IMPLEMENTATION AND OPTIMIZATION OF PCI TARGET DEVICE

Thesis submitted in the partial fulfillment of requirement for the award of degree of

Master of Technology

in

VLSI Design

Submitted by:

ABHINAV WADHWA

Roll No : 601061001

Under the guidance of:

Ms. MANU BANSAL

Assistant Professor

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

(Established under the section 3 of UGC Act, 1956)

PATIALA – 147004 (PUNJAB)

DECLARATION

I, Abhinav Wadhwa, hereby certify that the work which is being presented in this thesis entitled "IMPLEMENTATION AND OPTIMIZATION OF PCI TARGET DEVICE" by me in partial fulfillment of the requirements for the award of degree of Master of Technology in VLSI Design from Thapar University (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of "Ms. Manu Bansal".

The matter presented in this thesis has not been submitted in any other University / Institute for the award of any other degree.

Date: 22nd June, 2012

Abhinav Wadhwa
Roll No. 601061001

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

Date: 22nd June, 2012

Ms. Manu Bansal
Assistant Professor
ECED

Countersigned by:

Professor and Head ECED
Thapar University, Patiala
Date:

Dean of Academic Affairs
Thapar University, Patiala
Date:

ACKNOWLEDGEMENT

To discover, analyze and to present something new is to venture on an untrodden path towards and unexplored destination is an arduous adventure unless one gets a true torchbearer to show the way. I would have never succeeded in completing my task without the cooperation, encouragement and help provided to me by various people. Words are often too less to reveals one's deep regards. I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this thesis. I acknowledge with gratitude and humility my indebtedness to **Ms. Manu Bansal, Assistant Professor,** Electronics and Communication Engineering Department, Thapar University, Patiala, under whose guidance I had the privilege to complete this thesis. I wish to express my deep gratitude towards her for providing individual guidance and support throughout the thesis work.

I convey my sincere thanks to **Head of the Department, Dr. Rajesh Khanna** as well as **PG Coordinator, Dr. Kulbir Singh, Assistant Professor,** Electronics and Communication Engineering Department, entire faculty and staff of Electronics and Communication Engineering Department for their encouragement and cooperation.

My greatest thanks are to all who wished me success especially my parents. Above all I render my gratitude to the Almighty who bestowed self-confidence, ability and strength in me to complete this work for not letting me down at the time of crisis and showing me the silver lining in the dark clouds. I do not find enough words with which I can express my feelings of thanks to my dear friends for their help, inspiration and moral support which went a long way in successful competition of the present study.

ABSTRACT

Cost is the most important factor taken into consideration while designing a chip, which mainly includes the design cost and manufacturing cost. Manufacturing cost depends upon the number of chips produced on a single wafer, thus it depends upon the size of the chip. Thus to reduce the chip size the designers are constantly looking for various technologies. Among these technologies, the reuse technology is widely adopted. In brief, the reuse is to design the circuit again in such a way that some parts can be repeatedly used in the circuit, and minimize the area of the circuit. In this thesis we have re-reused PCI AD bus and C/BE# bus to communicate with the master.

We have designed two PCT Target Devices, one without implementing re-reuse technology and the other with re-reuse technology. The designs have been simulated and synthesized with 90 nm standard cell library on FPGA using Xilinx ISE and with 180 nm standard cell library on Synopsys Design Compiler.

Synthesis results show that with implementing re-reuse technology on the time sharing basis, the I/O pin count can be reduced by 35 pins. Though we need to implement a slightly complicated logic due to which the design area has increased but as the feature size is decreasing very rapidly, an increase in gate count by few hundreds can be incorporated very easily.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

PCI stands for Peripheral Component Interconnect. The PCI Local Bus is a high performance, 32-bit or 64-bit bus with multiplexed address and data lines. This is a fully synchronous bus with operation speed upto 33MHz or 66MHz. The bus is intended for use as an interconnect mechanism between highly integrated peripheral controller components, peripheral and processor/memory systems. Figure 1.1 shows a possible PCI system.

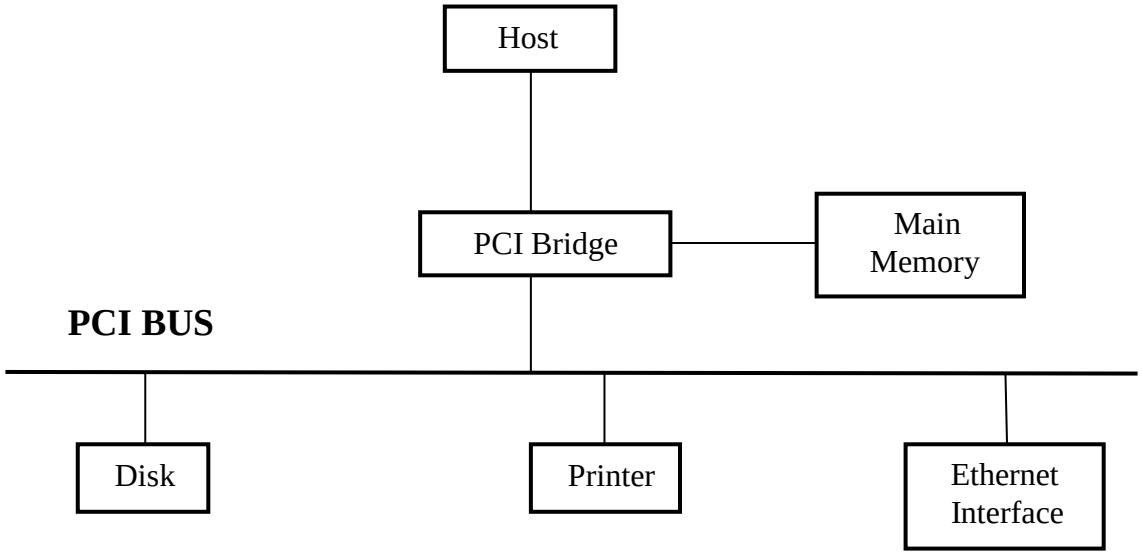


Figure 1.1 PCI System[1]

Master: an agent that initiates transactions on the PCI Bus. It drives commands on the address phase requesting write or read accesses to one of the three address spaces that PCI bus offers (Configuration, I/O, Memory).

Target: It is the slave, which claims and responds to the transaction initiated on the PCI bus by a master agent. Figure 1.2 is the master target device.

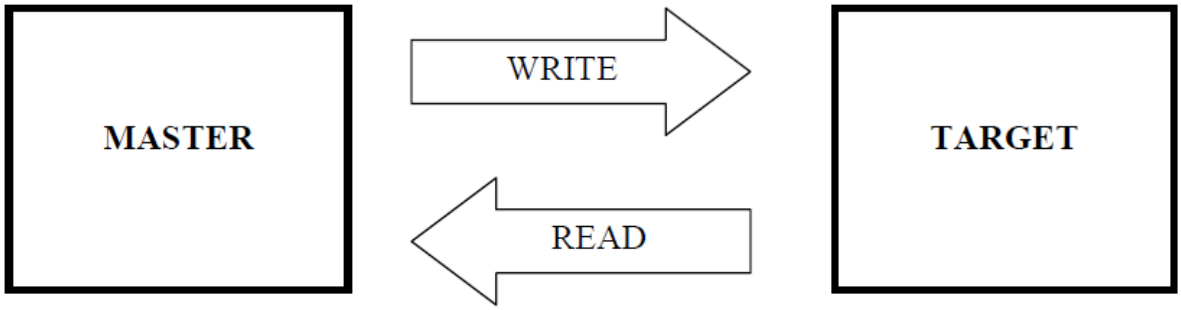


Figure 1.2 Master Target Device

1.1 Motivation

The notion of a computer "bus" evolved in the early 1960s along with the minicomputer. At that time, the minicomputer was a radical departure in computer architecture. Previously, most computers had been one-of-a-kind, custom built machines with relatively few peripherals a paper tape reader and punch, a teletype, a line printer and, if you were lucky, a disk. The peripheral interface logic was tightly coupled to the processor logic.

Graphics-oriented operating systems such as Windows have created a data bottleneck between the processor and its display peripherals in standard PC I/O architectures. Moving peripheral functions with high bandwidth requirements closer to the system's processor bus can eliminate this bottleneck. The advantages offered by local bus designs have motivated several versions of local bus implementations.

[2]First released in 1992, the Peripheral Component Interconnect (PCI) has rapidly evolved into a viable replacement for the ISA bus, and is the most commonly used method for adding boards to a PC. It solves many of the problems with older architectures, while at the same time delivering a substantial increase in processing speed. PCI provides a new way of connecting peripherals to both the system memory and the CPU, with the goal of alleviating many problems encountered when installing new cards in an ISA based system (IRQ conflicts, address conflicts, etc.). To ensure the longevity of the bus, not only are systems based on newer PCI specifications backward compatible with those designed to older ones, PCI boards may also be used in the

system that also uses ISA bus cards--however, this is becoming increasingly rate as ISA is rapidly becoming obsolete.

1.2 Thesis Organization

Chapter 1 gives the introduction about the PCI bus. Chapter 2 gives the description of various pins that a PCI compliant device is required to have. Chapter 3 describes various steps involved in data transfer through PCI. Chapter 4 gives the description about FPGA Design flow. Chapter 5 gives the description about synopsis DC design flow. Chapter 6 provides us the simulation and synthesis results of PCI Architectures. Chapter 7 concludes the thesis work and also provides the future scope.

Chapter 2

PCI SIGNALS

This chapter gives a general description about the features and functional organization of the PCI Core, without describing details but providing an overall view.

2.1 Specifications

- Complies with 32 bit / 33 MHz PCI local bus specifications.
- Support for 32 bit / 33 MHz PCI bus interface for master and target.
- Separated master and target functional blocks.
- Fully synchronous design.
- Single cycle and burst mode support for read and write cycles.
- Parity generation and checking for all data and address phases.

2.2 Supported PCI Features

This is a list of the main features supported by the PCI Core

- 32-bit data bus.
- 33 Mhz Bandwidth (132 Mbytes/sec peak).
- Read and write transactions to the three different PCI address spaces:
    - Memory, I/O and Configuration.
- Master initiated termination:
    - Completion: This is the normal termination, which occurs when the master has completed its transaction.
    - Time-out: It refers to the expiration of the Latency Timer when the master is keeping the bus for many PCI clock periods without the grant from the arbiter.

    - This timer avoids the extension of the access latency of other possible master of the Bus system.
    - Master-Abort: This is the protocol for the termination when no target claims the access initiated by the master.
- Target initiated termination:
    - Retry: This is a particular Disconnect without any data transferred, target is not ready for the transaction and stops it before the first data transfer. The master has to repeat this transaction later.
    - Disconnect: Target has claimed the access, started the data handshake; it wants to terminate the transaction before the Master completion.
    - Target-Abort: It refers to an abnormal end because the target detected a fatal error or it will never be able to complete the request; it can occur at any moment during the first or the subsequent data phases.
- Bus parking. The arbiter can select the default owner of the bus when no agent is currently using or requesting it. This master has to drive the bus lines so that it will not float.
- Parity is generated and possible errors are checked and reported to back-end application.
- Latency requirements: There is a fixed number of PCI clocks within the agent has to provide the next data valid. The PCI Core has internal timers to achieve this function.

2.3 PCI Signals

Figure 2.1 shows the signals defined in PCI. A PCI interface requires a minimum of 47 pins for a target-only device and 49 pins for a master. This is sufficient for a 32-bit data path running at up to 33 MHz and is mandatory for all devices claiming PCI compatibility. An additional 51 pins define optional features such as 64-bit transfers, interrupts and a JTAG interface.

A note about notation:

- A # sign at the end of a signal name, such as FRAME#, indicates that the signal is active or asserted in the low voltage state. Signal names without a # are asserted in the high voltage state.
- The notation [n::m], where n and m are integers such that n is greater than m, represents an "array" of signals with n–m+ 1 members. Thus AD[31::0] represents the 32-bit data bus consisting of signals AD[0] to AD[31] with AD[0] being the least significant bit.

2.3.1 Signal Groups

For purposes of definition, the PCI signals can be classified in several functional groups.

