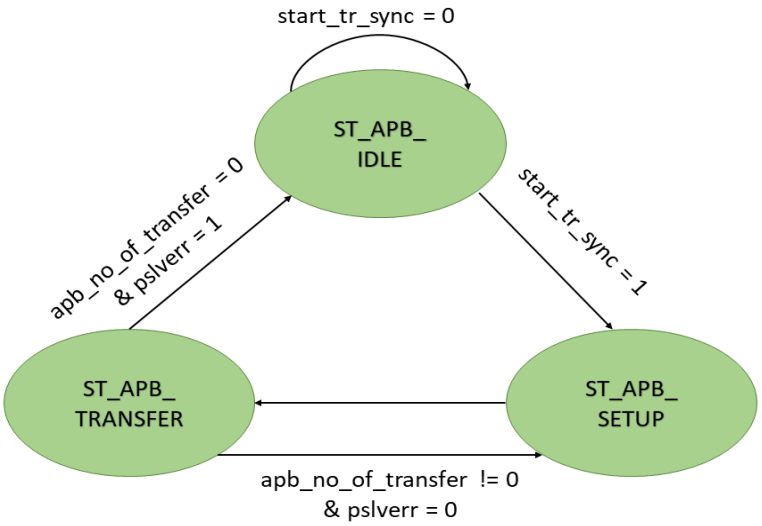


Figure 3.5. APB State Machine

ST_APB_SETUP In this state PSEL will be asserted depending on which slave is selected. PENABLE will remain de-asserted. State transition will occur to the ST_APB_TRANSFER state in the next clock cycle.

ST_APB_TRANSFER In this state, PENABLE will be asserted. It will remain in the same state till PREADY or PSLVERR is asserted by the slave. If there is a slave error, then it will abort the remaining transfers (if any) and go to the ST_APB_IDLE state. In case of no error, then it will go back to the ST_APB_SETUP state, if there are further transfers. Else, it will, go to the ST_APB_IDLE state. When the state transition occurs to the ST_APB_IDLE state, an acknowledgement will be sent to the AHB Slave interface

Chapter 4 SIMULATION RESULTS AND ANALYSIS

The synthesis of AHB2APB Bridge is done using Xilinx Vivado and a simple schematic of the synthesizable design is obtained as in Figure 4.1 which contains three blocks namely ahb2apb_sync, apb_master_interface and ahb_slave_interface.

