**IMPLEMENTATION OF APB PROTOCOL USING FPGA**

**Project ID:** **26037**

*B.Tech. Project Report*

*submitted for fulfillment of*

*the requirements for the*

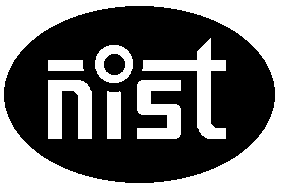
*Degree of Bachelor of Technology*

*Under Biju Pattnaik University of Technology*

*Submitted By*

**Kirti Kumar Roll No: ECE201910212**

**Kumar Anurag Roll No: ECE201910364**



*2022 – 2023*

*Under the guidance of*

**Dr. M. Suresh**

**NIST INSTITUTE OF SCIENCE & TECHNOLOGY (Autonomous)**

**Palur Hills, Berhampur, Odisha – 761008, India**

# ACKNOWLEDGEMENT

It is our proud privilege to epitomize our deepest sense of gratitude and indebtedness to our advisor, **Dr. M. Suresh** for his valuable guidance, keen and sustained interest, intuitive ideas and persistent endeavour. Her inspiring assistance, laconic reciprocation and affectionate care enabled us to complete our work smoothly and successfully.

We extend our sincere thanks to **Prof. Rajesh Kumar Dash, B.Tech. Project Coordinator and Prof. Bibhuti Bhusan Mishra, Department Project Coordinator**, for giving us the opportunity and motivating us to complete the project within stipulated period of time and providing a helping environment.

We acknowledge with immense pleasure the sustained interest, encouraging attitude and constant inspiration rendered by **Prof. (Dr.) Sukant K. Mohapatra** (Chairman), **Prof. (Dr.) Priyadarshi Tripathy** (Principal) and **Dr. M. Suresh** (HoD, Department of Electronics and Communication Engg.) N.I.S.T. Their continued drive for better quality in everything that happens at N.I.S.T and selfless inspiration has always helped us to move ahead.

# Kirti Kumar Kumar Anurag

# Roll No: ECE 201910212 Roll No: ECE 201910364

# TABLE OF CONTENTS

[**ACKNOWLEDGEMENT 1**](#_Toc105755785)

[**TABLE OF CONTENTS 2**](#_Toc105755786)

[**LIST OF FIGURES 3**](#_Toc105755787)

[**LIST OF TABLES 4**](#_Toc105755787)

[**PROBLEM STATEMENT 5**](#_Toc105755788)

**1. INTRODUCTION…………………………………………………………………6**

**2. FPGA……………..………………………………………..………………………..6**

**2.1) INTRODUCTION…………………………………….……………….6**

**2.2) HISTORY……………………………………………………….……….8**

**2.3) COMPONENTS…………………………………………………...….9**

**2.4) ADVANTAGES………………………………………………………..**

**2.5) BOARD USED IN PROJECT………………………………………**

**3. FINITE STATE MACHINE……………………………………………………..**

**3.1) INTRODUCTION…………………………………………………**

**3.2) TYPES …………………………………………….……………………**

**3.3) ADVANTAGES ………………………………………………**

**3.4) DISADVANTAGES ………………………………………………**

**4. VERILOG HDL**

**5. APB PROTOCOL………………………………….…………………….…….…11**

**3.1) INTRODUCTION……………………………………….……………….11**

**3.2) AMBA INTERCONNECT SPECIFICATION…….………...11**

**3.3) NEED FOR PROTOCOL…………………………….………...11**

**3.4) SIGNAL DESCRIPTION………………………..….………...11**

**3.5) OPERATING STATES……………………………….………………15**

**3.6) OUTPUT WAVEFORMS……………………………….………….16**

**6. SCHEMATIC.………………………………………………..….…………….....17**

**7. UTILIZATION REPORT………………………………………………………..19**

**8. DESIGN FLOW…………………….…………………………………..………..27**

**8.1) DESIGN ENTRY……………………………………………**

**8.2) SYNTHESIS…………………………………………**

**8.3) IMPLEMENTATION…………………………………………**

**8.4) PROGRAM FPGA…………………………………………**

**9. DESIGN CHALLENGES AND LIMITATIONS……………..………..27**

**10 .CONCLUSION……………………………………………………………………..20**

**REFERENCES……………………………………………………………………….…21**

# LIST OF FIGURES

Figure 1 Xilinx Nexys 3 FPGA Board …………………………………….8

Figure 2 FPGA Architecture…………….…………………………………….8

[Figure 3 Mealy FSM Block Diagram 14](#_Toc105755804)

[Figure 4 State diagram of Mealy machine 15](#_Toc105755805)

[Figure 5 Moore FSM Block Diagram 16](#_Toc105755806)

[Figure 6 State diagram of Moore machine 17](#_Toc105755807)

Figure 7 Logo of ARM Inc…………………..………..17

[Figure 8 APB Block Diagram …………………………..….18](#_Toc105755803)

[Figure 9 APB Operating States 14](#_Toc105755804)

[Figure 10 Output waveforms for READ Operation 16](#_Toc105755806)

[Figure 11 Output waveforms for WRITE Operation 17](#_Toc105755807)

Figure 12 RTL Schematic………………………………………………..………..17

[Figure 13 Post Implementation Schematic…………………………..….18](#_Toc105755803)

[Figure 14 FPGA Design Flow…………………………………………..….18](#_Toc105755803)

Fig. 15 – Device floorplan post Place & Route ……………………..

# LIST OF TABLES

TABLE 1 Post synthesis utilization report…………………………..….19

TABLE 2 Post implementation utilization report …………………..19

**PROBLEM STATEMENT**

Longer development cycle for interfaces as prior to the advent of arm AMBA protocols, there was no standard method through which data transfer took place.

Compatibility issues with different vendors as whenever a need arises for using modules from different vendors, interfacing between them proved to be a hassle.

Bandwidth and Latency restrictions as interfacing different blocks having no common protocol for data transmission resulted in reduction of bandwidth and thus increased the latency.

1. **INTRODUCTION**

APB stands for Advanced Peripheral Bus. It’s an on-chip interconnect specification provided by arm private ltd. A traditional AMBA based SoC uses AHB for specifying high bandwidth interfaces and APB for low bandwidth communication. It's a simple non-pipelined communication protocol which can be connected to a shared bus (like a bridge). It uses same set of control signals to read/write data. It requires a minimum of 9 signals for data transmission. It doesn’t support burst data transfer modes. Every data transfer takes at least two clock cycles (SETUP Cycle and ACCESS Cycle) to complete. It’s designed using Finite State Machine in any of the HDL available.

Despite its disadvantages, it's still used in today's system due to its simple operating principle. The main reason behind it’s use is that using higher complexity interconnects like AHB and AXI for connecting low bandwidth peripherals is overkill and consumes a lot of power unnecessarily. We plan to implement this protocol using Xilinx Nexys 3 FPGA board from Spartan 6 family.

1. **FPGA**

2.1) Introduction

It stands for Field Programmable Gate Array. It is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence the term field programmable. The FPGA configuration is generally specified using a hardware description language (HDL), like that used for an Application Specific Integrated Circuit (ASIC). It contains an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects allowing blocks to be wired together. Logic blocks can be configured to perform complex combinational functions, or act as simple logic gates. In most FPGAs, logic blocks also include memory elements, which may be simple flip flops or more complete blocks of memory. Many FPGAs can be reprogrammed to implement different logic functions, allowing flexible reconfigurable computing as performed in computer software. It has a remarkable role in embedded systems development due to their capability to start system software development simultaneously with hardware, enable system performance simulations at a very early phase of the development, and allow various system trials and design iterations before finalizing the system architecture. These are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. We will be using Xilinx Spartan 6 FPGA board which has 6 Input LUTs, 16 Switches and LEDs, 2 Push buttons, Temperature Sensor LM35, 5V SPDT Relay and a Buzzer.