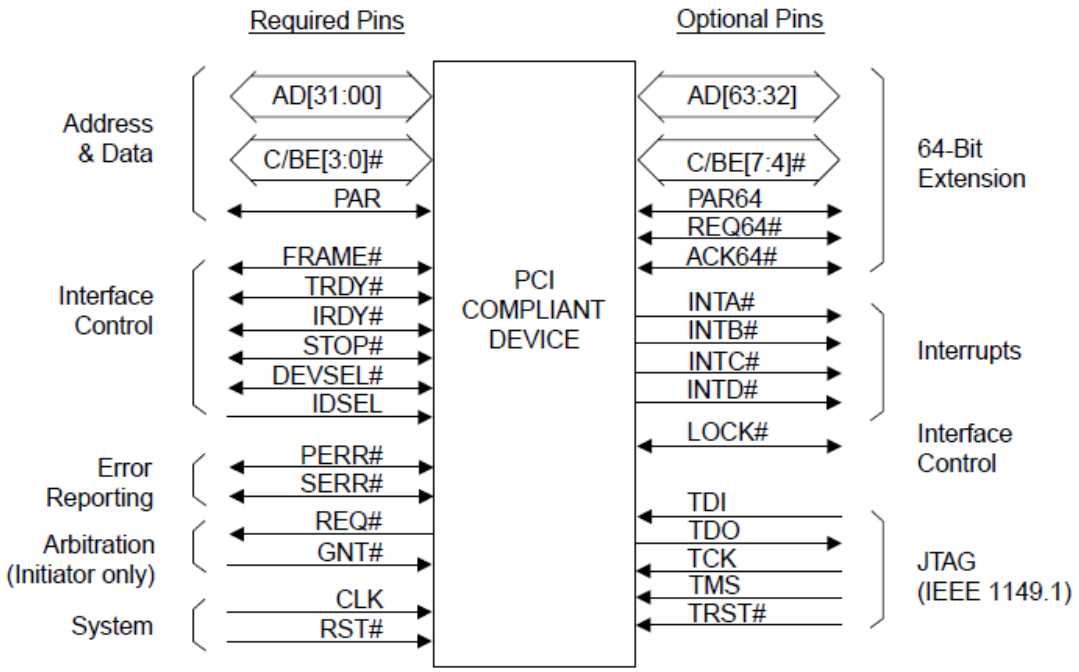


Figure 2.1 PCI pin diagram [3,4]

a) System

I. **CLK** It provides timing for all PCI transactions and is an input to every PCI device. All other PCI signals except RST# and INTA# through INTD# are sampled on the rising edge of CLK. (in)

II. **RST#** It brings PCI-specific registers, sequencers, and signals to a consistent state. Whenever RST# is asserted, all PCI output signals must be driven to their benign state. In general, this means they must be tri-stated. (in)

b) Address and Data

I. **AD[31::0]** Address and data bits are multiplexed on the same set of pins. A PCI transaction consists of an address phase followed by one or more data phases. (t/s)

II. **C/BE[3::0]** Bus command and byte enables are multiplexed on the same pins. During the address phase of a transaction, C/BE[3::0] define a bus command. During each data phase, C/BE[3::0] are used as byte enables to determine which byte lanes carry valid data. C/BE[0] applies to byte 0 (lsb) and C/BE[3] applies to byte 3 (msb). (t/s)

III. **PAR** Even Parity across AD[31::0] and C/BE[3::0]. All PCI agents are required to generate parity. (t/s)

c) Interface Control

I. **FRAME#** Driven by the current master to indicate the beginning and duration of a transaction. Data transfer continues while FRAME# is asserted. When FRAME# is de-asserted, the transaction is in its final data phase. (s/t/s)

II. **IRDY#** Initiator Ready indicates that the bus master is able to complete the current data phase. During a write, IRDY# indicates that valid data is present on AD[31::0] whereas during a read it indicates that the master is prepared to accept data. (s/t/s)

III. **TRDY#** Target Ready indicates that the selected target device is able to complete the current data phase. During a read, TRDY# indicates that valid data is present on AD[31::0] whereas during a write, it indicates that the target is prepared to accept data. A

data phase completes on any clock cycle during which both IRDY# and TRDY# are asserted. (s/t/s)

IV. **STOP#** Indicates that the selected target requests the master to terminate the current transaction. (s/t/s)

V. **LOCK#** Indicates an atomic operation that may require multiple transactions to complete. (s/t/s)

VI. **IDSEL** Initialization Device Select is a chip select used during configuration transactions. (in) **DEVSEL#** Device Select indicates that a device has decoded its address as the target of the current transaction. (s/t/s)

d) Arbitration

I. **REQ#** Request indicates to the central arbiter that an agent desires to use the bus. Every potential bus master has its own point to-point REQ# signal. (t/s)

II. **GNT#** Grant indicates to an agent that is asserting its REQ# signal that access to the bus has been granted. Every potential bus master has its own point-to-point GNT# signal. (t/s)

e) Error Reporting

I. **PERR#** For reporting data Parity Errors during all PCI transactions except a Special Cycle. (s/t/s)

II. **SERR#** System Error is for reporting address parity errors, data parity errors on Special Cycle commands. (o/d)

f) Interrupt (optional)

I. **INTA#** through **INTD#** are used by a device to request attention from its device driver. A single-function device may only use INTA#. Multi-function devices may use any combination of INTx# signals. (o/d)

g) 64-bit Bus Extension (optional)

I. **AD[63::32]** Upper 32 address and data bits. (t/s)
II. **C/BE[7::4]** Upper byte enable signals. They are generally not valid during address phase. (t/s)
III. **REQ64#** Request 64-bit Transfer indicates that the current bus master desires to execute a 64-bit transfer. (s/t/s)
IV. **ACK64#** Acknowledge 64-bit Transfer indicates that the selected target is willing to execute 64-bit transfers. 64-bit transfers can only occur when both REQ64# and ACK64# are asserted. (s/t/s)
V. **PAR64** Even Parity over AD[63::32] and C/BE[7::4]. (t/s)

h) JTAG/Boundary Scan (optional)

The PCI specification reserves a set of pins for implementing a Test Access Port (TAP) conforming to IEEE Standard 1149.1, Test Access Port and Boundary Scan Architecture. This provides a reliable, well-defined mechanism for testing a device or board.

i) Additional Signals

These signals are not part of the basic PCI protocol but implement additional features that are useful in certain operating environments.

I. **PRSNT[1:2]#** These are defined for add-in boards but not for motherboard devices. The Present signals indicate to the motherboard that a board is physically present and, if it is, its total power requirements. All boards are required to ground one or both PRSNT signals as follows: (in)

PRSNT1#	PRSNT2#	State
Open	Open	No expansion board present
Ground	Open	Present, 25 W maximum
Open	Ground	Present, 15 W maximum
Ground	Ground	Present, 7.5 W maximum

Table 2.1 PRSNT# Signals

II. **CLKRUN#** Clock Running is an optional input to a device to determine the state of CLK. It is output by a device that wishes to control the state of the clock. Assertion means the clock is running at its normal speed. De-assertion is a request to slow down or stop the clock. This is intended as a power saving mechanism in mobile environments and is described in the PCI Mobile Design Guide. The standard PCI connector does not have a pin for CLKRUN#. (in, o/d, s/t/s)

III. **M66EN 66MHz_Enable** indicates to a device that the bus segment is running at 66 MHz. (in)

IV. **PME#** Power Management Event is an optional signal that allows a device to request a change in the device or system power state. The operation of this signal is described in the PCI Bus Power Management Interface Specification. (o/d)

V. **3.3Vaux Auxiliary** 3.3 volt Power allows an add-in card to generate power management events even when main power to the card is turned off. The operation of this signal is described in the PCI Bus Power Management Interface Specification. (in)

2.3.2 Signal Types

Each of the signals listed above included a somewhat cryptic set of initials in parentheses. These designate the signal type. The signal types are:

I. in: Input only
   CLK, RST#, IDSEL, TCK, TDI, TMS, TRST#, PRSNT[1:2]#, CLKRUN#, M66EN, 3.3Vaux
II. out: Standard output only
   TDO
III. t/s: Bidirectional tri-state input/output
   AD[63:0], C/BE[7:0], PAR, PAR64, REQ#, GNT#, CLKRUN#
IV. s/t/s: Sustained tri-state. Driven by one owner at a time. Note that all of the s/t/s signals are assertion low. The owner must drive the signal high, that is to the unasserted state, for one clock before tri-stating. Another agent must not drive an s/t/s signal sooner than one clock after the previous owner has tri-stated it. s/t/s signals require a pull-up to sustain the signal in the unasserted state until another agent drives it. The pull-up is provided by the central resource.
   FRAME#, TRDY#, IRDY#, STOP#, LOCK#, PERR#, REQ64#, ACK64#
V. o/d: Open drain, wire-OR allows multiple devices to assert the signal simultaneously. A pull-up is required to sustain the signal in the unasserted state when no device is driving it. The pull-up is provided by the central resource.
   SERR#, INTA# - INTD#, CLKRUN#, PME#

2.3.3 Sideband Signals

The specification acknowledges that there may be a need for application specific signals that fall outside the scope of the PCI specifications. These are called sideband signals and are loosely defined as “ any signal not part of the PCI specifications that connects two or more PCI compliant agents and has meaning only to those agents.” Such signals are allowed provided they don’t interfere with the PCI protocol. No pins are provided on the add-in card connector to support sideband signals so they are restricted to so-called “planar devices” on the motherboard.

Chapter 3

METHODOLOGY

This chapter gives a detailed description about the PCI Target Device architectures without and with the application of re-reuse concept. The term re-reuse is used because AD bus has already been reused for sending data after the address, so we have re-reused to enable the peripheral device to send/receive data from or to the master device. It also explain the state machines involved in achieving the desired task.

3.1 Assumptions

Following are the assumptions taken into account while designing both the architectures

I. Only the target device has been implemented, so all the master operated signals have been manually driven.
II. Another assumption is made regarding generation of memory ready signal. In actual practice the memory controller generates it. Herr it has been modeled as an external pin in the target device (READY) which the user has to drive. This signal in turn drives TRDY.
III. Parity check has not been implemented so the following signals have not been utilized (PAR, PERR#, SERR#).
IV. As only the target is considered, therefore the bus arbiter is not required, hence not designed.

3.2 PCI Target block diagram

Figure 3.1 shows the block view of a PCI Target device. Following are the sub-blocks that comprise a PCI Target device.

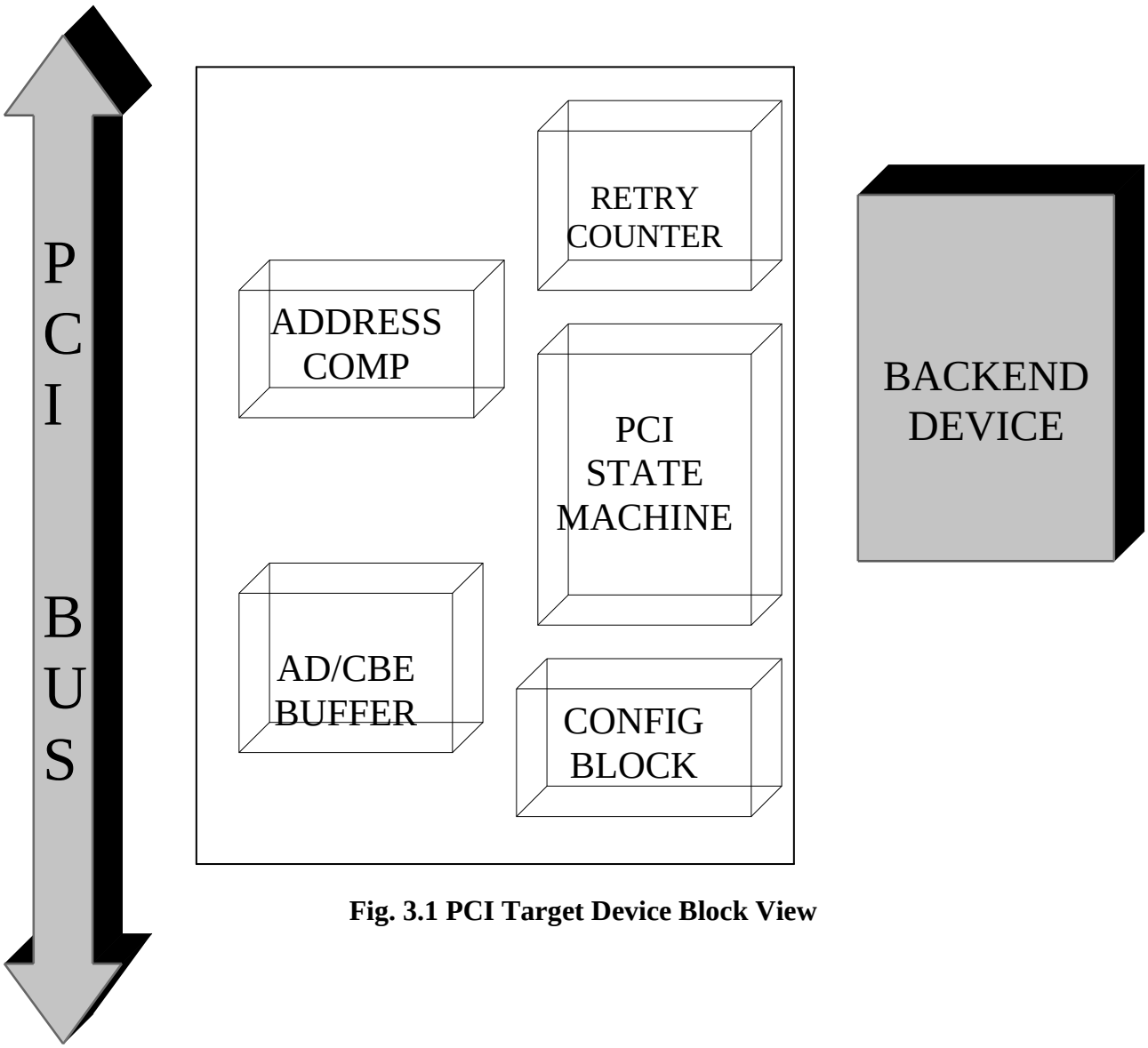


Fig. 3.1 PCI Target Device Block View

3.2.1 Address Comparator Block

This block is used to compare the address generated from the master in the first with the contents of bar address register (BAR). The BAR is a 32-bit register contains the address of the memory is compared with the address bits. If both the addresses get matched, then this block generates an indication by asserting the HIT to the state machine. The state machine will generate the indication to the master by asserting the DEVSEL. When the DEVSEL is asserted the master will decide the requested device is found.