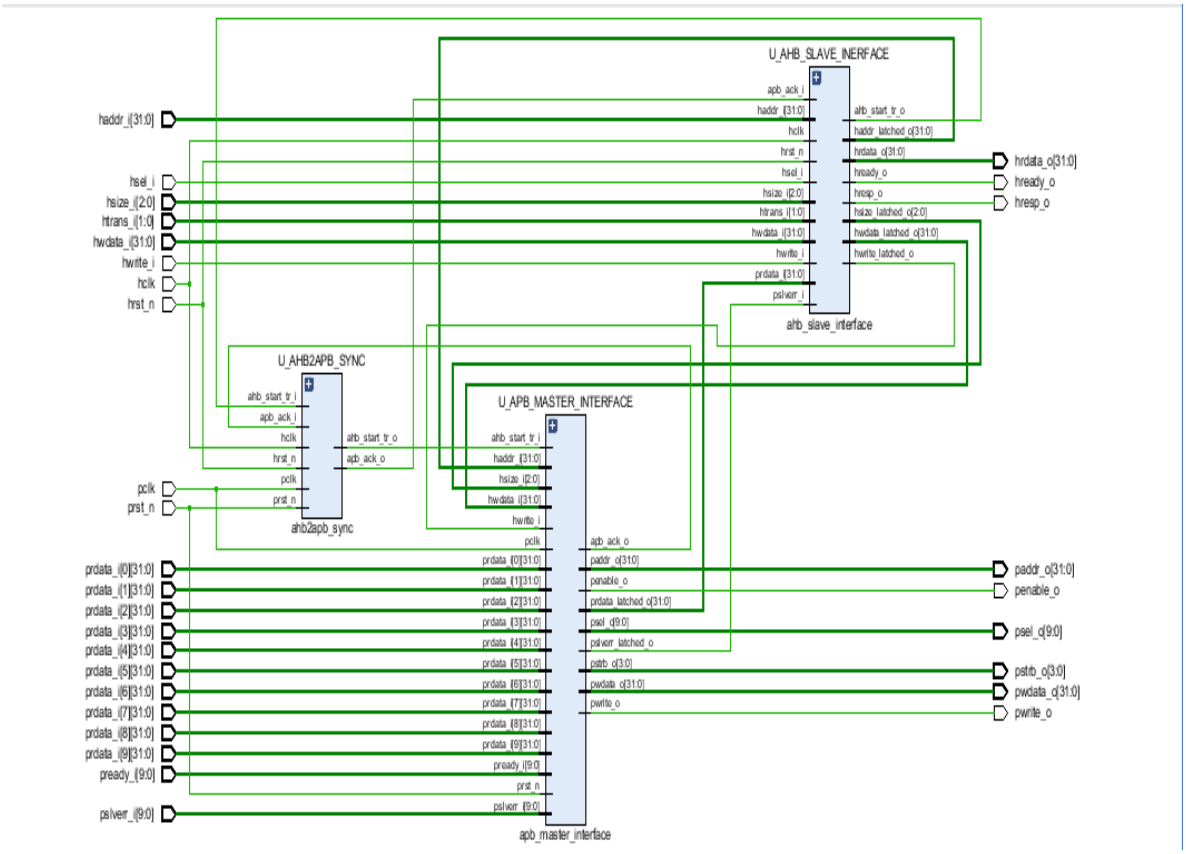


Figure 4.1. Schematic of AHB2APB Bridge

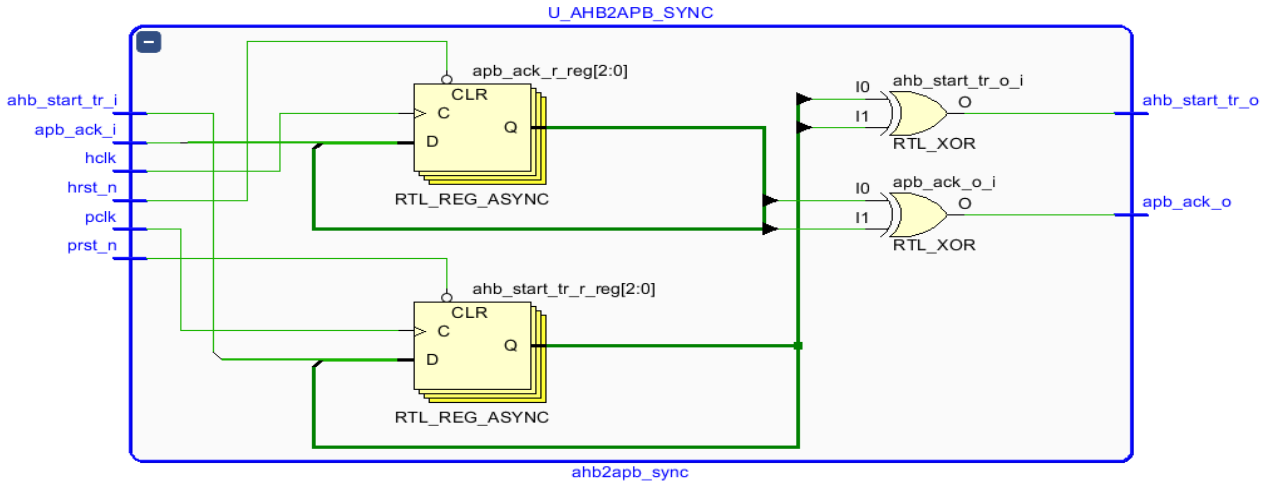


Figure 4.2. RTL Schematic of ahb2apb_sync

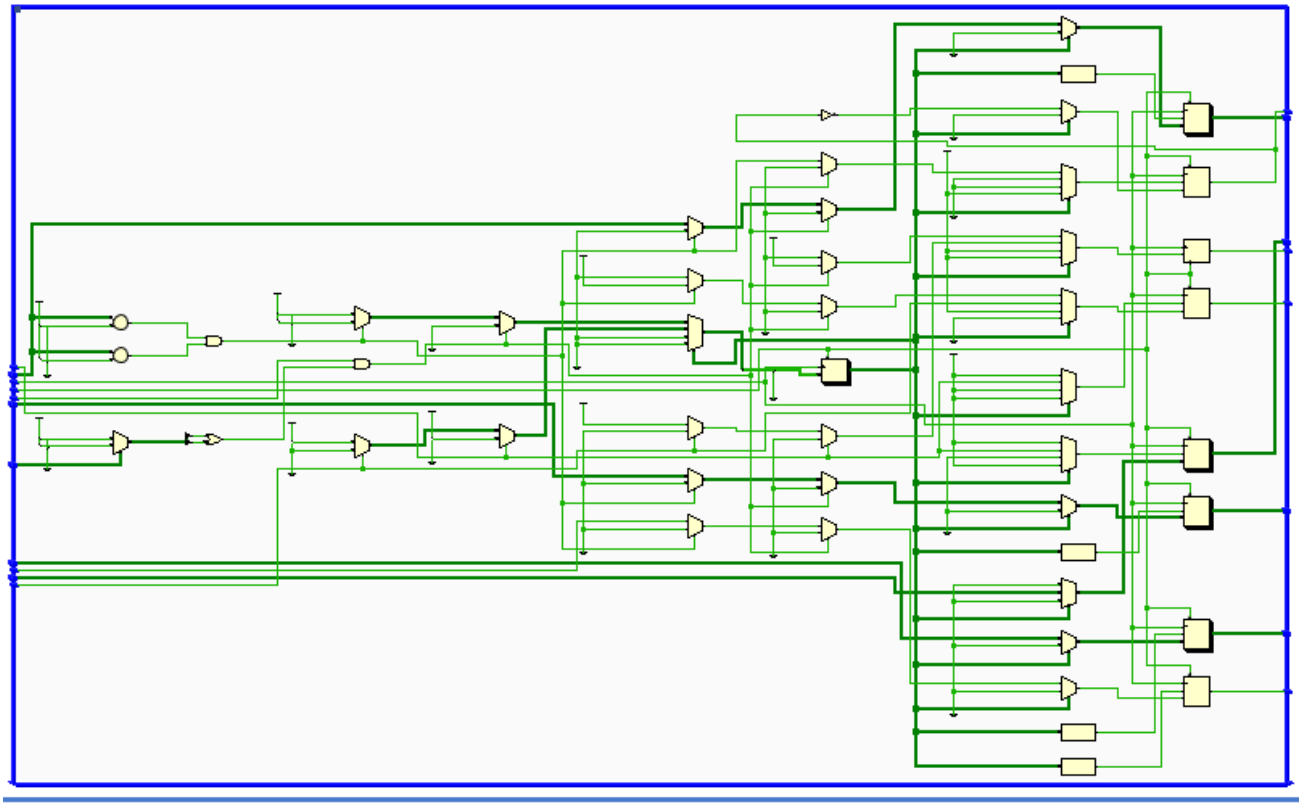


Figure 4.3. RTL Schematic of ahb_slave_interface

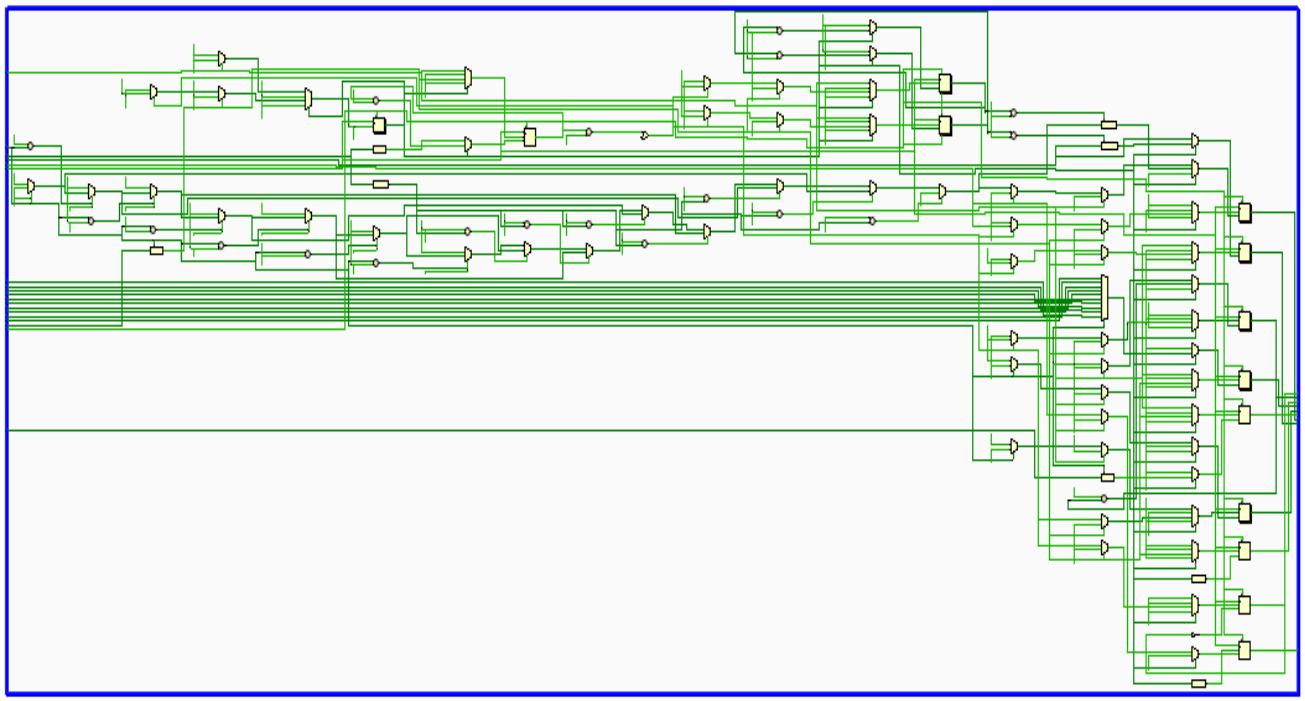


Figure 4.4. RTL Schematic of apb_master_interface

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	346	0	303600	0.11
LUT as Logic	346	0	303600	0.11
LUT as Memory	0	0	130800	0.00
Slice Registers	219	0	607200	0.04
Register as Flip Flop	219	0	607200	0.04
Register as Latch	0	0	607200	0.00
F7 Muxes	34	0	151800	0.02
F8 Muxes	0	0	75900	0.00

Table 4.1. Slice logic report

The above table depicts the slice logic report which is obtained by implementing the design on Virtex-7 pack. The entire design contains 533 cells, which contains 34 Muxes, 219 Flip flops.