It is highly customizable hardware down to the firmware and its architecture. It is used for emulation and prototyping for the ASIC due to its high speed and parallel processing capabilities. It is re-programmable due to which debugging is easy. It allows for efficient IP reuse due to which it results in faster time to market.

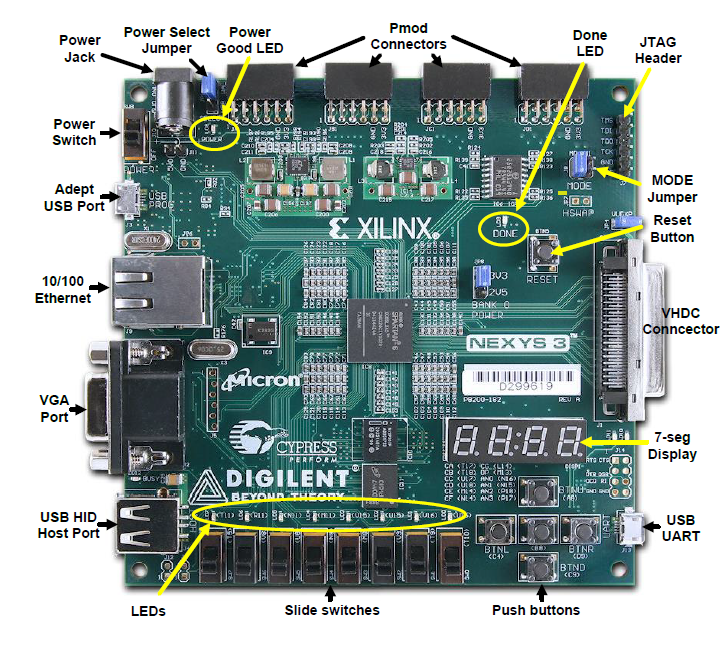


Fig. 1- Xilinx Nexys 3 FPGA Board

2.2) History

The FPGA industry sprouted from programmable read-only memory (PROM) and programmable logic devices (PLDs). PROMs and PLDs both had the option of being programmed in batches in a factory or in the field (field-programmable). Xilinx Inc. produced the first commercially viable field-programmable gate array in 1985 - the XC2064. The XC2064 had programmable gates and programmable interconnects between gates, the beginnings of a new technology and market. It had 64 configurable logic blocks (CLBs), with two three-input look-up tables (LUTs). The 1990s were a period of rapid growth for FPGAs, both in circuit sophistication and the volume of production. In the early 1990s, FPGAs were primarily used in telecommunications and networking. By the end of the decade, FPGAs found their way into consumer, automotive, and industrial applications. By 2013, Altera (31 percent), Actel (10 percent) and Xilinx (36 percent) together represented approximately 77 percent of the FPGA market.

Companies like Microsoft have started to use FPGAs to accelerate high-performance, computationally intensive systems (like the data centres that operate their Bing Search engine, due to the performance per watt advantage that the FPGAs deliver. Microsoft began using FPGAs to accelerate Bing in 2014, and in 2018 began deploying FPGAs across other data centre workloads for their Azure Cloud Platform.

2.3) Components

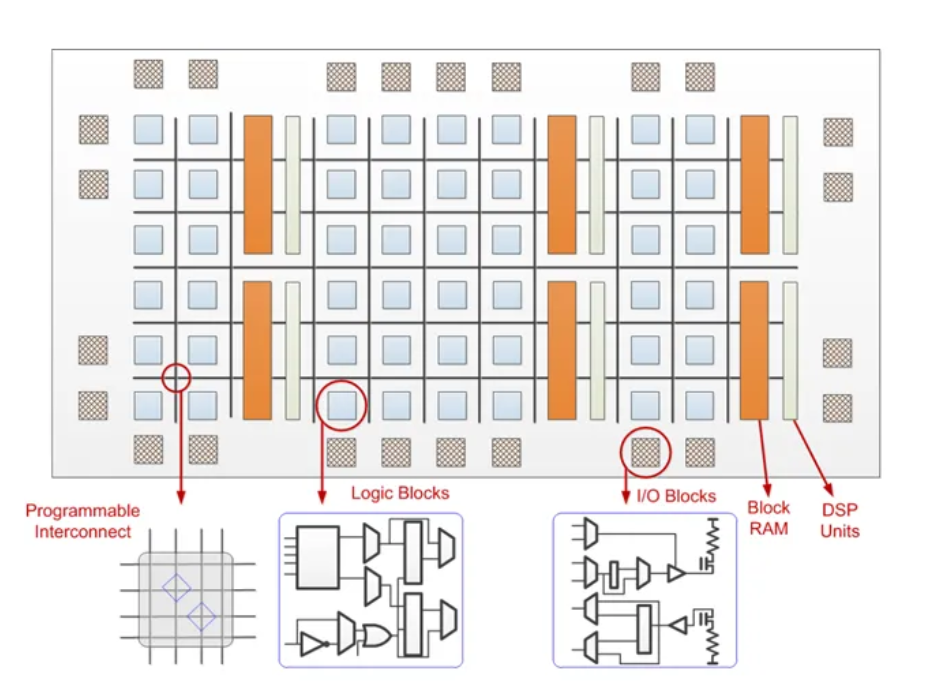


Fig. 2 - FPGA Architecture

* + 1. **Configurable Logic Block (CLB)**

A CLB is the fundamental piece of an FPGA and is what gives it its ability to take on different hardware configurations. An FPGA in its most basic form is a chip of CLBs–together, they make an FPGA. The many thousands of these that can be found on modern FPGAs can be programmed to perform virtually any logic function. An individual CLB consists of several discrete logic components itself, such as look-up tables (LUTs) and flip-flops.

* + 1. **Digital Signal Processing (DSP) Slice**

Referred to as a DSP slice, block, or cell, this is one of the specialized components in an FPGA. It is designed to carry out digital signal processing functions, such as filtering or multiplying, much more efficiently than if the same functions were implemented using many CLBs.

* + 1. **Transceivers**

The name of this component is a mash up of its functionality, as they are made to transmit and receive serial data (individual bits) to and from the FPGA at extremely high rates. The task of converting information on the FPGA into serial data, as well as receiving serial data externally and converting it into useful information, while checking for errors in the data becomes more difficult to do with the configurable logic of the FPGA as speeds increase, eventually hitting a speed cap. Having a dedicated component available for this allows for easy implementation of high-speed data transfer by the user without consuming the logic resources of the FPGA.

* + 1. **Block Random Access Memory (BRAM)**

Memory available on an FPGA board comes in a few flavors, however the dedicated memory on the chip itself is referred to as block RAM or BRAM. While each block individually is of a set size (36K bits for Xilinx 7 series chips), these blocks can be subdivided or cascaded to make smaller or larger sizes of BRAM available. They also are capable of a variety of operational settings and can support special functionality such as error correction.

* + 1. **Input/Output (IO) Blocks**

Input/output blocks are pretty much what they sound like. They are the components through which data transfers in to and out of the FPGA. Input and output on the chip go through component groups called IO banks, which consist of 50 individual IO blocks. The IO blocks themselves are configurable in a few ways depending on the type of data the user is either expecting to receive or transmit. These are like transceivers but operate at lower speeds and can maintain more functional flexibility. A simple analogy to distinguish the two would be to consider having a choice of vehicle between a car (IO block) and a jet (transceiver) for a commute. Even if the distance allowed you could get up to speed to take off in the jet (transceivers have minimum operating speeds), it would be wildly impractical.