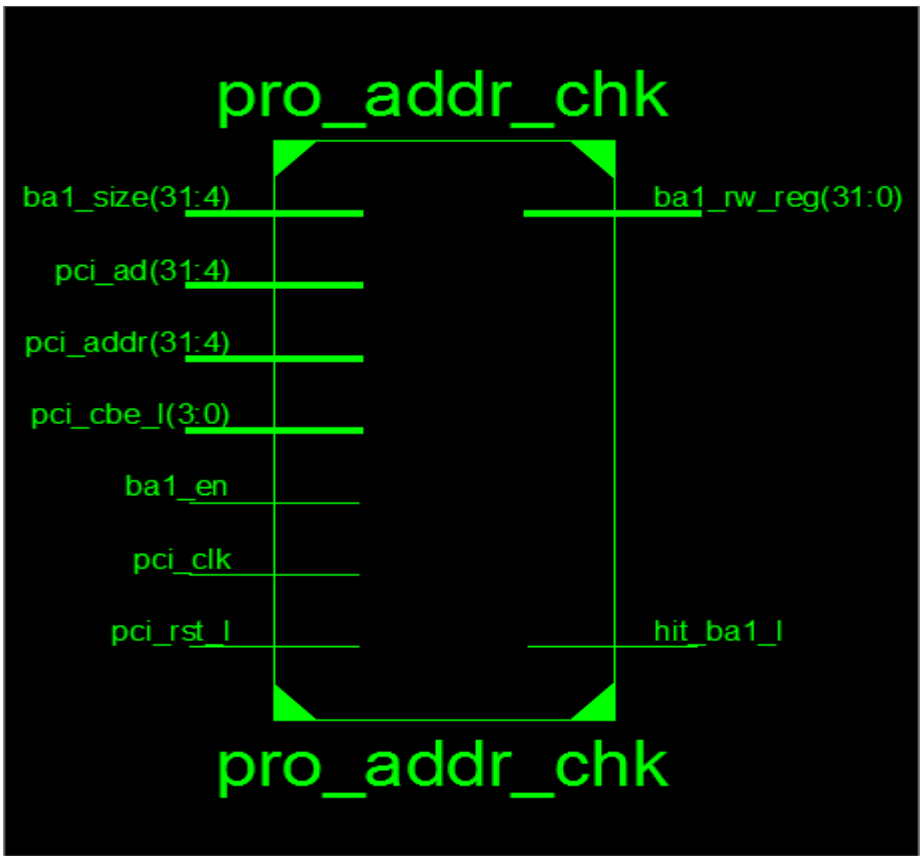


Fig 3.2 Address Check Block

3.2.2 Configuration Block

The configuration block contain various registers like Command register, Status register, Base Address register which contain information like vendor ID, device ID, memory space required by the device etc. All this information can be read or written onto the configuration space by performing special configuration read and write transactions.

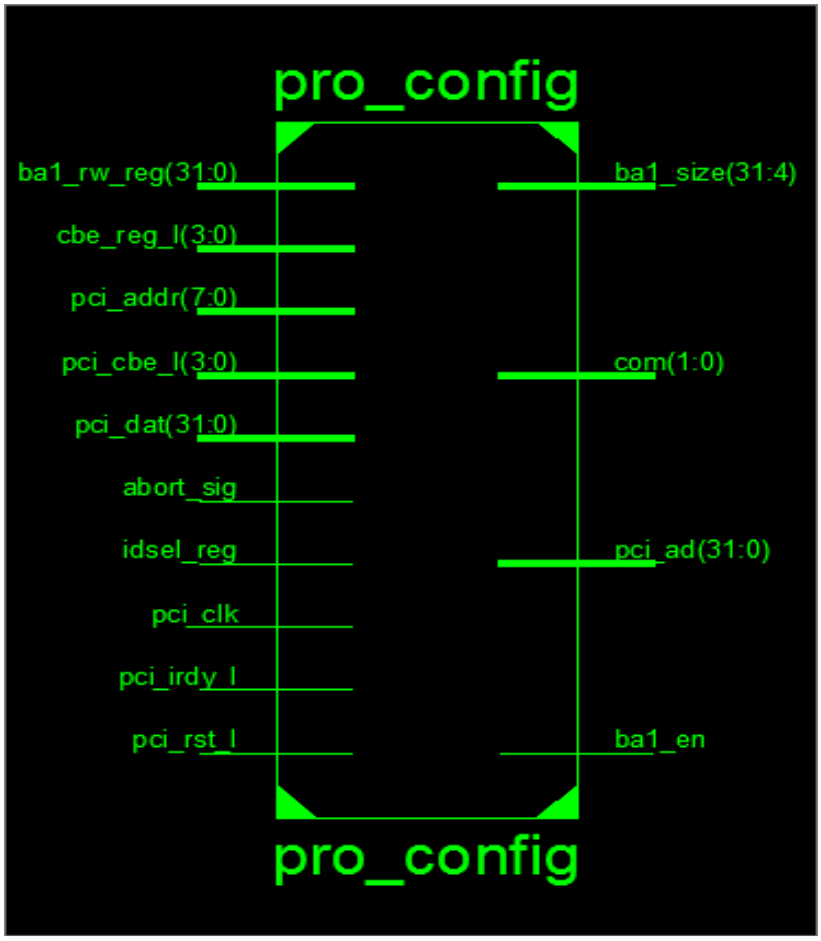


Fig. 3.3 Configuration Block

3.2.3 AD-C/BE# Buffer

This block is used to latch the address and command for the transaction in the address phase and supplies to the memory. This block consists of rising edge detector for the pci_ad_enable signal. The output of the edge detector will enable the address and command to be latched onto the address and command registers respectively. The address needs to be latched because address and data lines are multiplexed, i.e. same AD lines are used to carry data in the data phase.

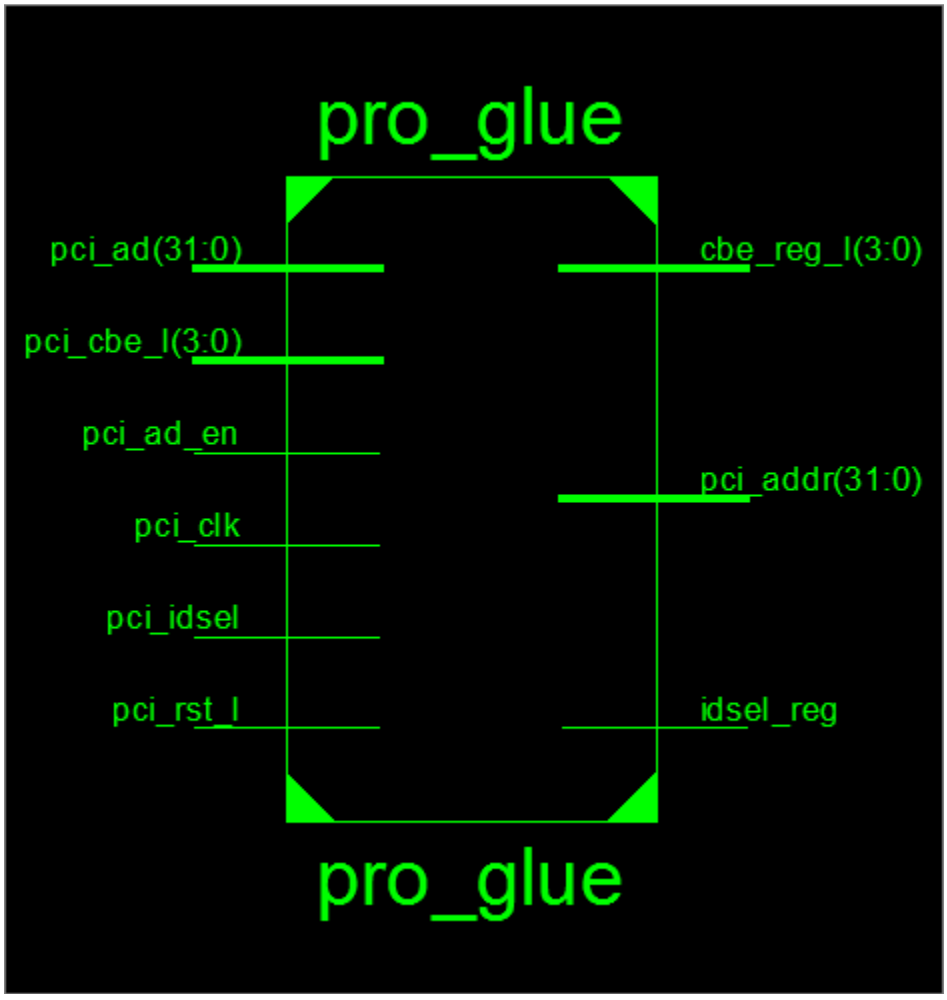


Fig. 3.4 AD-CBE# Latch Block

3.2.4 Retry Counter

Retry counter also known as the latency timer takes count enable and reset as inputs. The counter starts to count the number of clock pulses on the rising edge of the count enable signal. This counter is used to determine if the target in ready or not within the specified number of clock pulses after the assertion of frame signal.

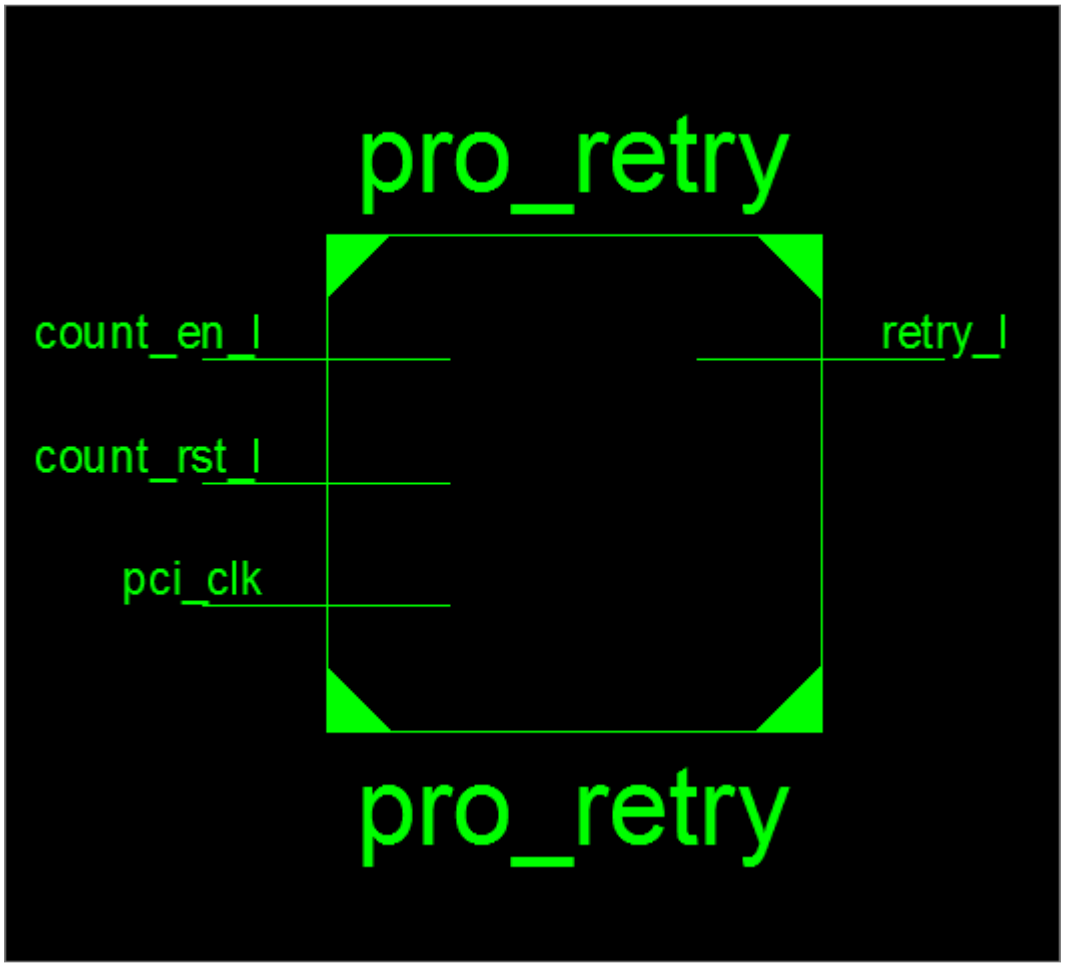


Fig 3.5 Retry Counter Block

3.2.5 State Machine

State machine block is just like the heart of the target device. It controls the signal flow to or from all the blocks shown in fig.3.1. It comprises of 9 states from idle through backoff. The state machine jumps back onto the idle state each time a transaction gets completed.