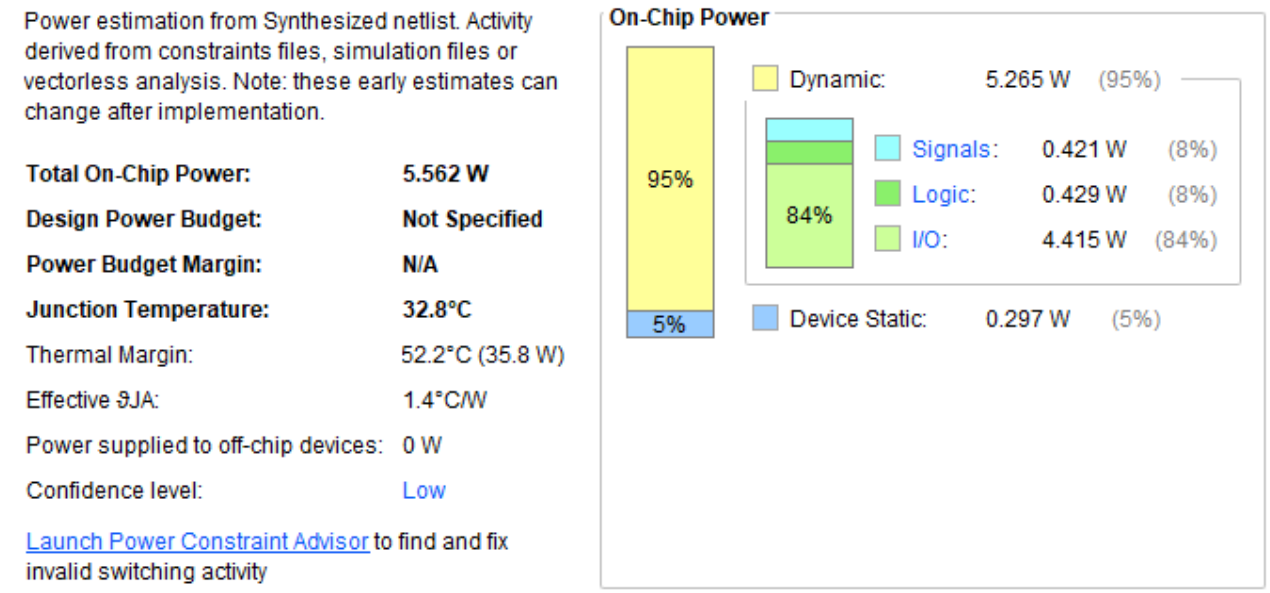


Figure 4.5. Power consumption

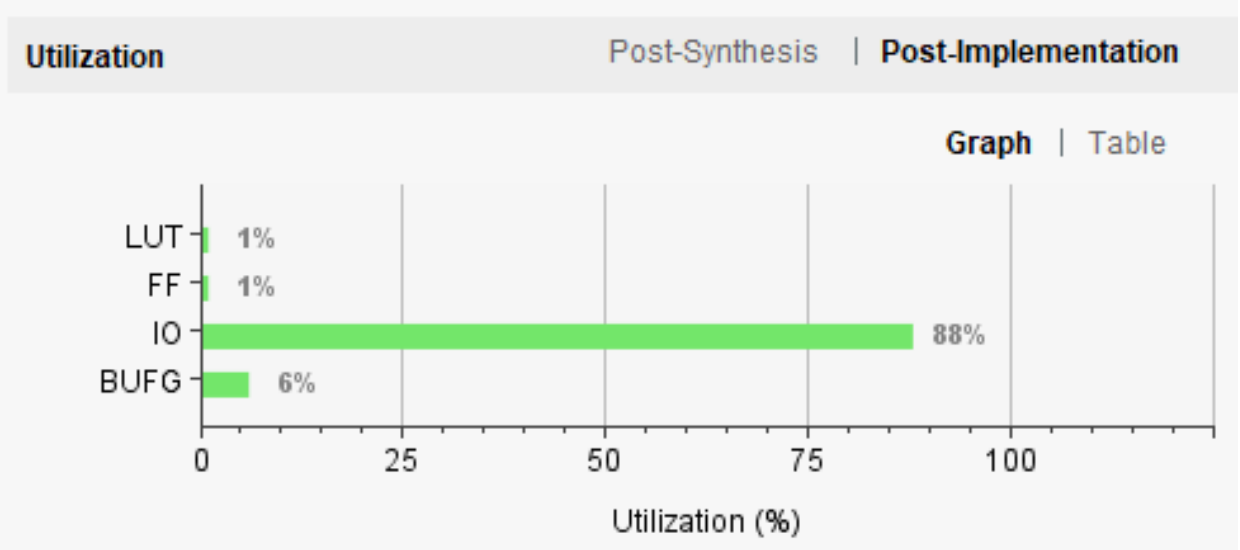
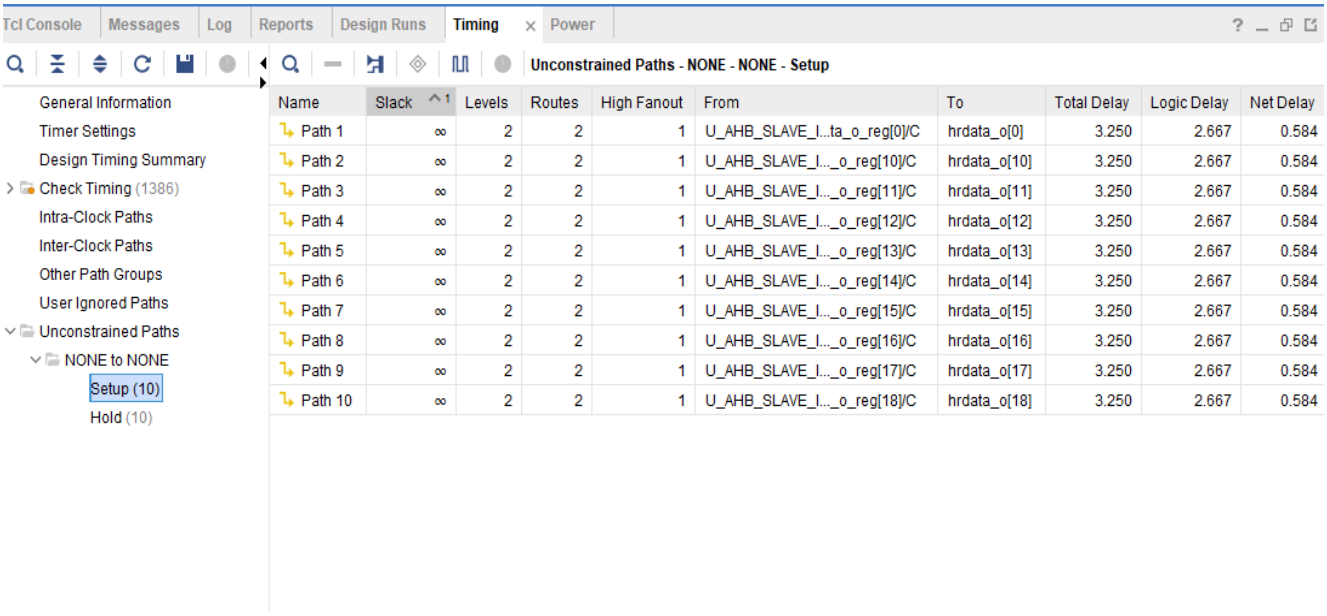


Figure 4.6. Utilization Graph

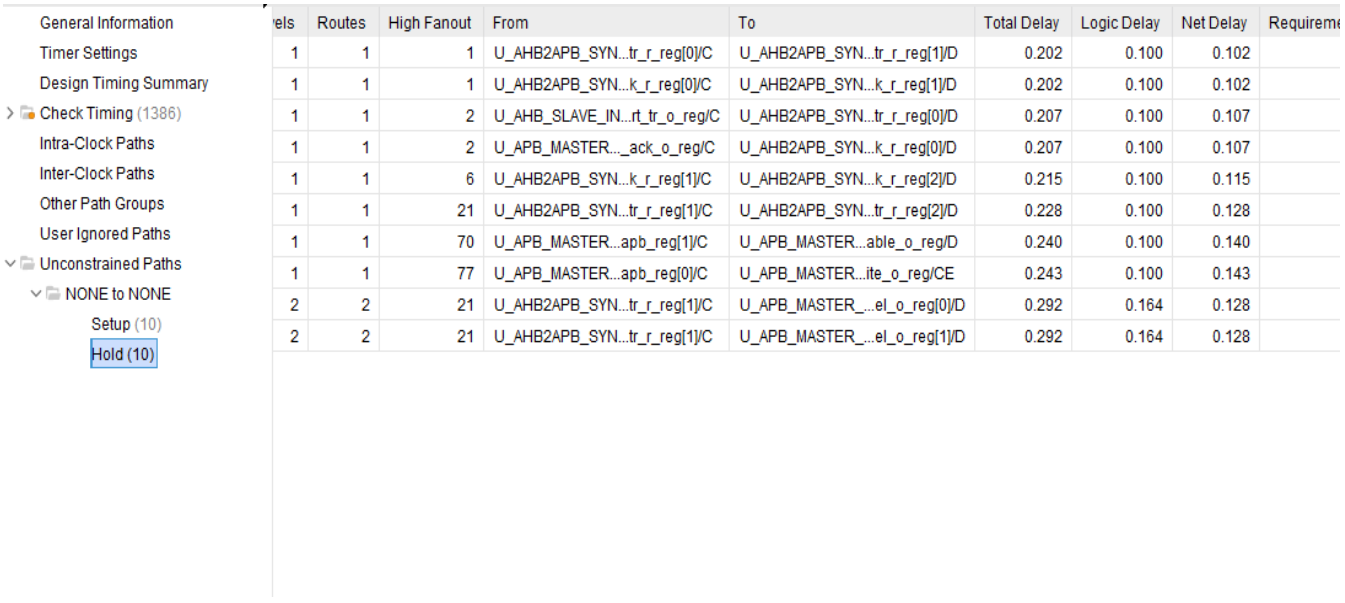
Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. The Total On-Chip Power is 5.562 W. Junction Temperature is 32.8°C. Thermal Margin is 52.2°C (35.8 W). Effective 9JA being 1.4°C/W. Power supplied to off-chip devices is 0 W and the Confidence level is Low.

The on-chip static power dissipation is 0.297W, and the dynamic power dissipation is 5.265W. By analyzing the timing report, the setup time is 3.25ns, and hold time 0.202ns



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	2	2	1	U_AHB_SLAVE_I...ta_o_reg[0]/C	hrdata_o[0]	3.250	2.667	0.584
Path 2	∞	2	2	1	U_AHB_SLAVE_I..._o_reg[10]/C	hrdata_o[10]	3.250	2.667	0.584
Path 3	∞	2	2	1	U_AHB_SLAVE_I..._o_reg[11]/C	hrdata_o[11]	3.250	2.667	0.584
Path 4	∞	2	2	1	U_AHB_SLAVE_I..._o_reg[12]/C	hrdata_o[12]	3.250	2.667	0.584
Path 5	∞	2	2	1	U_AHB_SLAVE_I..._o_reg[13]/C	hrdata_o[13]	3.250	2.667	0.584
Path 6	∞	2	2	1	U_AHB_SLAVE_I..._o_reg[14]/C	hrdata_o[14]	3.250	2.667	0.584
Path 7	∞	2	2	1	U_AHB_SLAVE_I..._o_reg[15]/C	hrdata_o[15]	3.250	2.667	0.584
Path 8	∞	2	2	1	U_AHB_SLAVE_I..._o_reg[16]/C	hrdata_o[16]	3.250	2.667	0.584
Path 9	∞	2	2	1	U_AHB_SLAVE_I..._o_reg[17]/C	hrdata_o[17]	3.250	2.667	0.584
Path 10	∞	2	2	1	U_AHB_SLAVE_I..._o_reg[18]/C	hrdata_o[18]	3.250	2.667	0.584