2.4) Advantages

It has following advantages over ASIC and other prototyping boards –

* Faster prototyping
* Relatively cheap for the computation power it provides
* Higher number of I/O ports than prototyping boards like Arduino with additional option for presence of expansion slots.
* Wide area of application
* Highly customizable hardware
* Easy to reprogram and debug
* Parallel processing capabilities

2.5) FPGA Board used in the project

We have used Xilinx Nexys 3 FPGA board. The Nexys 3 board is a complete, ready-to-use digital circuit development platform based on the latest Spartan 6 Field Programmable Gate Array (FPGA) from Xilinx. With its high-capacity FPGA (Xilinx part number XC7A35T-1CPG236C), low overall cost, and collection of USB, VGA, and other ports, the Nexys 3 can host designs ranging from introductory combinational circuits to complex sequential circuits like embedded processors and controllers. It includes enough switches, LEDs and other I/O devices to allow a large number designs to be completed without the need for any additional hardware, and enough uncommitted FPGA I/O pins to allow designs to be expanded using Digilent Pmods or other custom boards and circuits, thus providing high performance-per-watt fabric, transceiver line rates, DSP processing, and AMS integration in a cost-optimized FPGA making it perfect for multiple applications like those in the fields of avionics, medical imaging, image processing amongst others.

Some highlighted specifications of the board-

* 2278 slices with each slice contains four 6-input LUTs and 8 flip-flops
* 576 Kbits of block RAM
* 16 MB of cellular RAM
* 5 Clock tiles with 4 DCMs and 2 PLLs
* 32 DSP slices
* Internal clock speeds exceeding 500 Mhz
* 8 user switches
* 8 user LEDs
* 5 user pushbuttons
* 4-digit 7-segment display
* Four Pmod ports
* 12-bit VGA output
* Quad SPI Flash of 16 MB non-volatile memory
* USB UART and USB-HID Host for mice, keyboards and memory sticks

1. **FINITE STATE MACHINE**

3.1) Introduction

A finite-state machine (FSM), or simply a state machine, is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition. An FSM is defined by a list of its states, its initial state, and the inputs that trigger each transition. Finite-state machines are of two types—deterministic finite-state machines and non-deterministic finite-state machines. A deterministic finite-state machine can be constructed equivalent to any non-deterministic one. The behaviour of state machines can be observed in many devices in modern society that perform a predetermined sequence of actions depending on a sequence of events with which they are presented. Simple examples are: vending machines, which dispense products when the proper combination of coins is deposited; elevators, whose sequence of stops is determined by the floors requested by riders; traffic lights, which change sequence when cars are waiting; combination locks, which require the input of a sequence of numbers in the proper order.

The finite-state machine has less computational power than some other models of computation such as the Turing machine. The computational power distinction means there are computational tasks that a Turing machine can do but an FSM cannot. This is because an FSM's memory is limited by the number of states it has. A finite-state machine has the same computational power as a Turing machine that is restricted such that its head may only perform "read" operations, and always has to move from left to right.

3.2) Types of Finite State Machines

The finite state machines are classified into two types such as **Mealy state machine** and **Moore state machine.** Each have their own advantages and limitations.

3.2.1) Mealy Machine

When the outputs depend on the current inputs as well as states, then the FSM can be named to be a mealy state machine. The following diagram is the **mealy state machine block diagram.** The mealy state machine block diagram consists of two parts namely combinational logic as well as memory. The memory in the machine can be used to provide some of the previous outputs as combinational logic inputs.

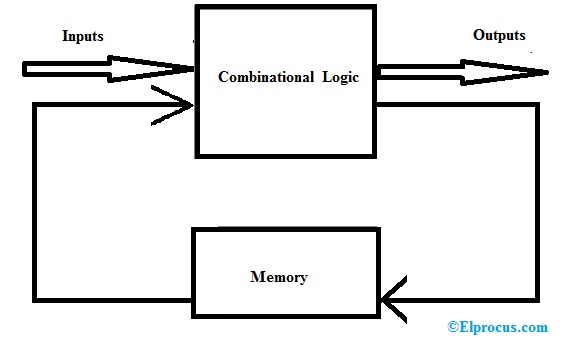


Fig. 3 - Mealy State Machine Block Diagram

Based on the current inputs as well as states, this machine can produce outputs. Thus, the outputs can be suitable only at positive otherwise negative of the CLK signal. The mealy state machine’s state diagram is shown below.

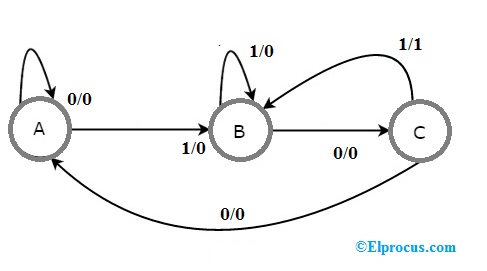


Fig. 4 - State Diagram of Mealy State Machine

The state diagram of mealy state machine mainly includes three states namely A, B, and C. These three states are tagged within the circles as well as every circle communicates with one state. Conversions among these three states are signified by directed lines. In the above diagram, the inputs and outputs are denoted with 0/0, 1/0, and 1/1. Based on the input value, there are two conversions from every state.

Generally, the number of required states in the mealy machine is below or equivalent to the number of required states in Moore state machine. There is an equal Moore state machine for every Mealy state machine. As a result, based on the necessity we can employ one of them.

3.2.1) Moore Machine

When the outputs depend on current states then the FSM can be named as Moore state machine. The Moore state machine’s block diagram is shown below. The Moore state machine block diagram consists of two parts namely combinational logic as well as memory.

In this case, the current inputs, as well as current states, will decide the next states. Thus, depending on further states, this machine will generate the outputs. So, the outputs of this will be applicable simply after the conversion of the state.

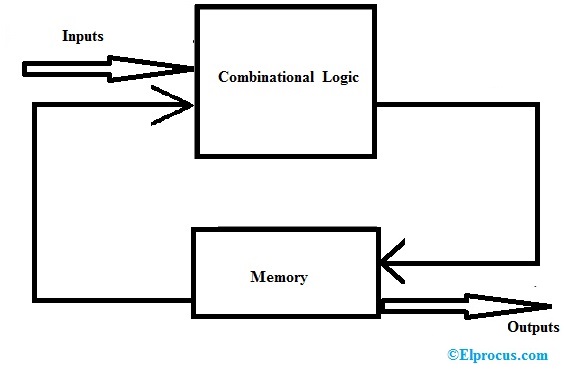


Fig. 5 - Moore State Machine Block Diagram

The Moore state machine state diagram is shown below. In the above state, the diagram includes four states like a mealy state machine namely A, B, C, and D. the four states as well as individual outputs are placed in the circles.

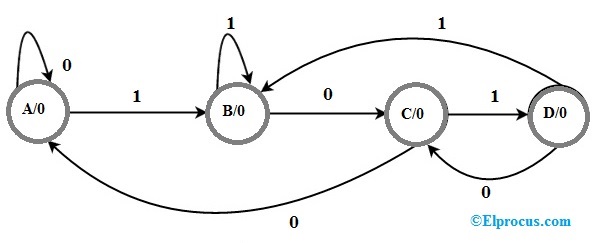


Fig. 6 - State Diagram of Moore State Machine

In the above figure, there are four states, namely A, B, C & D. These states and the respective outputs are labelled inside the circles. Here, simply the input worth is marked on every conversion. In the above figure includes two conversions from every state depending on the input value.

Generally, the number of required states in this machine is greater than otherwise equivalent to the required number of states in the mealy state machine.

3.3) Advantages of Finite State Machines

* Finite state machines are flexible
* Easy to move from a significant abstract to a code execution
* Low processor overhead
* Easy determination of reachability of a state

3.4) Disadvantages of Finite State Machines

* The expected character of deterministic finite state machines can be not needed in some areas like computer games.
* The implementation of huge systems using FSM is hard for managing without any idea of design.
* Not applicable for all domains
* The orders of state conversions are inflexible.

1. **VERILOG (HDL)**

Verilog HDL is a hardware description language used for modelling digital systems at various abstract design levels from algorithm level, gate level to switch level. The complexity of the digital system object being modelled can be between a simple gate and a complete electronic digital system. Digital systems can be described hierarchically, and timing modelling can be explicitly performed in the same description.