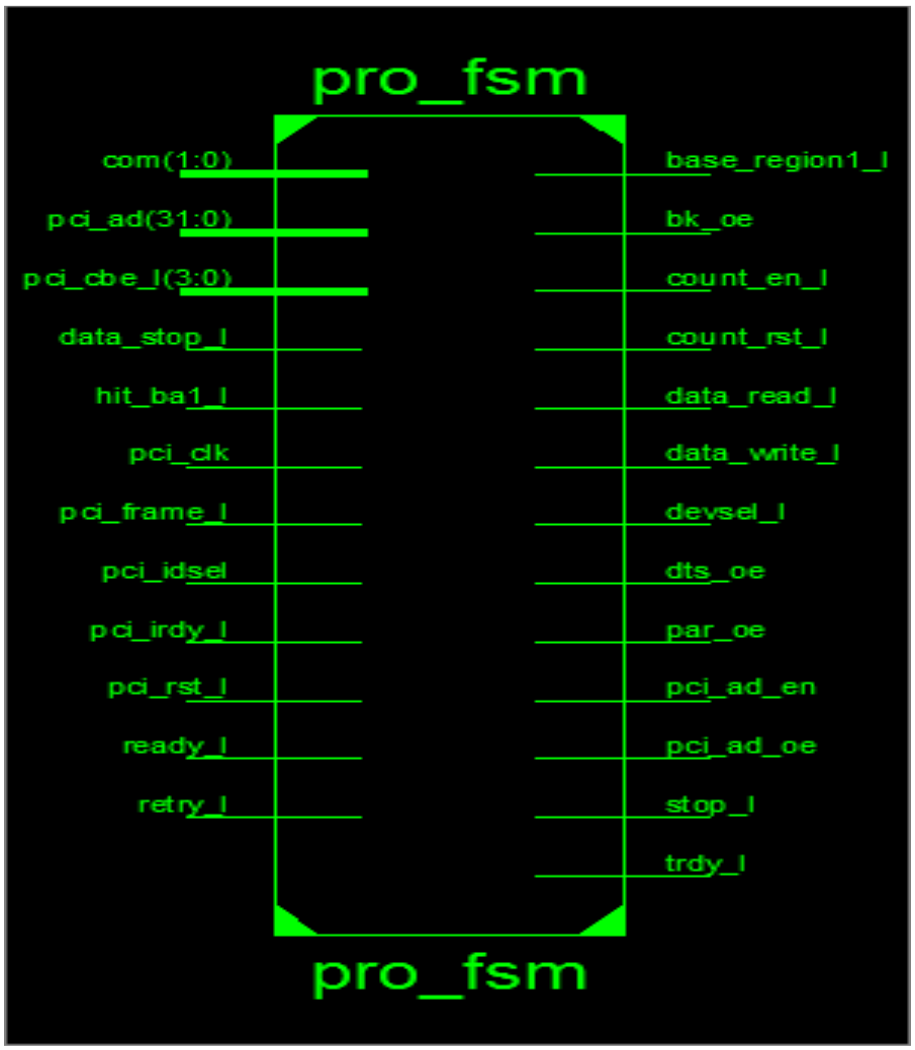


Fig 3.6 State Machine Block

3.3 PCI Architectures

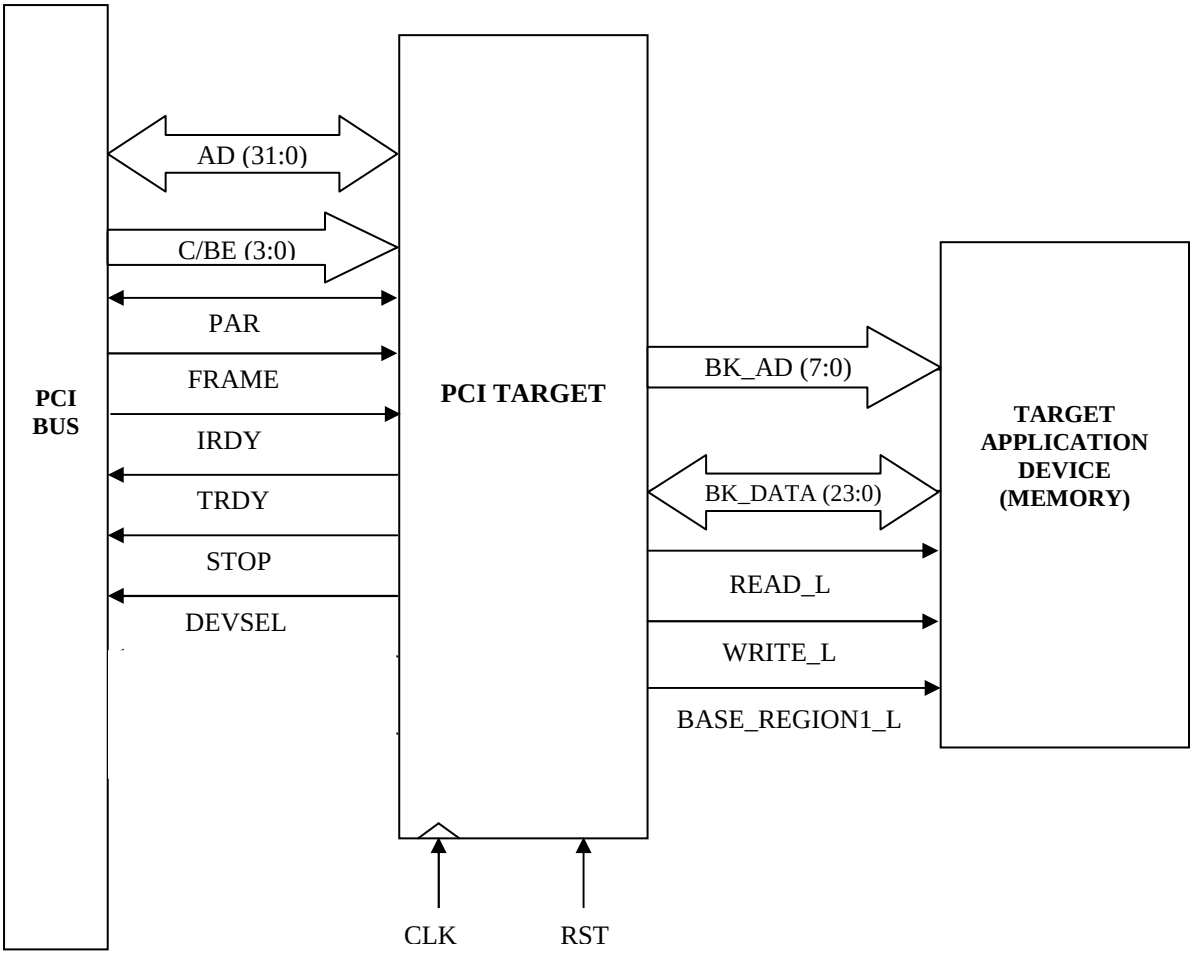


Fig. 3.7(a) Basic System Architecture of PCI Target Device[5,8,9]

Fig. 3.7 shows the system architectures of PCI Target device. Refer to figure 3.7 (b) it can clearly be observed that after re-reusing, while the PCI AD bus remains its original function, its lower 8 bits (AD[7:0]) also realizes the address bus of the peripheral device and the higher 24 bits (AD[31:8]) realizes the data bus of the peripheral device. At the same time it can also be observed that lower 3 bits (C/BE[2:0]) realizes all the control signals. This can be made possible by implementing the re-reuse logic.

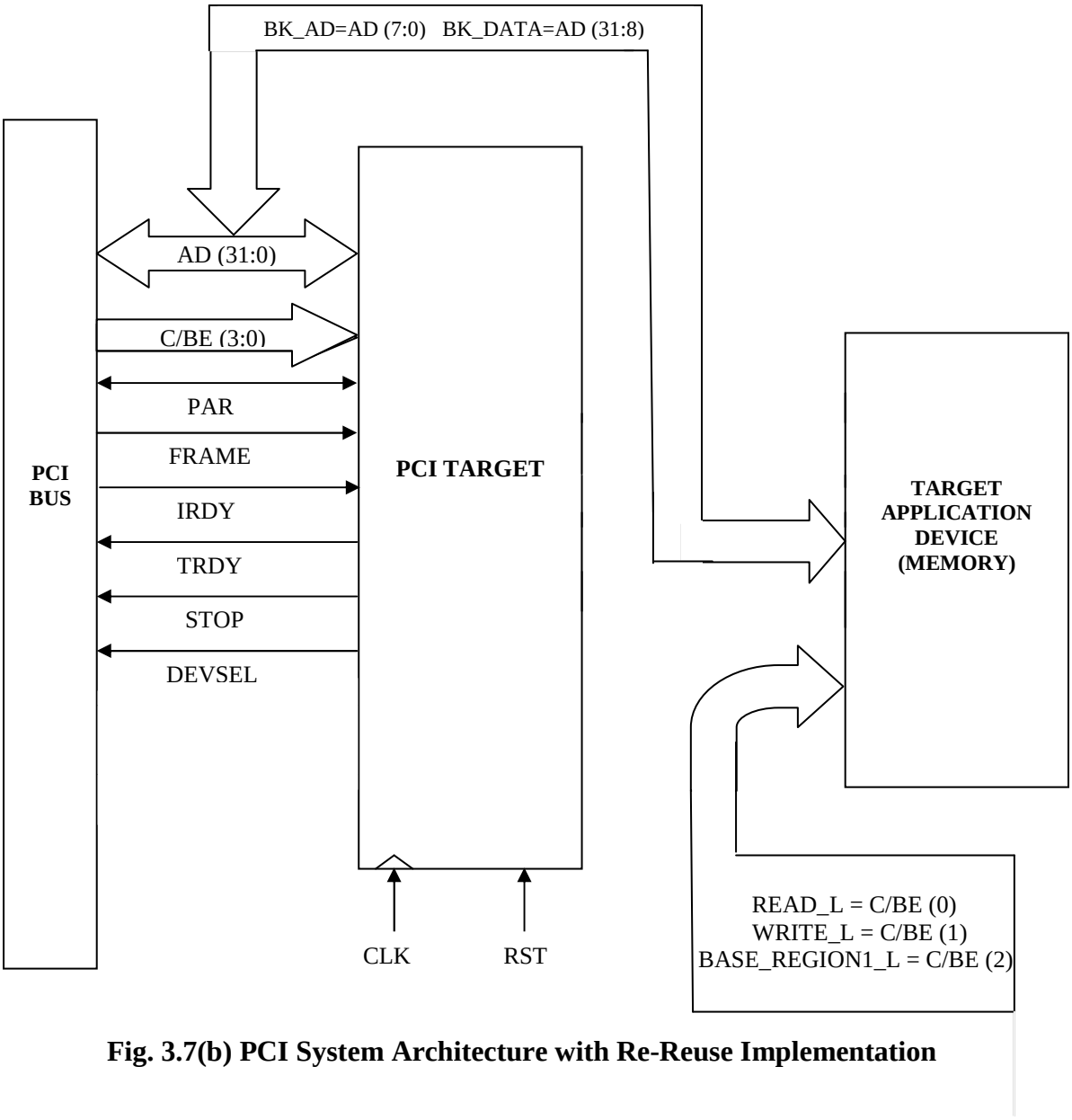


Fig. 3.7(b) PCI System Architecture with Re-Reuse Implementation

3.4 PCI Timing Characteristics

Figure 3.8(a) shows the timing of a typical read transaction, one that transfers data from the Target to the Initiator. Let's follow it cycle-by-cycle.

Clock

1 The bus is idle and most signals are tri-stated. The master for the upcoming transaction has received its GNT# and detected that the bus is idle so it drives FRAME# high initially.

2 Address Phase: The master drives FRAME# low and places a target address on the AD bus and a bus command on the C/BE# bus. All targets latch the address and command on the rising edge of clock 2.

3 The master asserts the appropriate lines of the C/BE# (byte enable) bus and also asserts IRDY# to indicate that it is ready to accept read data from the target. The target that recognizes its address on the AD bus asserts DEVSEL# to acknowledge its selection.