Figure 4.7. Setup check



els	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirem
1	1	1	U_AHB2APB_SYN...tr_r_reg[0]/C	U_AHB2APB_SYN...tr_r_reg[1]/D	0.202	0.100	0.102	
1	1	1	U_AHB2APB_SYN...k_r_reg[0]/C	U_AHB2APB_SYN...k_r_reg[1]/D	0.202	0.100	0.102	
1	1	2	U_AHB_SLAVE_IN...rt_tr_o_reg/C	U_AHB2APB_SYN...tr_r_reg[0]/D	0.207	0.100	0.107	
1	1	2	U_APB_MASTER..._ack_o_reg/C	U_AHB2APB_SYN...k_r_reg[0]/D	0.207	0.100	0.107	
1	1	6	U_AHB2APB_SYN...k_r_reg[1]/C	U_AHB2APB_SYN...k_r_reg[2]/D	0.215	0.100	0.115	
1	1	21	U_AHB2APB_SYN...tr_r_reg[1]/C	U_AHB2APB_SYN...tr_r_reg[2]/D	0.228	0.100	0.128	
1	1	70	U_APB_MASTER...apb_reg[1]/C	U_APB_MASTER...able_o_reg/D	0.240	0.100	0.140	
1	1	77	U_APB_MASTER...apb_reg[0]/C	U_APB_MASTER...ite_o_reg/CE	0.243	0.100	0.143	
2	2	21	U_AHB2APB_SYN...tr_r_reg[1]/C	U_APB_MASTER_...el_o_reg[0]/D	0.292	0.164	0.128	
2	2	21	U_AHB2APB_SYN...tr_r_reg[1]/C	U_APB_MASTER_...el_o_reg[1]/D	0.292	0.164	0.128	

Figure 4.8. Hold check

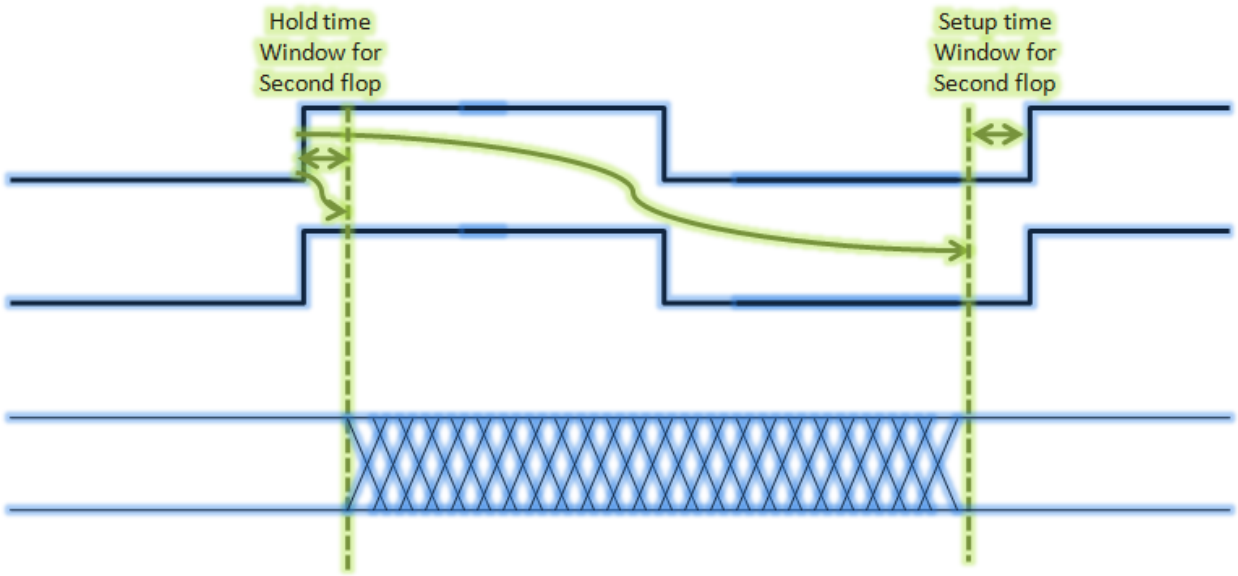


Figure 4.9. Waveform depicting setup and hold check.

Setup time is defined as the time before the active edge of the clock for which the data should be stable and Hold time is defined as the time after the active edge of the clock for which the data should be kept stable. If the setup or the hold time in the circuit is violated then it can result in the entire system being in a metastable state. The setup and the hold check is done on each active clock edge be it positive edge or a negative edge depending upon the requirement of the respective design.