 Verilog HDL language has the following description capabilities: design behaviour characteristics, design data flow characteristics, design structure composition, and delay and waveform generation mechanisms including response monitoring and design verification. All of them use the same modelling language. In addition, Verilog HDL language provides a programming language interface through which the design can be accessed from outside the design during simulation and verification, including the specific control and operation of the simulation.

Verilog HDL language not only defines the grammar, but also defines clear simulation and simulation semantics for each grammatical structure. Therefore, models written in this language can be verified using Verilog simulators. The language inherits various operators and structures from the C programming language. Verilog HDL provides extended modelling capabilities, many of which are difficult to understand at first. However, the core subset of Verilog HDL language is very easy to learn and use, which is sufficient for most modelling applications. Of course, a complete hardware description language is sufficient to describe from the most complex chip to a complete electronic system.

It was standardized as IEEE 1364 language used to model electronic systems. It is most commonly used in the design and verification of digital circuits at the register-transfer level of abstraction.It is also used in the verification of analog circuits and mixed-signal circuits, as well as in the design of genetic circuits. In 2009, the Verilog standard (IEEE 1364-2005) was merged into the SystemVerilog standard, creating IEEE Standard 1800-2009. Since then, Verilog is officially part of the SystemVerilog language.

Verilog was created by Prabhu Goel, Phil Moorby, Chi-Lai Huang, and Douglas Warmke between late 1983 and early 1984. Chi-Lai Huang had earlier worked on a hardware description LALSD, a language developed by Professor S.Y.H. Su, for his PhD work. The rights holder for this process, at the time proprietary, was "Automated Integrated Design Systems" (later renamed to Gateway Design Automation in 1985). Gateway Design Automation was purchased by Cadence Design Systems in 1990. Cadence now has full proprietary rights to Gateway's Verilog and the Verilog-XL, the HDL-simulator that would become the de facto standard (of Verilog logic simulators) for the next decade. Originally, Verilog was only intended to describe and allow simulation; the automated synthesis of subsets of the language to physically realizable structures (gates etc.) was developed after the language had achieved widespread usage. Verilog is a portmanteau of the words "verification" and "logic".

1. **APB PROTOCOL**

5.1) Introduction

It stands for Advanced Peripheral Bus. It is from the AMBA Interconnect Protocol. A traditional AMBA based SoC uses AHB for specifying high bandwidth interfaces and APB for low bandwidth communication. It's a simple non-pipelined communication protocol which can be connected to a shared bus ( like a bridge). It uses same set of control signals to read/write data. Burst data transfer modes aren't supported. Every data transfer takes at least two clock cycles (SETUP Cycle and ACCESS Cycle) to complete. In a SoC, AHB to APB bridge acts as the Master for APB while Peripherals are the slaves. Despite its disadvantages, it's still used in today's system due to its simple operating principle.

5.2) AMBA interconnect specification

Advanced Microcontroller Bus Architecture (AMBA) is a freely available, open standard for the connection and management of functional blocks in a System-on-Chip (SoC). The architecture facilitates right-first-time development of multi-processor designs, with large numbers of controllers and peripherals. AMBA specifications are royalty-free, platform-independent, and can be used with any processor architecture. Due to its widespread adoption, AMBA has a robust ecosystem of partners that ensures compatibility and scalability between IP components from different design teams and vendors.



Fig. 7 – Logo of Arm Inc.

AMBA was introduced by ARM in 1996. The first AMBA buses were the Advanced System Bus (ASB) and the Advanced Peripheral Bus (APB). In its second version, AMBA 2 in 1999, ARM added AMBA High-performance Bus (AHB) that is a single clock-edge protocol. These protocols are today the de facto standard for embedded processor bus architectures because they are well documented and can be used without royalties.

5.3) Need for APB

It is a synchronous circuit that has a simple design of circuit having high optimized design. This protocol is used for on-chip communication that is used for intra chip communication between modules. It is a low power design sufficient for connecting low bandwidth peripherals having up to 32 bits. It is a low latency interconnect specification having no fixed frequency for the clock. The clock frequency is totally dependent on the design. If the timing can be matched for the design, protocol will work based on that. It is used for connecting peripherals like Timer, Keypad, UART, I2C etc. Without this, there’ll be crisis for accessing the ports as the peripheral devices need to connect directly to the main core. It also adds the functionality of increased reusability and aids in an easy maintenance.

5.4) Signal Description

It consists of a lot of signals for data transfer-

1. PCLK

Since the design is synchronous, it is based of a clock. This signal provides the clock based on which at posedge all the necessary transactions take place.

1. PRESETn

This signal is the active high reset signal which directs everything to its default state. It is used to initialize the transaction and to reset it whenever needed.

1. PSELx

This contains acts as the selection line for the connecting the master to slaves through a multiplexor. Here x represents the number of slaves that are interfaced with the master.

1. PENABLE

This signal provides the enable signal for subsequent cycles.

1. PADDR

This is a bus which provides the address where the data transfer is to take place. This is same bus for both read/write operations.

1. PWRITE

This is a signal that defines the nature of operation to be performed. If its value is 0, it stands for READ operation, else it is a WRITE operation.

1. PWDATA

This is a bus that writes the data to the slaves upon reaching certain state.

1. PRDATA

This is the bus that reads the data from the slaves as per the command from the master.

1. PREADY

This signal marks if the slave is ready for transaction to take place or not. If this signal is de-asserted, it marks that the slave is not ready and the master will stay in access state until this signal is asserted.

1. transfer

This is a user input signal that marks the beginning of a data transfer. Without assertion of this signal, APB will always remain in IDLE state.

1. readaddr

This bus is user input and provides address upon which read operation is to be performed.

1. writeaddr

This bus is user input and provides address upon which write operation is to be performed.

1. datatowrite

This bus is user input which provides the data that is to be written at writeaddr field.

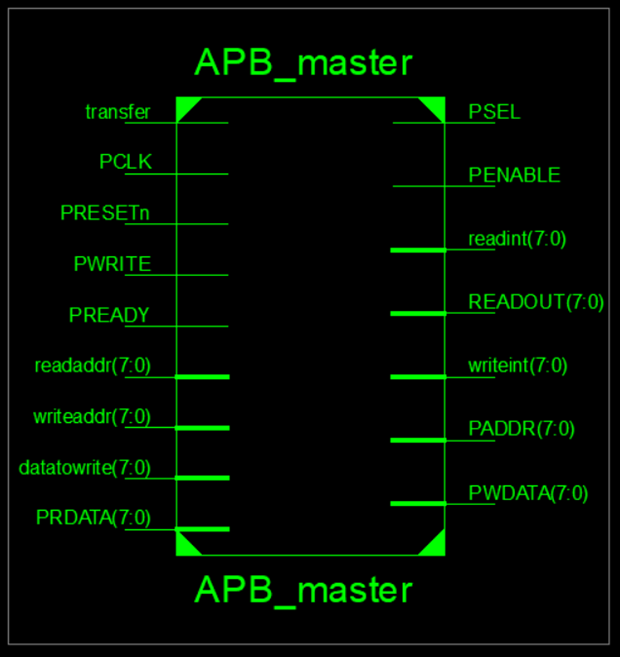
****

Fig. 8 – APB Block diagram

5.5) Operating States

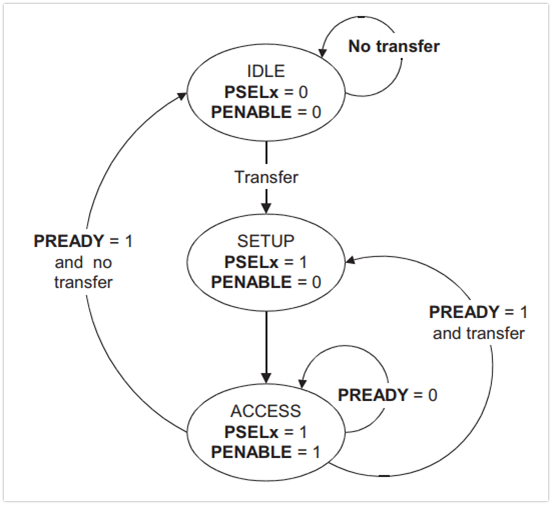


Fig. 9 - APB Operating states

It has three operating states namely IDLE, SETUP and ACCESS.