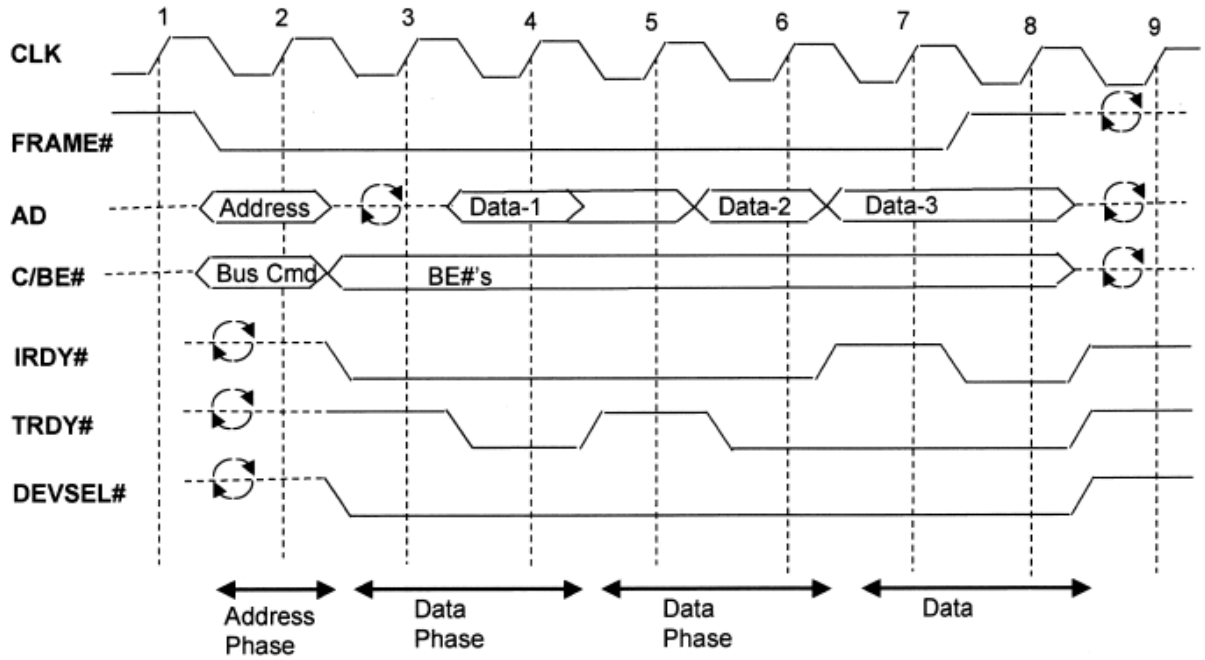


Figure 3.8(a) Timing diagram of Read Operation[3,5,6]

This is also a turnaround cycle: In a read transaction, the master drives the AD lines during the address phase and the target drives it during the data phases. Whenever more than one device can drive a PCI bus line, the specification requires a one-clock-cycle turnaround, during which neither device is driving the line, to avoid possible contention that could result in noise spikes and unnecessary power consumption. Turnaround cycles are identified in the timing diagrams by the two circular arrows chasing each other.

4 The target places data on the AD bus and asserts TRDY#. The master latches the data on the rising edge of clock 4. Data transfer takes place on any clock cycle during which both IRDY# And TRDY# are asserted.

5 The target deasserts TRDY# indicating that the next data element is not ready to transfer. Nevertheless, the target is required to continue driving the AD bus to prevent it from floating. This is a wait cycle.

6 The target has placed the next data item on the AD bus and asserted TRDY#. Both IRDY# and TRDY# are asserted so the master latches the data bus.

7 The master has deasserted IRDY# indicating that it is not ready for the next data element. This is another wait cycle.

8 The master has reasserted IRDY# and deasserted FRAME# to indicate that this is the last data transfer. In response the target deasserts AD, TRDY# and DEVSEL#. The master deasserts C/BE# and IRDY#. This is a master-initiated termination.

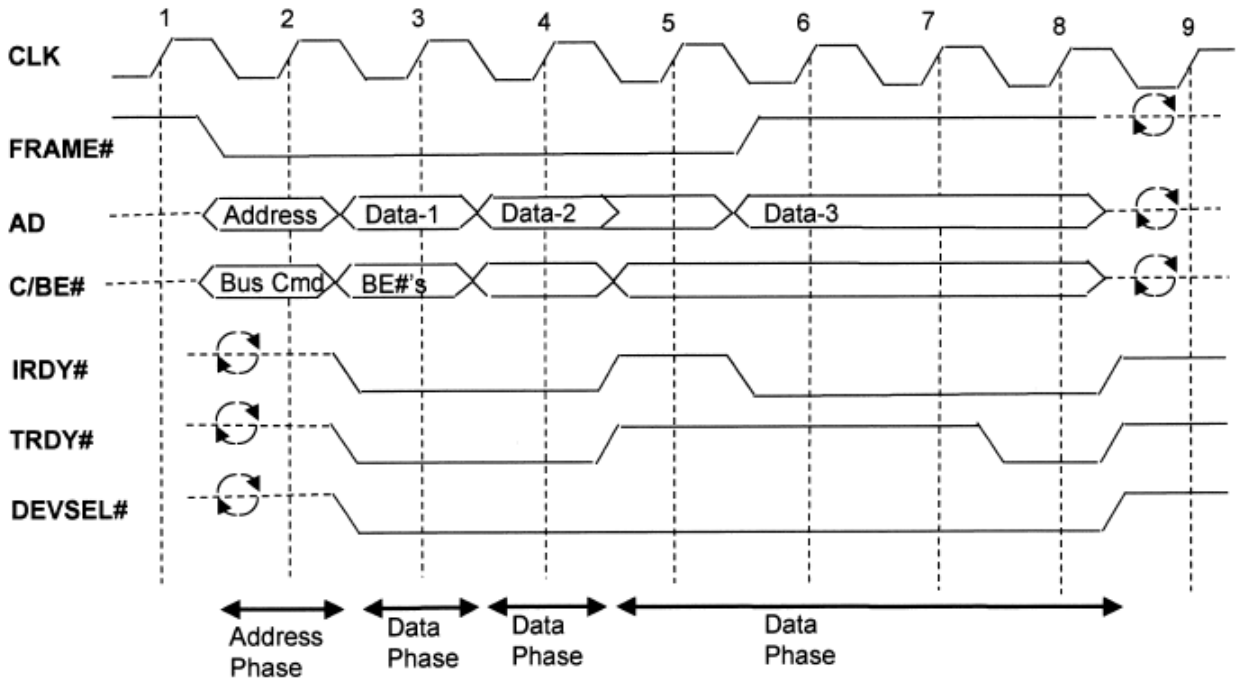


Figure 3.8(b)Timing diagram of write operation[3,5,6]

Figure 3.8(b) shows the details of a typical write transaction where data moves from the master to the target. The primary difference between the write transaction and the read transaction detailed in Figure 3.8(a) is that write does not require a turnaround cycle between the address

and first data phase because the same agent is driving the AD bus for both phases. Thus the master can drive data onto the AD bus during clock 3.

From above analysis we can see that during the read operation, after AD bus turn around, the target controls the AD bus and C/BE# bus during the data phase, by pulling up the TRDY signal target can insert the wait state. Bus turn around and wait state provide the timing possibility for re-reusing the PCI AD bus and C/BE# bus.

3.5 FSMs Implementing Re-reuse Technology

Two Finite State Machines have been adopted to control the data transactions. One is the main control unit shown in figure 3.9(a), the other control the re-reuse operation shown in figure 3.9(b). How to design them is the most important factor in the re-reuse technology.

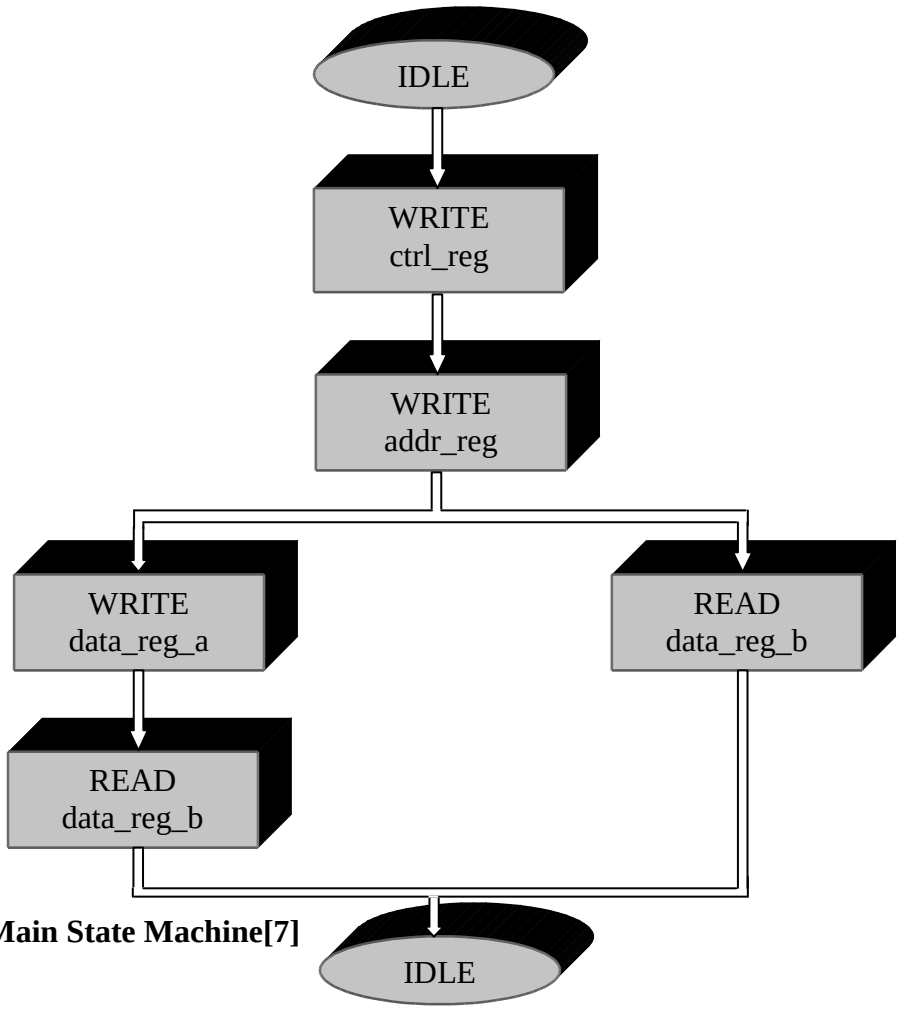


Fig 3.9(a) Main State Machine[7]

If the initiator wants to write to the peripheral device then, first set up ctrl_reg equal to 1, then write addr_reg and data_reg_a. These operations are all standard PCI I/O WRITE transaction process. After finishing the above operations, the initiator reads from the data_reg_b which will start the write process to the peripheral device. The read from data_reg_b is a special transaction based on the standard PCI I/O READ transaction.

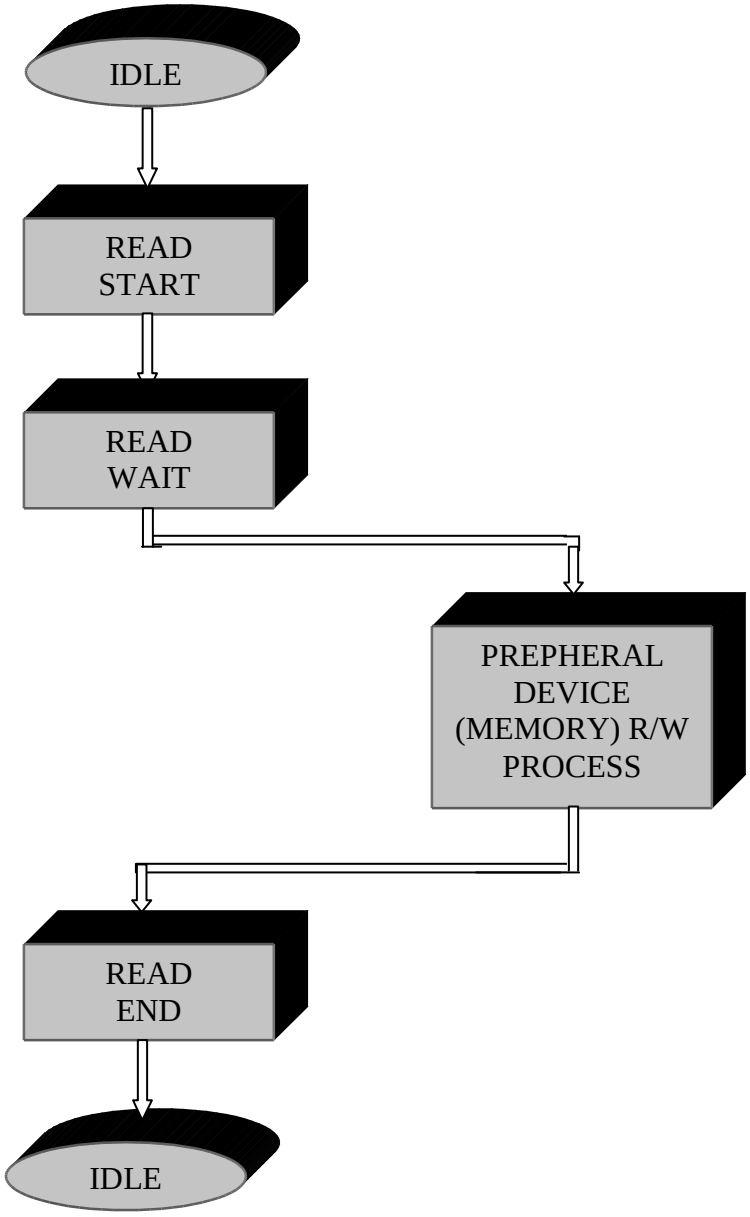


Fig. 3.9(b) FSM Implementing Re-reuse Operation[7]

When want to read from the peripheral device, first set up ctrl_reg equal to 0, then write addr_reg. Differ from write to the peripheral device, there no need to write data_reg_a as the data needs to be read from the memory. Having finished above operations, the initiator reads from the data_reg_b will start the read process to the peripheral device.