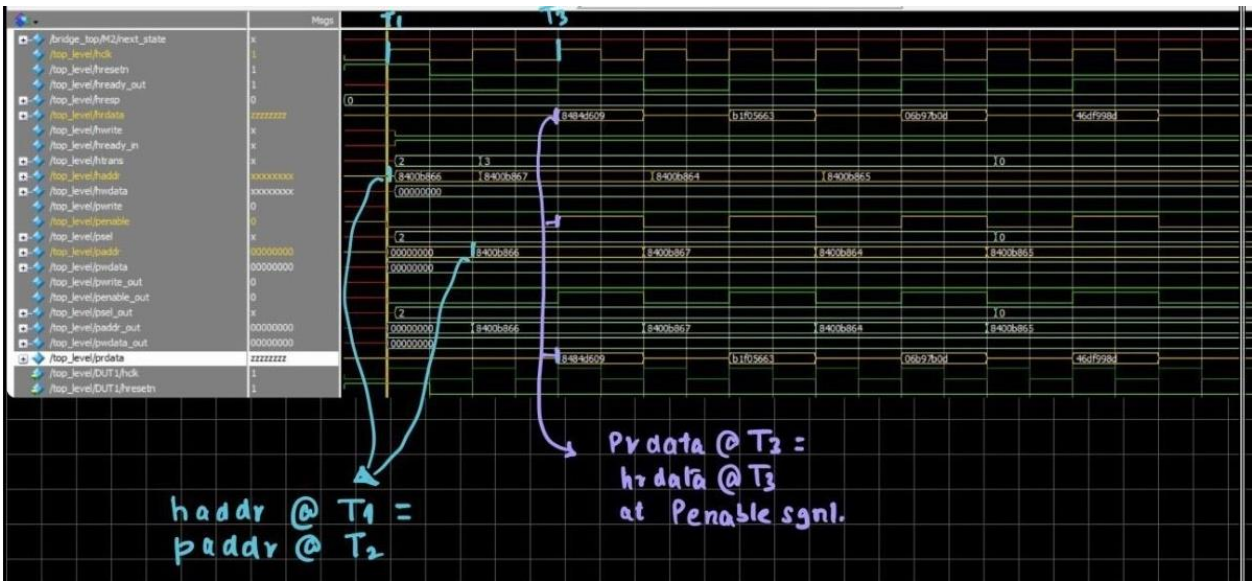


Figure 4.10. Burst write timing diagram

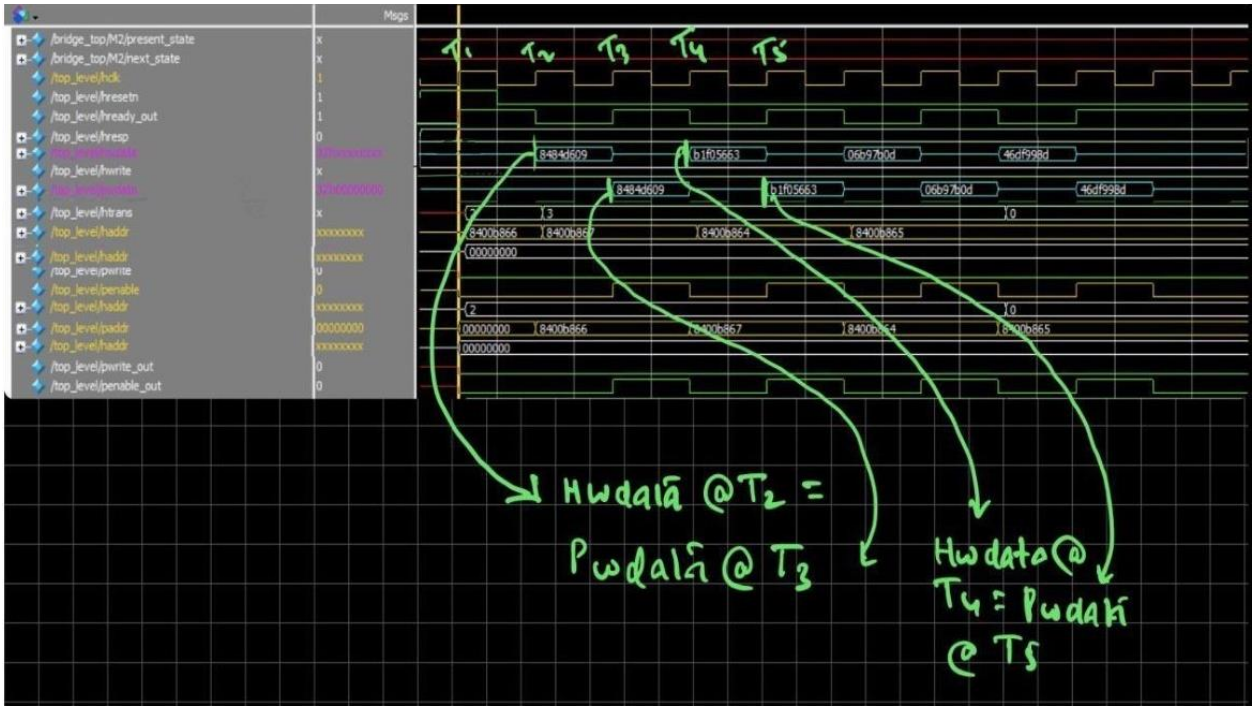


Figure 4.11. Burst read timing diagram

CHAPTER 5: CONCLUSION AND FUTURE SCOPE

The previous timing diagrams demonstrate that our RTL design, which is based on the AHBAPB bridge state schedule, is valid, as the Burst Read and Burst Write time diagrams demonstrate. The Xilinx Vivado tool was used to construct all of my RTL designs, and they were fully tested. The Advanced Trace Bus (ATB) and Advanced Expandable Interfaces (AXI) are given as part of online debugging solutions and solutions for even higher performance connections (ATB). The Sensical Hub Connector (CHI) standard, a large transport layer with traffic mitigation capabilities, was created.

These techniques have established the actual standard for integrated bus processor architecture since they are thoroughly defined and free to implement. Implementing the AHB2APB Bridge for many masters and slaves is one of the long-term aims. The design currently doesnot support a few features including Translation of HPROT to PPROT, Split and Retry slave responses, and, HMASTLOCK implementation which can be included in future designs. Verification methodologies can be further built using system verilog which can be quite useful in future.

APPENDIX A

```
module ahb2apb_top
  #(parameter AHB_DATAWIDTH = 32,
    parameter APB_DATAWIDTH = 32,
    parameter NUM_OF_APB_SLAVES = 10,
    parameter PERIPHERAL_START_ADDR = 32'h0008_0000,
    parameter PERIPHERAL_END_ADDR = 32'h0008_CFFF,
    parameter SYNC_STAGES = 2
    )
   (
    //Clock and reset signals
    //AHB Clock source
    input                       hclk,
    //APB Clock source
    input                       pclk,
    //Asynchronous active low reset with de-assertion synchronized to hclk
    input                       hrst_n,
    //Asynchronous active low reset with de-assertion synchronized to pclk
    input                       prst_n,
    //Input Signals from AHB Master
    //Select line from AHB Decoder
    input                       hsel_i,
    //Transfer type: 00 - Idle, 01- Busy, 10- Non-sequential, 11- Sequential
    input [1:0]                   htrans_i,
    //Write/read access 1- Write access; 0 - Read access
    input                       hwrite_i,
    //Indicates transfer size
    input [2:0]                   hsize_i,
    //Burst type; currently not used in the design
//    input [2:0]                    hburst_i,
    //Protection type; currently not used in the design
//    input [3:0]                    hprot_i,
    //Address bus
    input [31:0]                  haddr_i,
    //Write data bus
    input [AHB_DATAWIDTH-1:0]          hwdata_i,
    //Signals from all APB Slaves to the APB decode
    //Ready signal
    input [NUM_OF_APB_SLAVES-1:0]       pready_i,
    //Read data bus
    input [NUM_OF_APB_SLAVES-1:0] [APB_DATAWIDTH-1:0] prdata_i,
    //Slave error
    input [NUM_OF_APB_SLAVES-1:0]       pslverr_i,
    //APB Slave signals translated to AHB for sending it to AHB Master
    //Ready signal
    output logic                  hready_o,
    //Response signal; 0- OKAY; 1-ERROR
    output logic                  hresp_o,
```

```
    //Read data bus
    output logic [AHB_DATAWIDTH-1:0]     hrdata_o,
    //Signals translated from AHB Master sent to APB Slaves
    //Select signal
    output logic [NUM_OF_APB_SLAVES-1:0] psel_o,
    //Enable signal which is asserted from the second cycle of transfer to end of the transfer
    output logic                         penable_o,
    //Write/read access : 1-Write, 0-Read
    output logic                         pwrite_o,
    //Byte strobe signal to indicate active byte lane
    output logic [APB_DATAWIDTH/8-1:0]   pstrb_o,
    //Protection type
//    output logic [2:0]                  pprot_o,
    //Address bus
    output logic [31:0]                  paddr_o,
    //Write data bus
    output logic [APB_DATAWIDTH-1:0]     pwdata_o  );
   logic [31:0]                haddr_latched;
   logic [ 2:0]                hsize_latched;
   logic [31:0]                hwdata_latched;
   logic                       hwrite_latched;
 logic [AHB_DATAWIDTH-1:0]     prdata_frm_apb;
   logic                       pslverr_frm_apb;
   logic                       apb_ack;
   logic                       apb_ack_sync;
   logic                       ahb_start_tr;
   logic                       ahb_start_tr_sync;
  ahb_slave_interface
   #(
    .AHB_DATAWIDTH         (AHB_DATAWIDTH),
    .PERIPHERAL_START_ADDR  (PERIPHERAL_START_ADDR),
    .PERIPHERAL_END_ADDR    (PERIPHERAL_END_ADDR)
   )
   U_AHB_SLAVE_INERFACE
   (
    .hclk                 (hclk),
    .hrst_n               (hrst_n),
    .hsel_i               (hsel_i),
    .htrans_i             (htrans_i),
    .hwrite_i             (hwrite_i),
    .hsize_i              (hsize_i),
    .haddr_i              (haddr_i),
    .hwdata_i             (hwdata_i),
    .hrdata_o             (hrdata_o),
    .hready_o             (hready_o),
    .hresp_o              (hresp_o),
    .haddr_latched_o      (haddr_latched),
    .hsize_latched_o      (hsize_latched),
    .hwdata_latched_o     (hwdata_latched),
    .hwrite_latched_o     (hwrite_latched),
```

```
  .prdata_i              (prdata_frm_apb),
  .pslverr_i             (pslverr_frm_apb),
  .apb_ack_i              (apb_ack_sync),
  .ahb_start_tr_o         (ahb_start_tr)
 );
ahb2apb_sync
#(
   .SYNC_STAGES             (SYNC_STAGES)
 )
 U_AHB2APB_SYNC
 (
    .hclk                  (hclk),
    .hrst_n                 (hrst_n),
    .pclk                  (pclk),
    .prst_n                 (prst_n),
    .ahb_start_tr_i          (ahb_start_tr),
    .ahb_start_tr_o          (ahb_start_tr_sync),
    .apb_ack_i              (apb_ack),
    .apb_ack_o              (apb_ack_sync)
 );
apb_master_interface
#(
  .AHB_DATAWIDTH     (AHB_DATAWIDTH),
  .APB_DATAWIDTH     (APB_DATAWIDTH),
  .NUM_OF_APB_SLAVES (NUM_OF_APB_SLAVES)
)
U_APB_MASTER_INTERFACE
(
  .pclk              (pclk),
  .prst_n             (prst_n),
  .hwrite_i           (hwrite_latched),
  .hsize_i            (hsize_latched),
  .haddr_i            (haddr_latched),
  .hwdata_i           (hwdata_latched),
  .paddr_o            (paddr_o),
  .penable_o          (penable_o),
  .pwrite_o           (pwrite_o),
  .pwdata_o           (pwdata_o),
  .pstrb_o            (pstrb_o),
  .psel_o             (psel_o),
  .pready_i           (pready_i),
  .prdata_i           (prdata_i),
  .pslverr_i          (pslverr_i),
  .prdata_latched_o   (prdata_frm_apb),
  .pslverr_latched_o  (pslverr_frm_apb),
  .ahb_start_tr_i     (ahb_start_tr_sync),
  .apb_ack_o          (apb_ack)
 );

endmodule
```