1. IDLE

This is the default state of the APB interface. PSELx and PENABLE are both de-asserted in this state. It marks no transfer of data. In case of timeout or reset signal activation, the state machine reverts to this state.

1. SETUP

When a transfer is required, transfer signal is asserted. It changes to SETUP state in which PSELx is asserted. The interface only remains in the SETUP state for one clock cycle and always moves to the ACCESS state on the next rising edge of the clock. In this state, address is loaded into the PADDR bus wherever operation is to be performed based on the PWRITE signal.

1. ACCESS

PENABLE is asserted in the ACCESS state. Exit from this state is controlled by the PREADYsignal. Till PREADY is asserted, the system stays in ACCESS state. After completion of transaction, based on the value of the transfer (marking more data transfers after the present one), next state is changed to either SETUP or IDLE. This is the state in which actual operation takes place whether it’s a read operation or a write operation.

5.6) Output waveforms

**READ Operation**

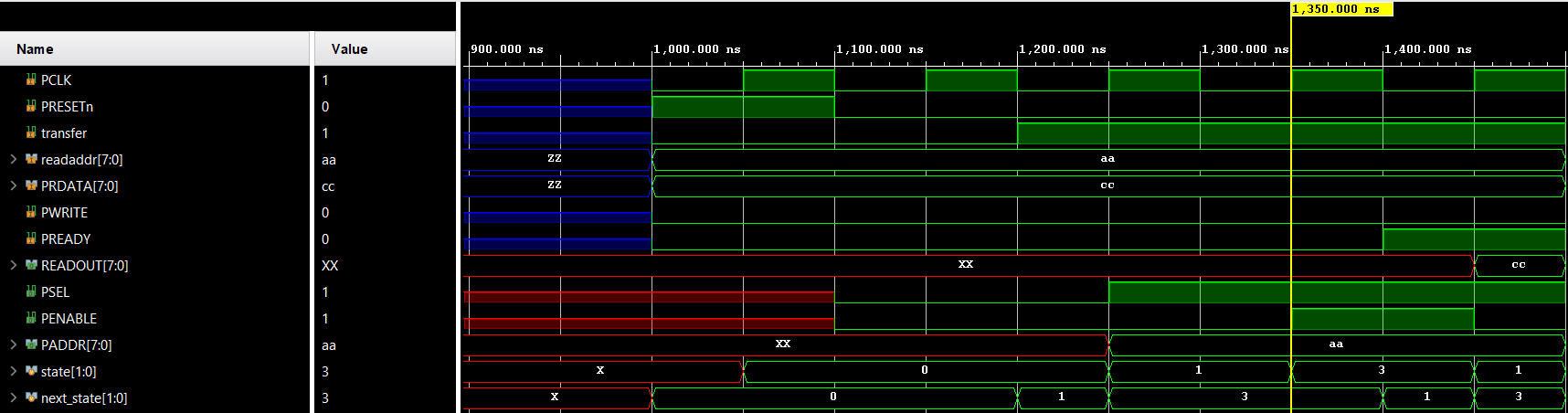


Fig. 10 - Output waveforms for READ Operation

It consists of initialization of state from IDLE state in which PSEL and PENABLE are both zero. After this in SETUP state, PSEL is asserted. Address is loaded into PADDR in this state. At last the system reaches to ACCESS state and based on the value of the PREADY, data transfer is complete and value is assigned to READOUT bus signal.

**WRITE Operation**

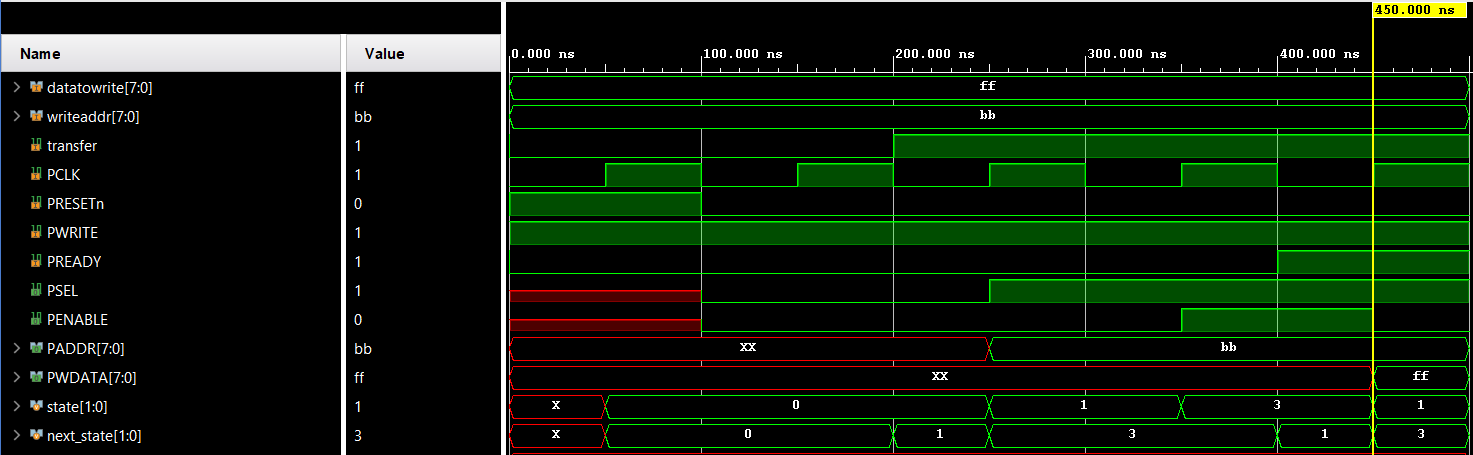


Fig. 11 - Output waveforms for WRITE Operation

It consists of initialization of state from IDLE state in which PSEL and PENABLE are both zero. After this in SETUP state, PSEL is asserted. Address is loaded into PADDR in this state. At last the system reaches to ACCESS state and based on the value of the PREADY, data transfer is complete and value from datatowrite signal is assigned to PWDATA bus signal.