The second FSM controls this special I/O READ transaction. The state diagram of second slate machine is shown in figure 3.9(b), we can see that once the read is starting, the target inserts the wait state to allow the peripheral device to use the PCI AD bus and C/BE bus. The target remains in the wait state until the peripheral device finishes its READ/WRITE operation and after the completion of transaction target jumps back to the idle state[7].

Chapter 4

FIELD PROGRAMMABLE GATE ARRAY

This chapter gives an introduction about basic concepts of FPGA and its Synthesis Flow. FPGA is a device that consists of Configurable Logic Blocks (CLB), I/O Buffers and Reconfigurable Interconnects. It can perform simple addition and subtraction to complex digital filtering and error detection and correction.

4.1 Introduction

Field Programmable Gate Array (FPGA) is a semiconductor device that can be programmed by the designer after manufacturing hence it is called “field- programmable”. FPGAs blend the benefits of both hardware and software. They are programmed using a logic circuit diagram or a source code in Hardware Description Language (HDL) like VHDL or Verilog. The ability to update the functionality after shipping offers advantages for many applications. FPGAs contain programmable logic components called Configurable Logic Blocks (CLBs), I/O Buffers and a hierarchy of Reconfigurable Interconnects that allow the blocks to be wired together somewhat like a one chip programmable breadboard. In most FPGAs, the logic block also includes memory elements, which may be simple flip flops or blocks of memory. CLBs can be configured to perform complex functions or merely simple logic gates like AND and XOR.

FPGAs implement circuits just like hardware performing tasks, yet can be reprogrammed cheaply and easily to implement a wide range of at the same time having huge power, area and performance benefits over softwares. A few "mixed signal FPGAs" have integrated peripheral Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs) with analog signal conditioning blocks allowing them to operate as a system on a chip.

[12]Such devices blur the line between an FPGA, which carries digital ones and zeros on its internal programmable interconnect fabric, and field-programmable analog array (FPAA), which carries analog values on its internal programmable interconnect fabric.

4.2 FPGA Implementation

The FPGA that is used for the implementation of the circuit is the Xilinx Spartan 3E (Family), XC3S5000 (Device). The working environment/tool for the design is the Xilinx ISE 8.2i is used for FPGA Design flow of Verilog code.

4.2.1 FPGA Design Flow

Complexity of FPGA has been increased since its evolution. Today, FPGA vendors provide a fairly complete set of design tools that allows automatic synthesis and compilation from design specifications in Hardware Descriptive Languages such as Verilog or VHDL, all the way down to a bit stream to program FPGA chips. A typical FPGA design flow is shown in Fig 4.1. Inputs to the design flow typically include

I. HDL specification of the design.
II. Design constraints.
III. Specification of target FPGA devices.

Design constraints typically include the expected operating frequencies of different clocks, the delay bounds of the signal path delays from input pads to output pads (I/O delay), from the input pads to registers (setup time), and from registers to output pads (clock-to-output delay). In some cases, delays between some specific pairs of registers may be constrained. Each FPGA vendor typically provides a wide range of FPGA devices, with different performance, cost, and power tradeoffs.

The selection of target device may be an iterative process. The designer may start with a low capacity device with a nominal speed-grade. If synthesis effort fails to map the design into the target device, the designer has to chose a high-capacity device. The cost of FPGA device can increase in some cases by 50% or even by 100%.

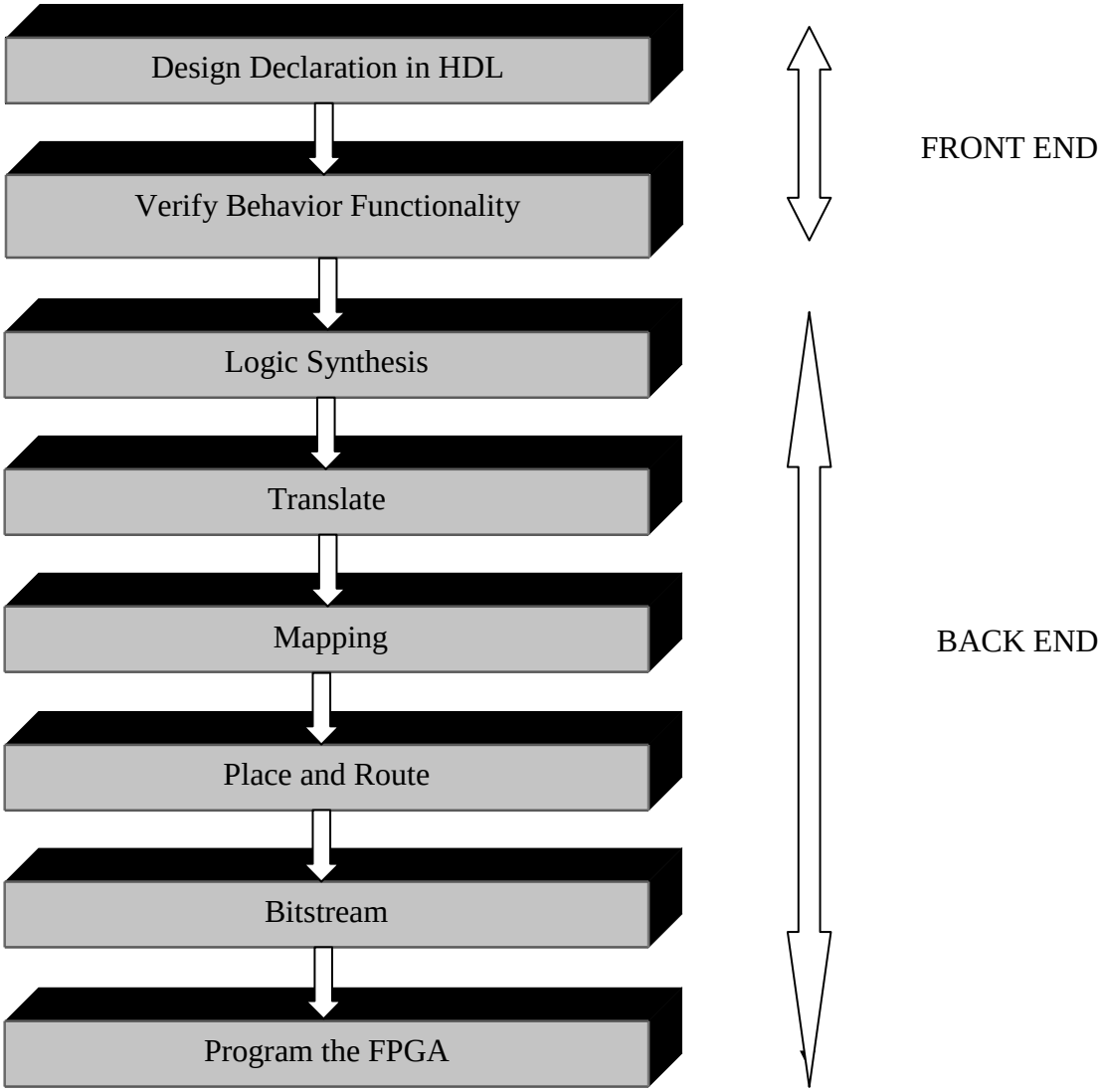


Figure 4.1: FPGA Design Flow[13]

Following are the steps involved in FPGA design flow

a) Design Entity

In this step, the basic architecture of the system is designed in a Hardware Description Language like Verilog or VHDL. The design module is split into two parts, each of which is called a design unit in Verilog. The external module which contain declarations and the internal module which represents the internal description of the design module i.e. its behavior or its structure or a mixture of both.

b) Functional Simulation

Now as the design is coded into an HDL, next step is to create a test bench waveform containing input stimulus to verify the functionality of the design using a simulation software i.e. Modelsim SE for to generate outputs and if it verifies then proceed further, otherwise modifications and necessary corrections will be done in the HDL code.

c) Logic Synthesis

During synthesis, the Xilinx ISE tool does the following operations:

I. HDL Compilation: The tool compiles all the sub-modules in a main module if any and checks the syntax of the design.

II. HDL Synthesis: In this process the code is translated into a device netlist format, i.e. a complete circuit with logical elements such as multiplexer, adder/subtractor, counters, registers, flip flops, latches, comparators, decoders, etc. for the design. Suppose if we have to implement a processor, we need a CPU as one design element and RAM as another and so on, and then the synthesis process generates netlist for each design element. The resulting netlist is saved to an NGC (Native Generic Circuit) file (for Xilinx® Synthesis Technology (XST)).

III. Advanced HDL Synthesis: It is further defined in terms of the low level blocks such as buffers, lookup tables. It also optimizes the design by eliminating the redundant logic. The tool then generates a 'netlist' file (NGC file) and then optimizes it. The final netlist output file has an extension of .ngc which contains both the design data and the constraints. The optimization goal is pre-specified to be the faster speed of operation or to minimize the area of implementation before running this process. The level optimization effort can also be specified. The higher the effort, the more optimized is the design but higher effort can also be specified. The higher the effort, the more optimized is the design but higher effort requires larger CPU time (i.e. the design time) as multiple optimization algorithms are tried to get the best result for the target architecture.

d) Design Implementation

The design implementation process consists of the following sub processes:

I. **Translation:** The Translate process combines all the input netlists and constraints to a logic design file. This information is saved an NGD (Native Generic Database) file. The file having extension .ngd describes the logical design reduced to the Xilinx device primitive cells. Here assignment of ports in the design to the physical elements (ex. pins, switches, buttons etc) of the targeted device is done. This information is stored in a file User Constraints File (UCF). Tools like PACE, Constraint Editor etc. are used to create or modify the UCF.

II. **Mapping:** It is done after the completion of translate process. Mapping process divides the whole design with logical elements into sub blocks such that they can fit into the FPGA logic blocks. That means map process fits the logic defined by the NGD file into the targeted FPGA elements Combinational Logic Blocks (CLB), I/O Blocks and generates a Native Circuit Description (NCD) file which physically represents the design mapped to the components of FPGA.

III. **Place and Route:** The place and route process places the sub blocks from the map process into logic blocks according to the constraints and connects the logic blocks. For example if a sub block is placed in a logic block which is placed very near to I/O pin, then it may save the time but it may affect some other constraint. So place and route process takes into account a tradeoff between all the constraints.

IV. **Bitstream Generation:** The collection of binary data used to program the reconfigurable logic device is most commonly referred to as a "bitstream". This usage may have originated based on the common method of configuring the FPGA from a serial bit stream, typically from a serial PROM or flash memory chip, although most FPGAs also support a byte-parallel loading method as well. The detailed format of the bitstream for a particular FPGA chip is usually considered proprietary to the FPGA vendor.

V. **Static timing analysis:** There are three types of static timing analysis that can be performed :

- **Post-fit Static timing analysis:** The Analyze Post-Fit Static timing process opens

the timing analyzer window, which interactively select the timing paths in design for tracing.

- **Post-Map Static Timing Analysis:** It analyzes the timing results of the Map process. These reports can be very useful in evaluating timing performance.
- **Post Place and Route Static Timing Analysis:** It analyzes the timing results of the Post- Place and Route process.
- **Timing Simulation:** Perform Post-Place and Route simulation after the design has been and routed. This simulation process allows to analyze how the design will behave in the circuit.

4.3 Applications of FPGA

Applications of FPGAs include digital signal processing (DSP), software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection and a growing range of other areas.

Chapter 5
SYNOPSYS DESIGN COMPILER

The Design Compiler is a synthesis tool from Synopsys Inc. Basically, the synthesis tool takes a Register Transfer Logic hardware description written in HDLs like VHDL or Verilog, and standard libraries as input and results in a technology dependent gate level netlist. It is nothing but structural representation of only standard cells based on the cells in the standard cell library.

5.1 DC Synthesis Flow

5.1.1 Reading of Design and Library

Design Compiler reads the RTL hardware description written in either HDLs like VHDL or Verilog. Design Compiler reads in technology libraries, design ware libraries, and symbol libraries to implement synthesis. During the synthesis process, it translates the RTL description to components extracted from the technology library and design ware library. The technology library contains basic logic gates and flip-flops. The design ware library consists of more complex cells for example adders and comparators which can be used for arithmetic building blocks. Design Compiler can automatically determine when to use design ware components and it can then efficiently synthesize these components into gate-level netlist.

5.1.2 Design Environment

The design environment is a set of attributes and constraints that model the surroundings of the design being synthesized. Design Compiler uses these attributes and constraints to compute the effect of the design environment on the circuit's performance. The design environment includes the following items:

a. Operating Conditions

The operating conditions include the operating manufacturing process, supply voltage and, operating temperature.