APPENDIX B

```
module ahb2apb_sync
  #(parameter SYNC_STAGES = 2
    )
    (

    //AHB Clock source
    input                       hclk,
    //AHB Asynchronous Reset
    input                       hrst_n,

    //APB Clock source
    input                       pclk,
    //APB Asynchronous Reset
    input                       prst_n,
    //AHB Start of transfer
    input                       ahb_start_tr_i,

    //Synchronized AHB signal
    output                       ahb_start_tr_o,

    //APB Acknowledgment signal
    input                       apb_ack_i,
    //Synchronized APB signal
    output                       apb_ack_o
    );

   //Signals for CDC
   //Signal used for synchronizing ahb_start_tr. Used in the generation of ahb_start_tr_sync
   logic [SYNC_STAGES :0]                  ahb_start_tr_r;

   //Signal used for synchronizing apb_ack. Used in the generation of apb_ack_sync
   logic [SYNC_STAGES :0]                  apb_ack_r;

   //Clock domain crossing between AHB and APB domains
   //Synchronizing APB signal to AHB domain
   always@(negedge(hrst_n),posedge(hclk))
     begin
       if (hrst_n == 1'b0)
         apb_ack_r               <= 0;
       else
         apb_ack_r               <= {apb_ack_r[SYNC_STAGES-1:0], apb_ack_i};
     end
   assign apb_ack_o = apb_ack_r[SYNC_STAGES] ^ apb_ack_r[SYNC_STAGES-1];

   //Synchronizing AHB signal to APB domain
   always@(negedge(prst_n), posedge(pclk))
     begin
```

```
    if (prst_n == 0)
      ahb_start_tr_r            <= 0;
    else
      ahb_start_tr_r            <= {ahb_start_tr_r[SYNC_STAGES-1:0], ahb_start_tr_i};
  end
 assign ahb_start_tr_o = ahb_start_tr_r[SYNC_STAGES] ^ ahb_start_tr_r[SYNC_STAGES-1];


endmodule
```

APPENDIX C

```
// Filename     : ahb2apb.sv
// Author       : Pulkit
// Date         : 27-11-2021
// Version      : System Verilog
// Description  : AHB Slave Interface implemented through a FSM
module ahb_slave_interface
  #(parameter AHB_DATAWIDTH = 32,
    parameter PERIPHERAL_START_ADDR = 32'h0008_0000,
    parameter PERIPHERAL_END_ADDR = 32'h0008_CFFF
    )
   (

    //Clock and reset signals
    //AHB Clock source
    input                       hclk,
    //Asynchronous active low reset with de-assertion synchronized to hclk
    input                       hrst_n,

    //Input Signals from AHB Master
    //Select line from AHB Decoder
    input                       hsel_i,
    //Transfer type: 00 - Idle, 01- Busy, 10- Non-sequential, 11- Sequential
    input [1:0]                   htrans_i,
    //Write/read access 1- Write access; 0 - Read access
    input                       hwrite_i,
    //Indicates transfer size
    input [2:0]                   hsize_i,
    //Burst type; currently not used in the design
//    input [2:0]                    hburst_i,
    //Protection type; currently not used in the design
//    input [3:0]                    hprot_i,
    //Address bus
    input [31:0]                  haddr_i,
    //Write data bus
    input [AHB_DATAWIDTH-1:0]          hwdata_i,

    //APB Slave signals translated to AHB for sending it to AHB Master

    //Ready signal
    output logic                  hready_o,
    //Response signal; 0- OKAY; 1-ERROR
    output logic                  hresp_o,
    //Read data bus
    output logic [AHB_DATAWIDTH-1:0]     hrdata_o,

    //Latched AHB signals
    output logic [ 2:0]               hsize_latched_o,
```

```
    output logic [31:0]                haddr_latched_o,
    output logic [AHB_DATAWIDTH-1:0]     hwdata_latched_o,
    output logic                       hwrite_latched_o,
    //Signals from APB Slaves
    //APB data bus translated to AHB
    //Slave error
    input                          pslverr_i,
    input [AHB_DATAWIDTH-1:0]          prdata_i,

    //APB acknowledgement
    input                          apb_ack_i,
    //Signal to indicate start of transfer from AHB to APB slave
    output logic                       ahb_start_tr_o

    );

   //States used in AHB State Machine
   enum                            {ST_AHB_IDLE, ST_AHB_TRANSFER,ST_AHB_ERROR}
state_ahb;

   //htrans value for non-sequential transfer
   localparam HTRANS_NONSEQ   =   2'b10  ;
   //htrans value for sequential transfer
   localparam HTRANS_SEQ      =   2'b11  ;

   //AHB State machine
   always_ff@(negedge(hrst_n), posedge(hclk))
     begin
       if (hrst_n == 1'b0)
         begin
           haddr_latched_o              <= '0;
           hsize_latched_o              <= '0;
           hwrite_latched_o             <= '0;
           hrdata_o                   <= '0;
           hwdata_latched_o             <= '0;
           hresp_o                    <= '0;
           hready_o                    <= '1;
           ahb_start_tr_o               <= '0;
         end
       else
         begin
           case (state_ahb)
             ST_AHB_IDLE :
               begin
                 if ((hsel_i == 1'b1) && (htrans_i == 2'b10 || htrans_i == 2'b11))
                   if ((haddr_i >= PERIPHERAL_START_ADDR) && (haddr_i <=
PERIPHERAL_END_ADDR))
                     begin
                       //Control signals and address signals are provided in the first cycle of transfer
                       //So they are latched in the idle state itself
```

```
          haddr_latched_o          <= haddr_i;
          hsize_latched_o          <= hsize_i;
          hwrite_latched_o         <= hwrite_i;
          //Inserting wait states by pulling hreadyout to 0 till end of transaction
          hready_o                 <= 1'b0;
          hresp_o                  <= 'b0;
          //Toggling the signal to indicate start of transfer
          ahb_start_tr_o           <= ~ahb_start_tr_o;
        end
       else
        begin
          haddr_latched_o          <= '0;
          hsize_latched_o          <= '0;
          hwrite_latched_o         <= '0;
          hready_o                 <= '0;
          //Indicating error since HADDR does not lie within valid range
          hresp_o                  <= 2'b01;
        end
      else
       begin
         haddr_latched_o           <= '0;
         hsize_latched_o           <= '0;
         hwrite_latched_o          <= '0;
         hready_o                  <= 1'b1;
         hresp_o                   <= '0;
       end
      hrdata_o                     <= '0;
   end
 ST_AHB_TRANSFER :
   begin
     //Write data will be provided in the second cycle of transfer.
     //So it is latched in this state, instead of the idle state.
     hwdata_latched_o              <= hwdata_i;

     //When acknowledgement is received from the APB state machine, that the transfer is
     //complete, AHB output signals will be sent
     if (apb_ack_i == 1'b1)
       begin
         hrdata_o          <= prdata_i;
         //In case of no error, it is a single cycle response with
         //HREADY as 1 and HRESP as 0
         if (pslverr_i == 1'b0)
           begin
             hready_o   <= 1'b1;
             hresp_o    <= '0;
         end
         //In case of error, AHB will have 2 cycle response.
         //The first cycle will assert HRESP with HREADY as 0
         //In the next cycle (state will then be ST_AHB_ERROR state), both will be 1.
         else
```

```
                    begin
                      hready_o      <= 1'b0;
                      hresp_o       <= 2'b01;
                    end
                  end

                 else
                   //Wait cycles will be inserted till acknowledgement is received from the APB slave
                   hready_o          <= 1'b0;
                end
            ST_AHB_ERROR :
              begin
                //Second cycle of error response wherin hready will be pulled high, to indicate end of
transfer
                hready_o            <= 1'b1;
                hresp_o             <= 2'b01;
              end
            default :
              begin
                haddr_latched_o       <= '0;
                hsize_latched_o       <= '0;
                hwrite_latched_o      <= '0;
                hrdata_o              <= '0;
                hwdata_latched_o      <= '0;
                hresp_o               <= '0;
                hready_o              <= '1;
                ahb_start_tr_o        <= '0;
              end
          endcase
        end
    end
  //AHB State transitions
  always@(negedge(hrst_n), posedge(hclk))
    begin
      if (hrst_n == 0)
        state_ahb                     <= ST_AHB_IDLE;
      else
        case(state_ahb)
          ST_AHB_IDLE :

            //State transition will occur if HSEL is asserted and HTRANS indicates SEQ or NON-SEQ
transfer,
            //Else it will remain in the same state
            if ((hsel_i == 1'b1) && (htrans_i == 2'b10 || htrans_i == 2'b11))
              //If valid address then, go to ST_AHB_TRANSFER; else go to ST_AHB_ERROR
              if ((haddr_i >= PERIPHERAL_START_ADDR) && (haddr_i <=
PERIPHERAL_END_ADDR))
                state_ahb                 <= ST_AHB_TRANSFER;
              else
                state_ahb                 <= ST_AHB_ERROR;
```

```
          else
            state_ahb                 <= ST_AHB_IDLE;
        ST_AHB_TRANSFER :
          //If acknowledgment is received then state transition will occur.
          //If error then state will be ST_AHB_ERROR, else it will go to the ST_AHB_IDLE state
          if (apb_ack_i == 1'b1)
            if (pslverr_i == 1'b1)
              state_ahb               <= ST_AHB_ERROR;
            else
              state_ahb               <= ST_AHB_IDLE;
          else
            state_ahb                 <= ST_AHB_TRANSFER;
        //It is a fixed one cycle state, wherein it transits to ST_AHB_IDLE in the next cycle without
any condiiton
        ST_AHB_ERROR :
          state_ahb                   <= ST_AHB_IDLE;
        default :
          state_ahb                   <= ST_AHB_IDLE;
      endcase
  end
endmodule
```