1. **SCHEMATIC**

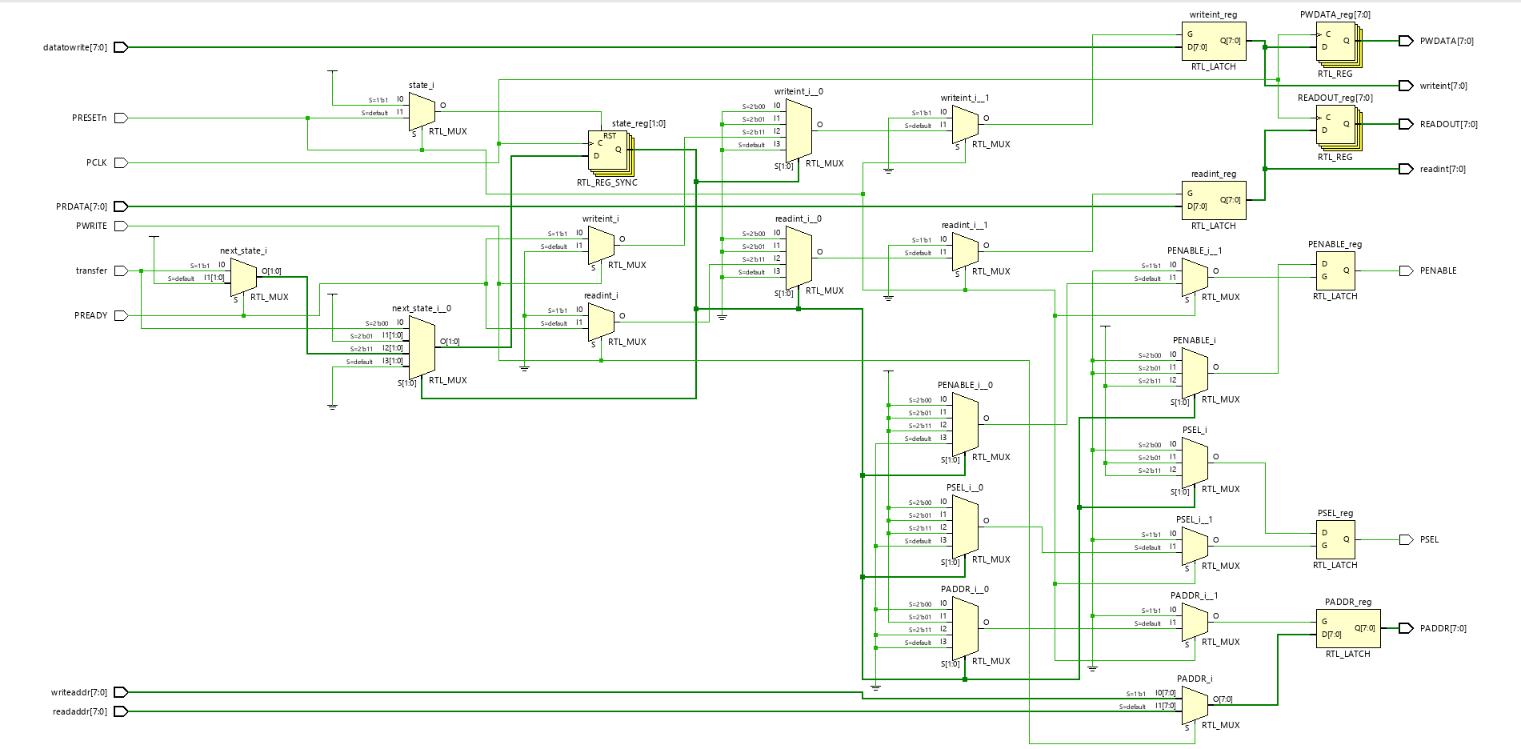
****

Fig. 12 - RTL Schematic

The figure shows Register Transfer Level Schematic prior to synthesis and implementation showing design using D Flip Flop, Multiplexer and consists of 49 cells, 79 IO ports and 110 nets.

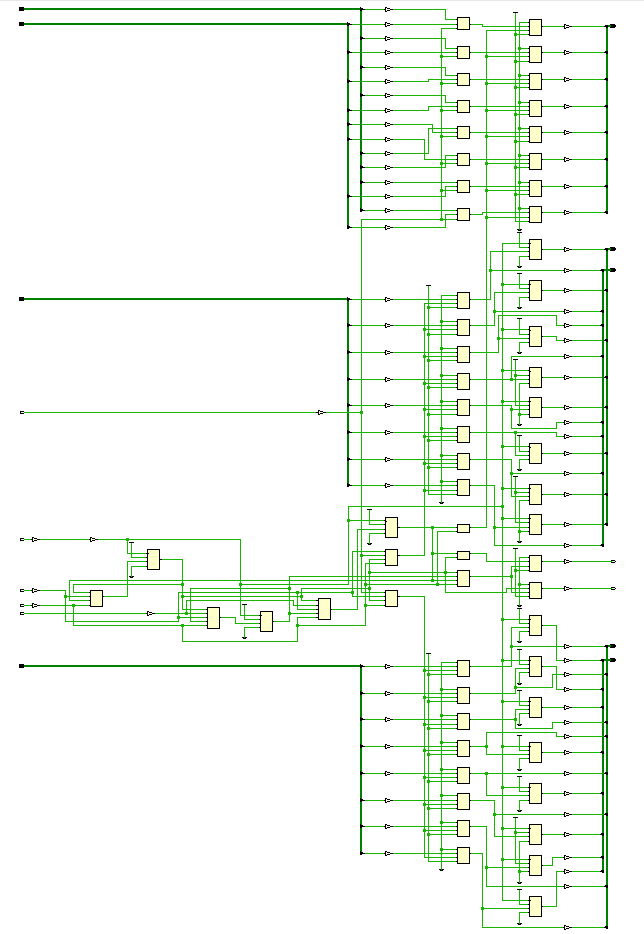


Fig. 13 - Post Implementation Schematic

This schematic provides a post implementation schematic of the system containing 141 cells, 79 IO ports and 182 nets.

1. **REPORTS GENERATED**

These reports are targeted based on Xilinx Nexys 3 FPGA board. It consists of dual registers 6 input LUTs.

7.1) Post synthesis utilization

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S. No. | Site type | Used | Available | Utilization % |
| 1 | Slice LUTs | 8 | 134600 | <0.01 |
| 2 | Slice registers | 45 | 269200 | <0.01 |
| 3 | Bonded IOB | 79 | 285 | 27.72 |
| 4 | Clocking | 1 | 32 | 3.13 |

Table 1 – Post synthesis utilization report

7.2) Post implementation utilization

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S. No. | Site type | Used | Available | Utilization % |
| 1 | Slice LUTs | 8 | 133800 | <0.01 |
| 2 | Slice registers | 45 | 269200 | <0.01 |
| 3 | Bonded IOB | 79 | 285 | 27.72 |
| 4 | Clocking | 1 | 32 | 3.13 |

Table 2 – Post implementation utilization report

7.3) Timing reports

All the timings are in nanoseconds and the max time taken is highlighted in each case.

|  |
| --- |
| **Setup/Hold to clock PCLK** |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Source** | **Max Setup to clk (edge)** | **Process Corner** | **Max Hold to clk (edge)** | **Process Corner** | **Internal Clock(s)** | | PREADY | 0.563(R) | FAST | -0.094(R) | SLOW | PCLK\_BUFGP | | transfer | 0.618(R) | FAST | -0.177(R) | SLOW | PCLK\_BUFGP | |
| |  | | --- | | **Setup/Hold to clock PRESETn** | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Source** | **Max Setup to clk (edge)** | **Process Corner** | **Max Hold to clk (edge)** | **Process Corner** | | PRDATA<0> | 0.247(R) | SLOW | 3.162(R) | FAST | | PRDATA<1> | 0.813(R) | SLOW | 3.162(R) | FAST | | PRDATA<2> | -0.681(R) | SLOW | 3.543(R) | FAST | | PRDATA<3> | -0.195(R) | SLOW | 3.543(R) | FAST | | PRDATA<4> | -0.484(R) | SLOW | 3.604(R) | FAST | | PRDATA<5> | -0.515(R) | SLOW | 3.604(R) | FAST | | PRDATA<6> | -0.855(R) | SLOW | 3.604(R) | FAST | | PRDATA<7> | -0.847(R) | SLOW | 3.604(R) | FAST | | PREADY | 3.792(R) | SLOW | -0.269(R) | SLOW | | PWRITE | 3.694(R) | SLOW | -0.171(R) | SLOW | | datatowrite<0> | 1.549(R) | SLOW | 3.101(R) | FAST | | datatowrite<1> | 1.531(R) | SLOW | 3.162(R) | FAST | | datatowrite<2> | 0.857(R) | SLOW | 3.543(R) | FAST | | datatowrite<3> | 0.287(R) | SLOW | 3.543(R) | FAST | | datatowrite<4> | 1.033(R) | SLOW | 3.543(R) | FAST | | datatowrite<5> | 0.857(R) | SLOW | 3.543(R) | FAST | | datatowrite<6> | 1.228(R) | SLOW | 3.604(R) | FAST | | datatowrite<7> | 1.435(R) | SLOW | 3.604(R) | FAST | | |
| |  | | --- | | **Clock PCLK to Pad** | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Destination** | **Max (slowest) clk (edge) to PAD** | **Process Corner** | **Min (fastest) clk (edge) to PAD** | **Process Corner** | | PWDATA<0> | 7.806(R) | SLOW | 5.618(R) | FAST | | PWDATA<1> | 7.806(R) | SLOW | 5.618(R) | FAST | | PWDATA<2> | 7.426(R) | SLOW | 5.260(R) | FAST | | PWDATA<3> | 7.426(R) | SLOW | 5.260(R) | FAST | | PWDATA<4> | 7.426(R) | SLOW | 5.260(R) | FAST | | PWDATA<5> | 7.426(R) | SLOW | 5.260(R) | FAST | | PWDATA<6> | 7.426(R) | SLOW | 5.260(R) | FAST | | PWDATA<7> | 7.426(R) | SLOW | 5.260(R) | FAST | | READOUT<0> | 7.806(R) | SLOW | 5.618(R) | FAST | | READOUT<1> | 7.806(R) | SLOW | 5.618(R) | FAST | | READOUT<2> | 7.426(R) | SLOW | 5.260(R) | FAST | | READOUT<3> | 7.426(R) | SLOW | 5.260(R) | FAST | | READOUT<4> | 7.426(R) | SLOW | 5.260(R) | FAST | | READOUT<5> | 7.426(R) | SLOW | 5.260(R) | FAST | | READOUT<6> | 7.426(R) | SLOW | 5.260(R) | FAST | | READOUT<7> | 7.426(R) | SLOW | 5.260(R) | FAST | | |