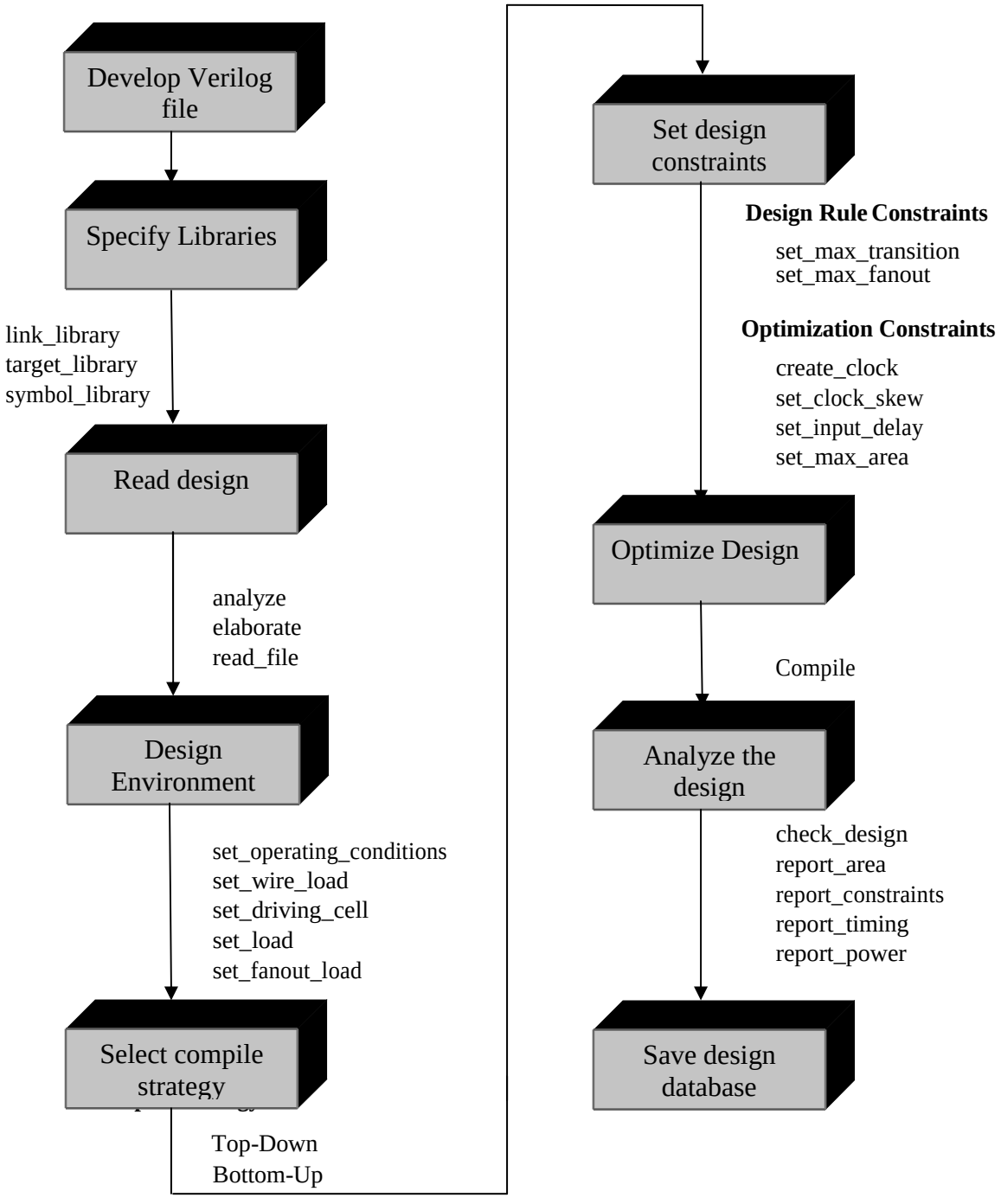


Figure 5.1: Design Compiler Synthesis Flow [14]

b. Wire Load Models

Wire load models estimates the effect of wire length and fan-out on resistances and capacitances to estimate wire delays.

c. System Interface

The system interface includes the cells driving the design and the loads driven by the design.

5.1.3 Compilation Strategy

There are two main strategies in synthesis compilation Top-Down and Bottom-Up. Different strategy can be used for different designs.

a. Top-Down Compilation

In this the designer only needs to be concerned with top-level design constraints. The design constraints of submodules in lower-level hierarchy need not to be taken into account. Submodule timing information is handled by Design Compiler from top-level hierarchy. Top-Down Compilation is not advisable for large designs. The compilation time can be much longer in the case where the design is large, moreover the Design Compiler might crash due to insufficient memory.

b. Bottom-Up Compilation

In this compilation method, compilation begins at the submodule level and moves towards the top level. In this case the designer needs to know the timing information for the input ports and output ports for each submodule so the designer needs to perform time budgeting on the submodules. It is much faster than the Top-Down Compilation and advisable for large designs.

5.1.4 Design Constraints

A designer, in order to achieve optimum results for the design, has to constrain the design by describing the design environment, target objectives and design rules. These constraints contain timing and/or area information which are usually derived from the design specifications. The synthesis tool uses these constraints to perform synthesis and tries to optimize the design.

a. Design Rule Constraints

Design rule constraints have two sources, one is attributes specified in the technology library and other is attributes applied explicitly. If a technology library defines these attributes, then Design Compiler implicitly applies them to the design using that library when it compiles the design or creates a constraint report. The design rule attributes defined in the technology library cannot be removed as they are requirements for the technology, but these can be made more restrictive to suit the design.

The most commonly specified design rule constraints are:

- Transition time constraints
- Fan-out load constraints
- Capacitance constraints

I. **Transition Time Constraints:** The transition time is the time required for its driving pin to change logic values. Design Compiler calculates the transition time for each net by multiplying the driving resistance of the pin by the sum of the capacitive loads connected to the driving pin. The maximum transition time restriction specified in a technology library can be attained by using the set_max_transition command. This command sets a maximum transition time for the nets attached to the identified ports or to all the nets in a design by setting the max_transition attribute on the named objects.

II. **Fan-out Load Constraints:** The maximum fan-out load for a net is the maximum number of loads each net can drive. The fan-out load value does not represents capacitance; it represents the weighted numerical contribution to the total fan-out load. Design Compiler calculates the fan-out of a driving pin by adding the fan-out load values of all inputs driven by that pin. To determine if the pin meets the maximum fan-out load restriction, Design Compiler compares the calculated fan-out load value with that of the pin's max_fanout value. The command set_max_fanout is used to set the maximum number of fan-out allowed on input port of design. The set_fan_out_load command is used to set the fan-out load on an output port of a design.

III. **Capacitance Constraints:** The maximum capacitance constraint is used to control capacitance directly. DC calculates the capacitance on a net by adding the wire capacitance to the capacitance of the pins attached to that net.

To determine if the net meets the capacitance restrictions, Design Compiler compares the calculated capacitancevalue with that of the pin's max_capacitance value. set_max_capacitance command is used to change or add to the maximum capacitance restriction specified in a technology library. The set_max_capacitance command is used to set a maximum capacitance for all the nets attached to the named ports or to all the nets in the design by setting max_capacitance attribute on the specified objects.

b. Optimization Constraints

Timing constraints specify the required performance of the design. To set the timing constraints clock, clock latency, I/O delay etc. have to be defined.

I. **Defining a Clock:** For synchronous designs, the clock period is the most important constraint as it constrains all register-to-register paths in the design. The command create_clock is used to define a clock object with a particular period and duty cycle. The period option defines the clock period, while the waveform option controls the duty cycle and the starting edge of the clock.

II. **Specifying Clock Network Delay:** By default, Design Compiler assumes that clock networks have no delay (ideal clocks). The set_clock_latency is used command to specify timing information about the clock network delay.

III. **Input Delay:** It specifies the input arrival time of a signal in relation to the clock. It is used on the input ports to specify the time taken by the data to be stable after the clock edge. The set_input_delay command defines the arrival times of inputs.

IV. **Output Delay:** It specifies the time taken by the data to be available after the clock edge. The set_output_delay command defines the required arrival time of outputs.

c. Area Constraints

The set_max_area command specifies maximum area for the current design by placing a max_area attribute on the current design. We need to specify the area in the same units used for area in the technology library. Design area is composed of the areas of each component and nets. This is to be noted that the components with unknown areas and technology-independent generic cells are ignored while Design Compiler calculates design area.

5.2 Save the Design

Design Compiler can save a database in different formats. A common format in which to save a synthesized database is the Synopsys DB format.

Chapter 6

SIMULATION AND SYNTHESIS RESULTS

This chapter includes simulation and synthesis results of both PCI Target Device Architectures, one before re-reuse of PCI AD bus and C/BE bus and the other after re-reuse of PCI AD bus and C/BE bus simulated by ModelSim 10.1 and synthesized by Xilinx ISE and Synopsys Design Compiler tools.

6.1 Simulation Results

a) Simulation Results of PCI Target Device before Re-reuse Implementation

Figure 6.1 shows the simulation results of WRITE Transaction with PCI Target device before implementing re-reuse concept (Write Address= 03h & Write Data= aaaaaah). It can be clearly observed that transaction takes place when both pci_trdy and pci_irdy signals are asserted.

- pci_clk : Input clock
- pci_ad : Input (32 bits)
- pci_cbe_ l : Input (4 bits)
- pci_frame_l : Input
- pci_irdy_l : Input
- pci_trdy_l : Output
- pci_devsel_l : Output
- base_region1_l : Output
- data_write_l : Output
- data_read_l : Output

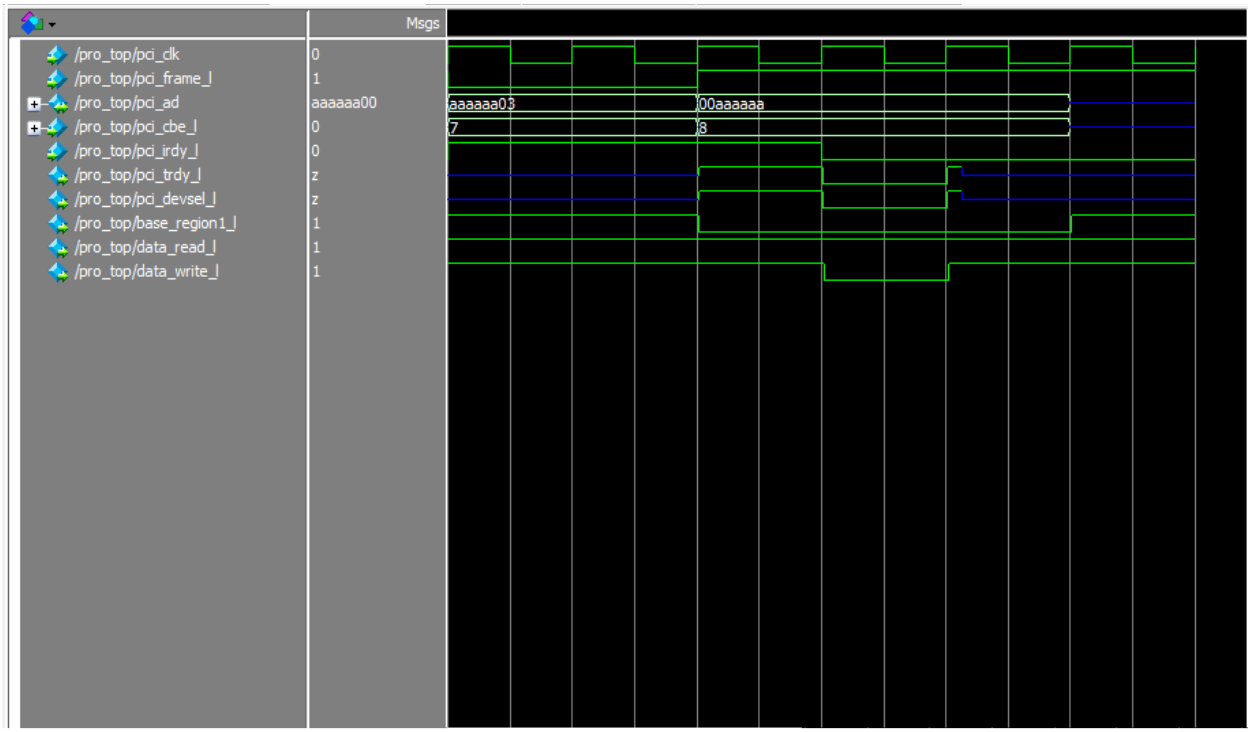


Fig 6.1 Simulation Waveforms for WRITE Transaction

b) Simulation Results of PCI Target Device after Re-reuse Implementation

Figure 6.2 shows the simulation results of WRITE Transaction with PCI Target device after implementing re-reuse concept. It can be clearly observed that transaction takes place when a wait state is inserted during standard PCI READ Transaction.