APPENDIX D

```
module apb_master_interface
  #(parameter AHB_DATAWIDTH = 32,
    parameter APB_DATAWIDTH = 32,
    parameter NUM_OF_APB_SLAVES = 10
    )
   (

    //Clock and reset signals
    //APB Clock source
    input                           pclk,
    //Asynchronous active low reset with de-assertion synchronized to pclk
    input                           prst_n,
    //Input Signals from AHB Master
    //Write/read access 1- Write access; 0 - Read access
    input                           hwrite_i,
    //Indicates transfer size
    input [2:0]                      hsize_i,
    //Address bus
    input [31:0]                     haddr_i,
    //Write data bus
    input [AHB_DATAWIDTH-1:0]          hwdata_i,

    //Signals from all APB Slaves to the APB decoder
    //Ready signal
    input [NUM_OF_APB_SLAVES-1:0]       pready_i,
    //Slave error
    input [NUM_OF_APB_SLAVES-1:0]       pslverr_i,
    //Read data bus
    input [NUM_OF_APB_SLAVES-1:0] [APB_DATAWIDTH-1:0] prdata_i,

    //Signals translated from AHB Master sent to APB Slaves
    //Select line to select the corresponding APB slave
    output logic [NUM_OF_APB_SLAVES-1:0] psel_o,
    //Enable signal which is asserted from the second cycle of transfer to end of the transfer
    output logic                    penable_o,
    //Write/read access : 1-Write, 0-Read
    output logic                    pwrite_o,
    //Byte strobe signal to indicate active byte lane
    output logic [APB_DATAWIDTH/8-1:0]   pstrb_o,
    //Address bus
    output logic [31:0]               paddr_o,
    //Write data bus
    output logic [APB_DATAWIDTH-1:0]     pwdata_o,
    //Signals sent to AHB slave
    //Slave error indication
    output logic                    pslverr_latched_o,
    //Read data used by AHB interface
```

```
    output logic [AHB_DATAWIDTH-1:0]     prdata_latched_o,

    //Start transfer signal from AHB interface
    input                          ahb_start_tr_i,
    //Acknowledgement signal used to indicate end of transfer to AHB
    output logic                    apb_ack_o
    );

   //States used in APB State Machine
   enum                           {ST_APB_IDLE, ST_APB_SETUP,ST_APB_TRANSFER} state_apb;

   //Indicates which bits of HADDR should be considered for detecting the active byte lane
   localparam UPPER_INDEX = $clog2(AHB_DATAWIDTH/8);

   localparam LOWER_INDEX = ($clog2(APB_DATAWIDTH/8));

   //Used to pad zeros for PADDR, to make it aligned with the data bus; For 32-bit data bus, last 2
bits of PADDR
   //should be zero. Also used in deriving PSTRB
   localparam NO_OF_LSB_0              = ($clog2(APB_DATAWIDTH/8));
   localparam [NO_OF_LSB_0-1:0] LSB_ZERO =  '0;
   //Used to find the offset by which HWDATA should be shifted
   logic [$clog2(AHB_DATAWIDTH)-1:0]          byte_offset;
   logic [$clog2(AHB_DATAWIDTH)-1:0]          upper_strobe_index;
   logic [$clog2(AHB_DATAWIDTH)-1:0]          upper_data_index;
   //Used when transfer size is greater than APB_DATAWIDTH and requires multiple APB transfers
   //Indicates the MSB for data register
   integer                                pdata_msb_r;
   //Indicates the MSB for strobe register
   integer                                pstrb_msb_r;
   //Indicates all the active byte lanes
   logic [AHB_DATAWIDTH/8 -1 :0]              pstrb_full;
   //Keeps a count of remaining APB transfers
   logic [AHB_DATAWIDTH/APB_DATAWIDTH-1:0]      transfers_pending_r;

   //Used to address the APB slave to which AHB wishes to have access
   logic [$clog2(NUM_OF_APB_SLAVES)-1:0]        slv_id;


  //Functions used in the design
  //Used to derive the number of APB tranfers for each AHB transfer
  function logic [AHB_DATAWIDTH/APB_DATAWIDTH-1:0] apb_pending_transfers;
    input [2:0] hsize;
    case (hsize)
      3'b000 : apb_pending_transfers      = 7/APB_DATAWIDTH;
      3'b001 : apb_pending_transfers      = 15/APB_DATAWIDTH;
      3'b010 : apb_pending_transfers      = 31/APB_DATAWIDTH;
      3'b011 : apb_pending_transfers      = 63/APB_DATAWIDTH;
      3'b100 : apb_pending_transfers      = 127/APB_DATAWIDTH;
      3'b101 : apb_pending_transfers      = 255/APB_DATAWIDTH;
```

```
    3'b110 : apb_pending_transfers      = 511/APB_DATAWIDTH;
    3'b111 : apb_pending_transfers      = 1023/APB_DATAWIDTH;
    default: apb_pending_transfers      = 0;
  endcase
endfunction


//Determining the number of bytes to be transferred
function logic [($clog2(AHB_DATAWIDTH/8)):0] hsize_in_bytes;
  input [2:0]hsize;
  case(hsize)
    3'b000 : hsize_in_bytes      = 'd1;
    3'b001 : hsize_in_bytes      = 'd2;
    3'b010 : hsize_in_bytes      = 'd4;
    3'b011 : hsize_in_bytes      = 'd8;
    3'b100 : hsize_in_bytes      = 'd16;
    3'b101 : hsize_in_bytes      = 'd32;
    3'b110 : hsize_in_bytes      = 'd64;
    3'b111 : hsize_in_bytes      = 'd128;
    default: hsize_in_bytes      = 'd1;
  endcase
endfunction

//Used to generate the byte strobe which indicates which bytes in HWDATA are valid
function logic [(AHB_DATAWIDTH/8)-1:0] full_byte_strobe;
  input [2:0] hsize;
  input [NO_OF_LSB_0-1:0] haddr;
  logic [($clog2(AHB_DATAWIDTH/8)):0] transfer_size;
  transfer_size = hsize_in_bytes(hsize);
  full_byte_strobe = '0;
  for (integer index = 0; index < transfer_size; index = index + 1)
    full_byte_strobe[haddr + index] = 1'b1;
endfunction

//Generation of byte strobe which indicates which bytes in the HWDATA should be sent
always_comb
begin
  byte_offset                       <= '0;
  byte_offset[UPPER_INDEX-1:0]          <= haddr_i[UPPER_INDEX-1:0];
  byte_offset[LOWER_INDEX-1:0]          <= '0;
  upper_strobe_index                  <= byte_offset + APB_DATAWIDTH/8 -1;
  upper_data_index                    <= {byte_offset,3'b0} + APB_DATAWIDTH-1;
  pstrb_full                        <= full_byte_strobe(hsize_i, haddr_i[NO_OF_LSB_0-1:0]);

end

//Decoding the slave address. Each slave has a address space of 4KB. In case of different address
//allocation, change the bit indices of HADDR accordingly
assign slv_id   = haddr_i[15:12];
 //APB state machine
```

```
always_ff@(negedge(prst_n), posedge(pclk))
  begin
    if (prst_n == 0)
      begin
        psel_o                  <= '0;
        penable_o                 <= 1'b0;
        pwrite_o                 <= 1'b0;
        pstrb_o                 <= '0;
        pslverr_latched_o           <= 1'b0;
        transfers_pending_r          <= 'b0;
        pdata_msb_r               <= APB_DATAWIDTH-1;
        pstrb_msb_r               <= APB_DATAWIDTH/8 -1;
        psel_o                  <= '0;
        apb_ack_o                  <= 1'b0;
        prdata_latched_o             <= '0;
        paddr_o                 <= '0;
        pwdata_o                  <= '0;
      end
    else
      case (state_apb)
        ST_APB_IDLE :
          begin
            //APB signals are put in the bus, when start transfer is indicated from the AHB
            if (ahb_start_tr_i == 1)
              begin
                pwdata_o            <= hwdata_i[upper_data_index-:APB_DATAWIDTH];
                pwrite_o            <= hwrite_i;
                paddr_o             <= {haddr_i[31:NO_OF_LSB_0], LSB_ZERO};
                psel_o[slv_id]       <= 1'b1;
                penable_o           <= 1'b0;
                pslverr_latched_o     <= 1'b0;
                if (hwrite_i == 1'b1)
                  pstrb_o           <= pstrb_full[upper_strobe_index-:APB_DATAWIDTH/8];
                else
                  pstrb_o           <= '0;
                transfers_pending_r   <= apb_pending_transfers(hsize_i);
                pdata_msb_r          <= upper_data_index + APB_DATAWIDTH;
                pstrb_msb_r          <= upper_strobe_index + APB_DATAWIDTH/8;
                pslverr_latched_o     <= 1'b0;
                prdata_latched_o      <= '0;
              end
            else
              begin
                paddr_o             <= '0;
                psel_o              <= '0;
                penable_o            <= 1'b0;
                pwrite_o            <= 1'b0;
                pstrb_o             <= '0;
                //To retain the data so that AHB has sufficient time to sample it
                pslverr_latched_o     <= pslverr_latched_o;
```

```
            prdata_latched_o      <= prdata_latched_o;
          end
        end
      //PENABLE is asserted in the second cycle of transfer
      ST_APB_SETUP :
        penable_o                 <= 1'b1;
      ST_APB_TRANSFER:
        begin
          //Once PREADY is asserted, the PRDATA is taken . If in case of PSLVERR, the transfer is
halted
          //i.e., if there are any pending transfers, they will be aborted. Acknowledgement is sent
indicating
          //end of transfer. In case of no error, the next transfer will be initiated until all the valid data
in
          //HWDATA is sent
          if (pready_i[slv_id] == 1)
            begin
              penable_o           <= 0;
              prdata_latched_o      <= {prdata_latched_o, prdata_i[slv_id]};
              pslverr_latched_o     <= pslverr_i[slv_id];
            if (pslverr_i[slv_id] == 1 || transfers_pending_r == 0)
              begin
                apb_ack_o          <= ~apb_ack_o;
                psel_o             <= '0;
              end
              else
                begin
                  paddr_o[31:LOWER_INDEX]  <= paddr_o[31:LOWER_INDEX] + 1;
                  pwdata_o               <= hwdata_i[pdata_msb_r-:APB_DATAWIDTH];
                  if (hwrite_i == 1'b1)
                    pstrb_o              <= pstrb_full[pstrb_msb_r-:APB_DATAWIDTH/8];
                  pdata_msb_r             <= pdata_msb_r + APB_DATAWIDTH;
                  pstrb_msb_r             <= pstrb_msb_r + APB_DATAWIDTH/8;
                  penable_o               <= 1'b0;
                end
              transfers_pending_r       <= transfers_pending_r - 1;
            end
          end
        default :
          begin
            paddr_o                 <= '0;
            psel_o                  <= '0;
            penable_o                 <= 1'b0;
            pwrite_o                <= 1'b0;
            pstrb_o                 <= '0;
          end
      endcase
    end
  //APB State Transition
  always@(negedge(prst_n), posedge(pclk))
```

```
    begin
      if (prst_n == 0)
        state_apb                   <= ST_APB_IDLE;
      else
        case(state_apb)
          ST_APB_IDLE :
            //APB transfer starts with the reception of start transfer signal from AHB slave
            if (ahb_start_tr_i == 1)
              state_apb             <= ST_APB_SETUP;
            else
              state_apb             <= ST_APB_IDLE;
          ST_APB_SETUP :
            state_apb               <= ST_APB_TRANSFER;
          ST_APB_TRANSFER :
            //Stay in the same state until PREADY is received. If there is a slave error,
            //or if all transfers are completed, then move to idle state. If there are
            //pending transfers, go to the st_apb_enable state
            if (pready_i[slv_id] == 1)
              if(pslverr_i[slv_id] == 1 || transfers_pending_r == 0)
                state_apb           <= ST_APB_IDLE;
            else
              state_apb             <= ST_APB_SETUP;
            else
              state_apb             <= ST_APB_TRANSFER;
          default :
            state_apb               <= ST_APB_IDLE;
        endcase
    end
endmodule
```