|  |
| --- |
| **Clock PRESETn to Pad** |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Destination** | **Max (slowest) clk (edge) to PAD** | **Process Corner** | **Min (fastest) clk (edge) to PAD** | **Process Corner** | | PENABLE | 9.077(R) | SLOW | 7.244(R) | FAST | | PSEL | 9.077(R) | SLOW | 7.244(R) | FAST | | readint<0> | 8.697(R) | SLOW | 6.886(R) | FAST | | readint<1> | 8.697(R) | SLOW | 6.886(R) | FAST | | readint<2> | 8.697(R) | SLOW | 6.886(R) | FAST | | readint<3> | 8.697(R) | SLOW | 6.886(R) | FAST | | readint<4> | 8.697(R) | SLOW | 6.886(R) | FAST | | readint<5> | 8.697(R) | SLOW | 6.886(R) | FAST | | readint<6> | 8.697(R) | SLOW | 6.886(R) | FAST | | readint<7> | 8.697(R) | SLOW | 6.886(R) | FAST | | writeint<0> | 8.697(R) | SLOW | 6.886(R) | FAST | | writeint<1> | 8.697(R) | SLOW | 6.886(R) | FAST | | writeint<2> | 8.697(R) | SLOW | 6.886(R) | FAST | | writeint<3> | 8.697(R) | SLOW | 6.886(R) | FAST | | writeint<4> | 8.697(R) | SLOW | 6.886(R) | FAST | | writeint<5> | 8.697(R) | SLOW | 6.886(R) | FAST | | writeint<6> | 8.697(R) | SLOW | 6.886(R) | FAST | | writeint<7> | 8.697(R) | SLOW | 6.886(R) | FAST | |

|  |
| --- |
| **Clock to Setup on destination clock PCLK** |
| |  |  |  | | --- | --- | --- | | **Source Clock** | **Src:Rise Dest:Rise** | **Src:Fall Dest:Rise** | | PCLK | 1.272 |  | | PRESETn | 6.887 | 0.763 | |

|  |
| --- |
| **Clock to Setup on destination clock PRESETn** |
| |  |  |  | | --- | --- | --- | | **Source Clock** | **Src:Rise Dest:Rise** | **Src:Fall Dest:Rise** | | PCLK | 4.978 |  | |

Total On-chip power = 11.078 Watt

Junction Temperature = 52.3 °C

1. **DESIGN FLOW FOR THE PROJECT**

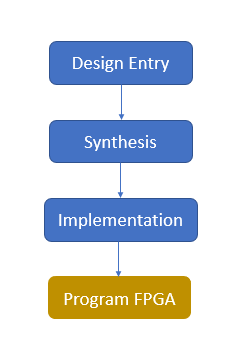


Fig. 14 – Block diagram of FPGA Design flow

8.1) Design entry

This is the stage where based on specifications, design is built keeping in mind it should be synthesizable. This design is tested against stimulus and upon observing correct behavior, it’s verified.

Here’s my code for APB Master –

module APB\_master (

input transfer,

input [7:0] readaddr,writeaddr,

input [7:0] datatowrite,

input PCLK,

input PRESETn,

input [7:0] PRDATA, //reading from slave

input PWRITE, //0 is READ 1 is WRITE

input PREADY, //coming from slave

output reg [7:0] readint, //READ INTERMEDIATE

output reg [7:0] READOUT, //FINAL READOUT

output reg [7:0] writeint,//WRITE INTERMEDIATE

output reg PSEL,

output reg PENABLE,

output reg [7:0]PADDR,//read or write address

output reg [7:0]PWDATA //FINAL WRITE

);

reg [1:0] state, next\_state;

parameter IDLE = 2'b00, SETUP = 2'b01, ACCESS = 2'b11;

assign readaddr=8'b00000010;

assign writeaddr=8'b00000010;

always @(posedge PCLK)

begin

if(PRESETn)

state <= IDLE;

else

state <= next\_state;

end

always@(\*)

begin

if (PRESETn) //reset

next\_state <= IDLE;

else

begin

case(state)

IDLE:

begin

PSEL<=0;

PENABLE<=0;

if(transfer)

next\_state<= SETUP;

else

next\_state<=IDLE;

end

SETUP:

begin

PSEL<=1;

PENABLE<=0;

if (PWRITE) // WRITE OP

begin

PADDR<=8'b00000010;

next\_state<=ACCESS;

end

else // READ OP

begin

PADDR<=8'b00000010;

next\_state<=ACCESS;

end

end

ACCESS:

begin

PSEL<=1;

PENABLE<=1;

if (PWRITE) //WRITE OP

begin

if(PREADY) //slave ready

begin

writeint<=datatowrite;

if (transfer) //more op after this

next\_state<=SETUP;

else

next\_state<=IDLE;

end

else //slave not ready for write

next\_state<=ACCESS;

end

else //READ OP

begin

if (PREADY) //slave ready

begin

readint<=PRDATA;

if(transfer) //more op after this

next\_state<=SETUP;

else

next\_state<=IDLE;

end

else //slave not ready for read

next\_state<=ACCESS;

end

end

default:next\_state<=IDLE;

endcase

end

end

always@(posedge PCLK)

begin

PWDATA<=writeint;

READOUT<=readint;

end

endmodule

8.2) Synthesis

In this stage, HDL code is converted into a gate level netlist. This is why it’s important in the design entry to keep the entire HDL code synthesizable. This stage takes a constraint file or a User Constraint File (UCF) having an extension of .xdc or .ucf depending on the generation of FPGA board used. This file is used to specify the interconnection between FPGA slices and design ports. This mapping defines the connection between physical properties of board to design ports like mapping FPGA switches to defined ports. I have used 8 switches and 8 LEDs keeping my clock signal as the block on-chip clock. It also checks the syntax of the design file.