REFERENCES

1. "AMBA Specification (Rev 2.0)", available at http://www.arm.com.
2. Kiran Rawat et al. (2015). "RTL Implementation for AMBA ASB APB Protocol at System on Chip Level" 2nd International Conference on Signal Processing and Integrated Networks (SPIN,) pp. 927- 930.
3. Jasmine Chhikara et al. (2015). "Implementing Communication Bridge between 12C and APB" IEEE International Conference on Computational Intelligence & Communication Technology, pp. 235-238.
4. ARM. "AMBA Open Specifications" http://www.arm.com/products/system-ip/amba/ambaopen specifications.php.
5. Raed M. Salih and Laszek T. Lilien (2015). "Protecting Users' Privacy in Healthcare Cloud Computing with APB-TTP" IEEE International Conference on Pervasive Computing and Communication Workshops (Per Com Workshops), pp. 236-238.
6. L. Benini, A. Macii et al. (2000). "Architectures and Synthesis Algorithms for Power-efficient Bus Interfaces," IEEE Trans. Computer-Aided Design, vol. 19, pp. 969–980.
7. "AdvancedMicrocontrollerBusArchitecture (AMBA)"http://en.wikepedia.org/wiki/ Advanced_MicrocontrollerBus_Architecture.
8. Roopa M. et al. (2013) "UART Controller as AMBA APB Slave" National Conference on Challenges in Research & Technology in the Coming Decades, pp. 1-6.
9. Ashutosh Gupta et al. (2016). "Physical Design Implementation of 32-bit AMBA ASB APB module with improved performance" International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), pp. 3121-3124.
10. Ge Zhiwei et al. (2009). "Design of On-chip Image Processing Based on APB Bus with CMOS Image Sensor" IEEE 8th International Conference on ASIC, pp. 963-966.
11. Gandhani, P., & Patel, C. (2011). Moving fromAMBA AHB to AXI Bus in SoC Designs: A Comparative Study. International Journal of Computer Science & Emerging Technologies (IJCSET), 2(4), 476-479.
12. Yasemin M. Akay et al. (2009). "Hippocampal gamma Oscillations in Rats" IEEE Engineering in Medicine and Biology Magazine, vol. 28, pp. 92-95.

13. Chenghai Ma et al. (2011) “Design and Implementation of APB Bridge based on AMBA 4.0” International Conference on Consumer Electronics, Communications and Networks (CECNet), pp. 193-196.

14. Sharma, Archana C., and C. Z. Ali. "Construct High-Speed SDRAM Memory Controller Using Multiple FIFO’s for AHB Memory Slave Interface." International Journal of Emerging Technology and Advanced Engineering3, no. 3 (2013): 907-916.

15. Kiran Rawat et al. (2015). “Implementation of AMBA APB Bridge with Efficient Deployment of System Resources” International Conference on Computer, Communication and Control (IC4), pp. 1- 4.

16. M. R. Stan and W. P. Burleson (1995). “Bus Invert Coding for Low Power I/O,” IEEE Trans. VLSI Systems, vol. 3, pp. 49–58.

17. Acasandrei, Laurentiu, and Angel Barriga. "AMBA bus hardware accelerator IP for Viola-Jones face detection."IET Computers & Digital Techniques 7, no. 5 (2013): 200-209.

18. Kanishka Lahiri and Anand Raghunanthan (2004). “Power Analysis of System-Level On-Chip Communication Architectures” International Conference on Hardware/Software Codesign and System Synthesis, pp. 236-241.

19. C.-T. Hsieh and M. Pedram (2002). “Architectural Power Optimization by Bus Splitting,” IEEE Trans. Computer-Aided Design, vol. 21, pp. 408–414.

20. Kandiya, M. M. N., Harniya, M. M. K., & Govani, K. K. (2014). Implementation of Read/Write operation for AMBA AXI4 Bus using VHDL. IJFTMR. I, IV, 1-3.

21. Kiran Rawat et al. (2014). “Design of AMBA APB Bridge with Reset Controller for Efficient Power Consumption” 9th International Conference on Industrial and Information Systems (ICIIS), pp.1-5.

22. Miss. Dhage Naiyna Kashinath and Prof. S.I. Nipanikar (2015). “AMBA Bus with Multiple Masters Using VLSI” IJEDR, vol. 3, pp. 97-102.

23. J. Y. Chen et al. (1999). “Segmented Bus Design for Low Power,” IEEE Trans. VLSI Systems, vol. 7, pp. 25–29.

24. 17 P. P. Sotiriadis and A. P. Chandrakasan (2002). “A Bus Energy Model for Deep Submicron Technology,” IEEE Trans. VLSI Systems, vol. 10, pp. 341–350.

25. T. Lv et al. (2003). “A Dictionary-based En/decoding Scheme for Low-power Data buses,” IEEE Trans. VLSI Systems, vol. 11, pp. 943–951.

26. Samir Palnitkar. (2003). Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition Publisher:Prentice Hall PTR. [28] Xilinx Datasheet: www.xilinx.com/zynq7000-Pkg-Pinout [29] www.aldec.com/en/company/blog/144--introduction-to-ZYNQ-architecture [30] www.store.digilentic.com

27. Kareemullah Shaik, Mohammad Mohiddin, Md. Zabirullah, “A Reduced Latency Architecture for Obtaining High System Performance”, IJRTE, 2012.