Here’s my constraint file –

# This file is a general.ucf for Nexys3 rev B board

## To use it in a project:

## - remove or comment the lines corresponding to unused pins

## - rename the used signals according to the project

## Clock signal

NET "PCLK" LOC = "V10" | IOSTANDARD = "LVCMOS33";

## Leds

NET "READOUT[0]" LOC = "U16" | IOSTANDARD = "LVCMOS33";

NET "READOUT[1]" LOC = "V16" | IOSTANDARD = "LVCMOS33";

NET "PWDATA[0]" LOC = "U15" | IOSTANDARD = "LVCMOS33";

NET "PWDATA[1]" LOC = "V15" | IOSTANDARD = "LVCMOS33";

NET "PSEL" LOC = "M11" | IOSTANDARD = "LVCMOS33";

NET "PENABLE" LOC = "N11" | IOSTANDARD = "LVCMOS33";

NET "PADDR[0]" LOC = "R11" | IOSTANDARD = "LVCMOS33";

NET "PADDR[1]" LOC = "T11" | IOSTANDARD = "LVCMOS33";

## Switches

NET "datatowrite[0]" LOC = "T10" | IOSTANDARD = "LVCMOS33";

NET "datatowrite[1]" LOC = "T9" | IOSTANDARD = "LVCMOS33";

NET "PWRITE" LOC = "V9" | IOSTANDARD = "LVCMOS33";

NET "PREADY" LOC = "M8" | IOSTANDARD = "LVCMOS33";

NET "transfer" LOC = "N8" | IOSTANDARD = "LVCMOS33";

NET "PRDATA[0]" LOC = "U8" | IOSTANDARD = "LVCMOS33";

NET "PRDATA[1]" LOC = "V8" | IOSTANDARD = "LVCMOS33";

NET "PRESETn" LOC = "T5" | IOSTANDARD = "LVCMOS33";

NET "PRESETn" CLOCK\_DEDICATED\_ROUTE = FALSE;

8.3) Implementation

In this stage, the synthesized design is translated into transistors and then mapped according to the circuit specifications of the design. This stage also does mapping, placing and routing of FPGA interconnects with design ports. This stage also utilized constraint file. After this stage, timing report can also be generated. After this stage, all the reports, schematic and reports that are generated are most accurate. Here is the design after implementation.

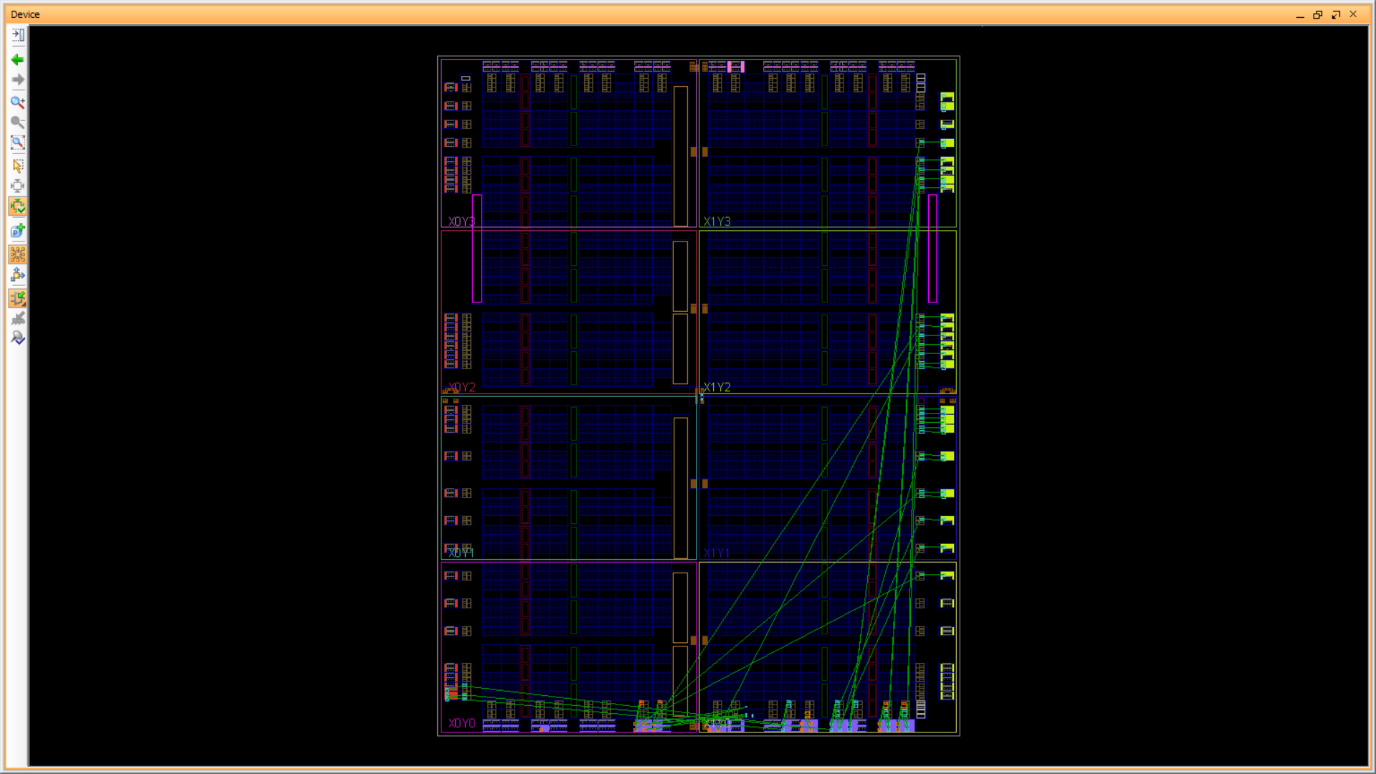


Fig. 15 – Device floorplan post Place & Route

8.4) Program FPGA

Following successful implementation and having met all constraints, the bitstream file can be generated which has the extension of <name>.bit . It gives the entire information of design source and constraint to a single file which also helps in reusability and transfer of data. This bitstream file when dumped into an FPGA board gives it all the necessary port mapping and design can be emulated using FPGA board.

1. **DESIGN CHALLENGES AND LIMITATIONS**

Firstly, it has some limitations like it’s non-pipelined which limits it’s prefetching capabilities. Adding to that, it can only have a single master which restricts the parallelism supported by modern day cores. It also suffers degrade in performance as bus gets loaded more. These limitations are overcome by newer generations interconnect protocols from the same family like AHB or AXI.

Creating the design for the APB Master, we have faced some design challenges like –

1. Maintaining the synchronous nature of design with such large interdependency of signals such that only on active edges of clock, values are derived.
2. Maintaining a clear difference between write and read operations and their directions.
3. Reducing the number of I/O to match the availability of switches and LEDs such that all the behavior can be displayed.
4. Hardcoding the values of Address for both write and read operations to reduce the number of switches used.
5. Adding intermediate signals, written in lowercase letters, to better specify the intention of operation and obtain better clarity.
6. Designing only master module as Design source and making FPGA inputs as User Input testbench to aid the emulation.
7. Defining the Finite State Machine allowed in the specification for AMBA APB version 2.0 by arm.
8. **CONCLUSION**

APB Protocol is very important protocol which allows for efficient and speed data transfer from peripheral devices to the main core/master. This protocol is clock frequency flexible and facilitates the working of other protocols like SPI, UART and I2C.

**REFERENCES**

[1] ARM, “AMBA Specification Overview”, <http://www.arm.com/>

[2] APB Example-AMBA system, technical reference manual, ARM Inc.,1999.

[3] Akhilesh Kumar, Richa Sinha, “Design and Verification analysis of APB3 Protocol with Coverage,” IJAET, Nov 2011.

[4] Santhi Priya Sarekokku, K. Rajasekhar, “Design and Implementation of APB Bridge based on AMBA

AXI 4.0,” IJERT, Vol.1, Issue 9, Nov 2012.

[5] VERILOG Reference Manual, <http://www.accellera.com>

[6] Samir Palnitkar, “Verilog HDL: A guide to Digital Design and Synthesis (2nd Edition), Pearson, 2008.

[7] Flynn, D. Adv. RISC Machines Ltd., Cambridge, “*AMBA:* enablingreusableon-chip designs”, IEEE Micro, Aug 1997.