- pci_clk : Input clock
- pci_ad : Inout (32 bits)
- pci_cbe_ l : Inout (4 bits)
- pci_frame_l : Input
- pci_irdy_l : Input
- pci_trdy_l : Output
- pci_devsel_l : Output
- base_region1_l
- write_l
- read_l

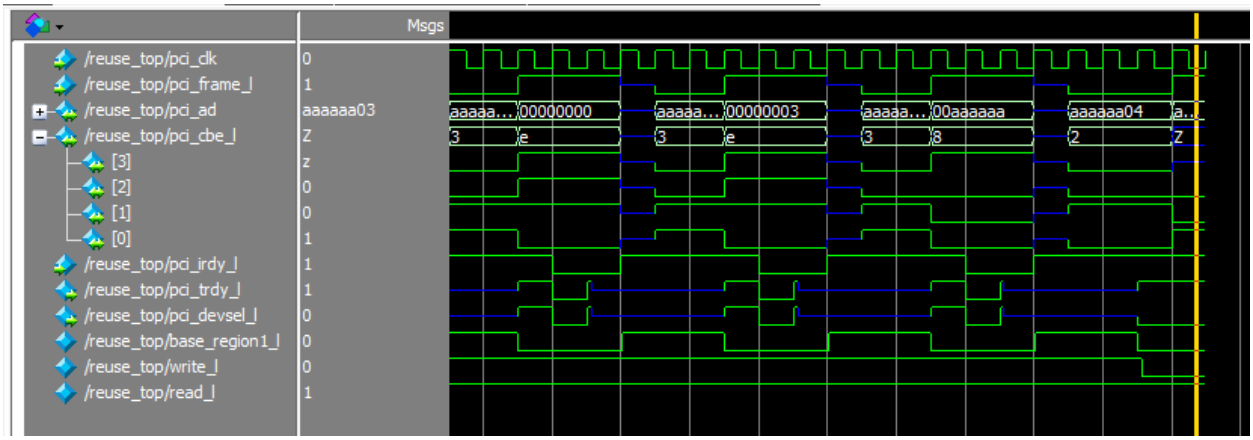


Fig 6.2 Simulation Waveforms for WRITE Transaction

6.2 Synthesis Results

6.2.1 Synthesis Results on FPGA

The Xilinx ISE 12.4 is used for implementation of all the circuits. The Working Environment for the design is:

I. Target Device : XC3S500E-4FG320

II. Technology : 90 nm

III. Tool Version : ISE 12.4

IV. FSM Style : LUT

V. Design Strategy : Balanced

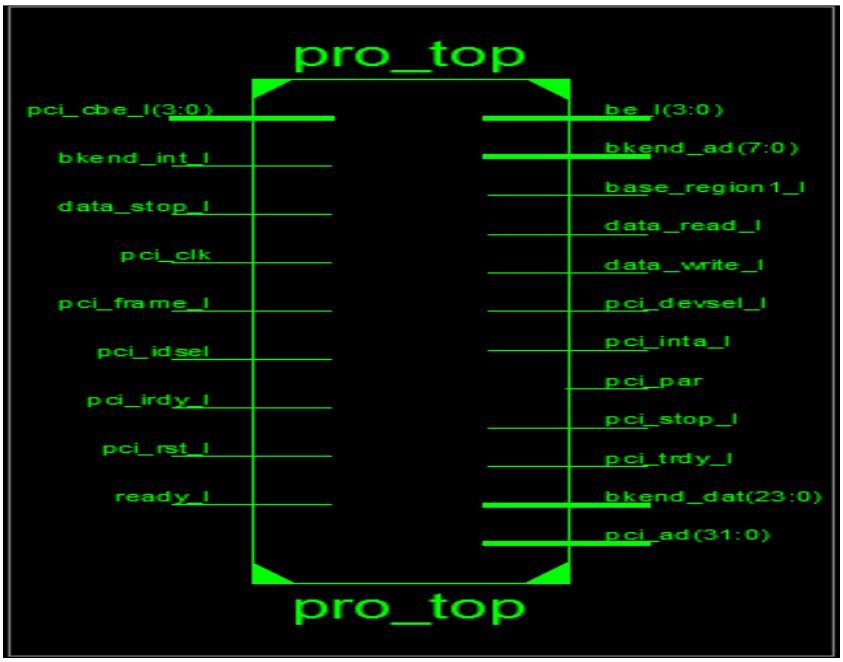
a) Synthesis Results of PCI Target Device before Re-reuse Implementation

I. Table 6.1 shows the synthesis results of PCI Target Architecture before applying the re-reuse process.

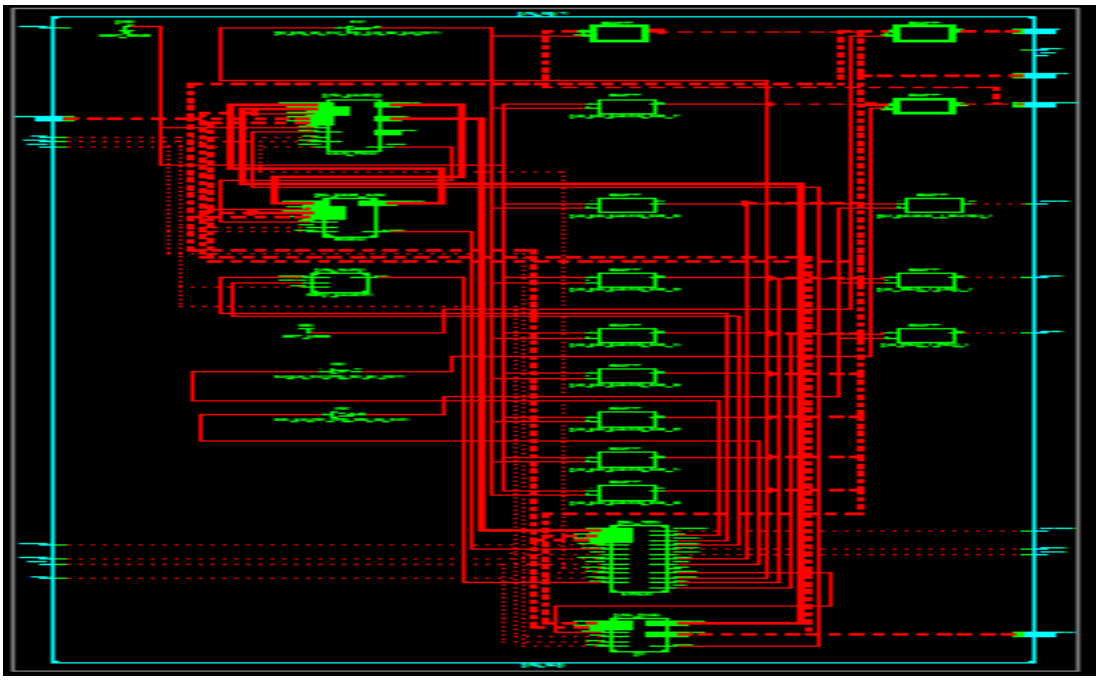
Number of Slices:	81
Number of Slice Flip Flops:	89
Number of 4 input LUTs:	128
Number of IOs:	88
Number of bonded IOBs:	87
Number of GCLKs:	1

Table 6.1 Synthesis Results of PCI Target Device before Re-reuse Implementation

II. RTL View



(a)



(b)

Fig 6.3 (a) & (b) RTL View of PCI Target Device before Re-reuse Implementation

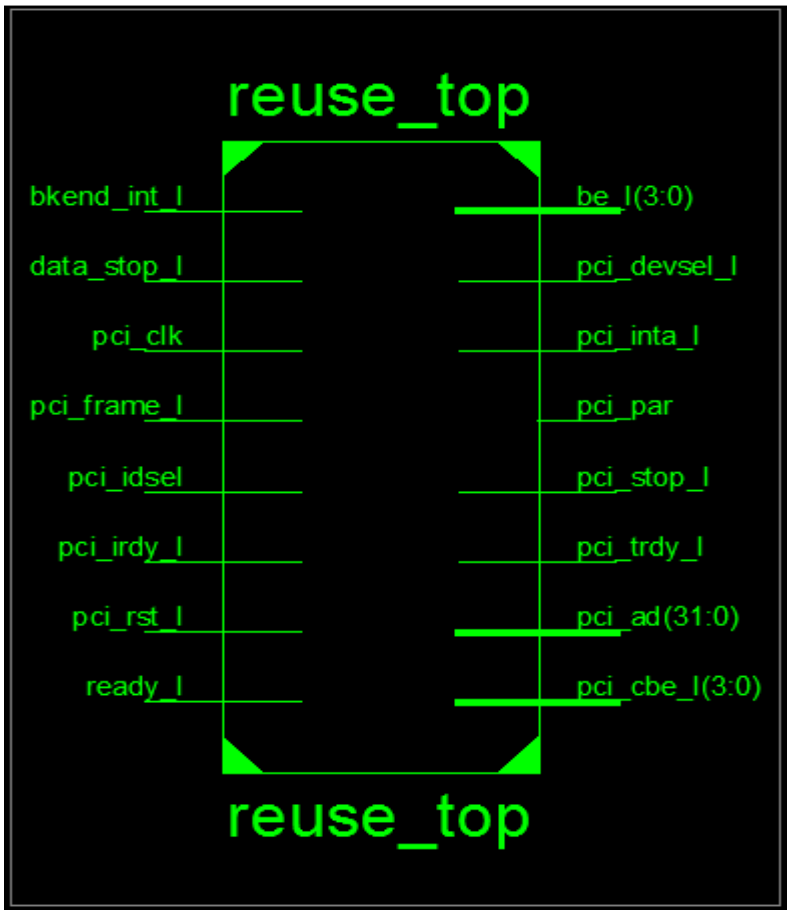
b) Synthesis Results of PCI Target Device after Re-reuse Implementation

I. Table 6.2 shows the synthesis results of PCI Target Architecture after applying the re-reuse process.

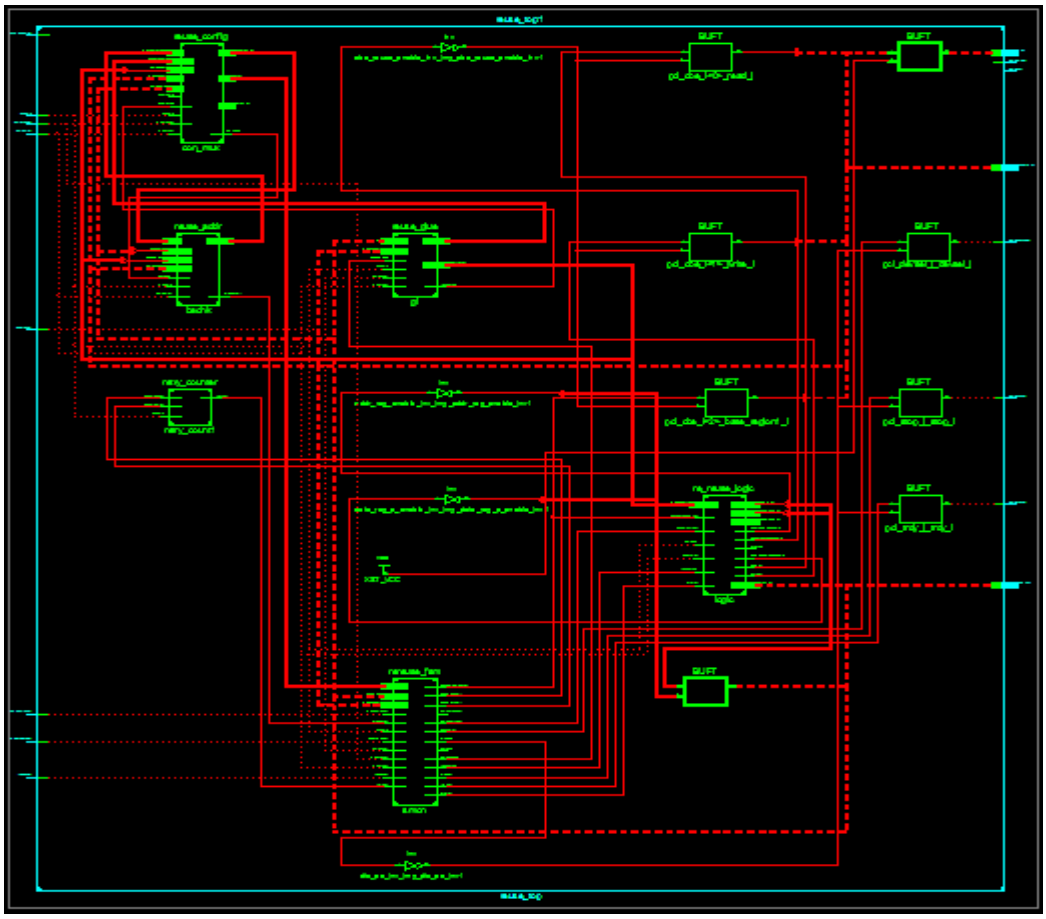
Number of Slices:	105
Number of Slice Flip Flops:	119
Number of 4 input LUTs:	123
Number of IOs:	53
Number of bonded IOBs:	52
Number of GCLKs:	1

Table 6.2 Synthesis Results of PCI Target Device after Re-reuse Implementation

II. RTL View



(a)



(b)

Fig 6.4 (a) & (b) RTL View of PCI Target Device after Re-reuse Implementation

c) Analysis

Let us analyze the results shown in table 6.1 and 6.2 with the help of a basic FPGA Architecture.

Figure 6.3 shows the basic FPGA architecture. It comprises of three basic resources

I. CLBs-They contain combinatorial logic and register resources. Each CLB contain four slices.

II. IOBs-They are the interface between the FPGA and the outside world.

III. Programmable interconnect.

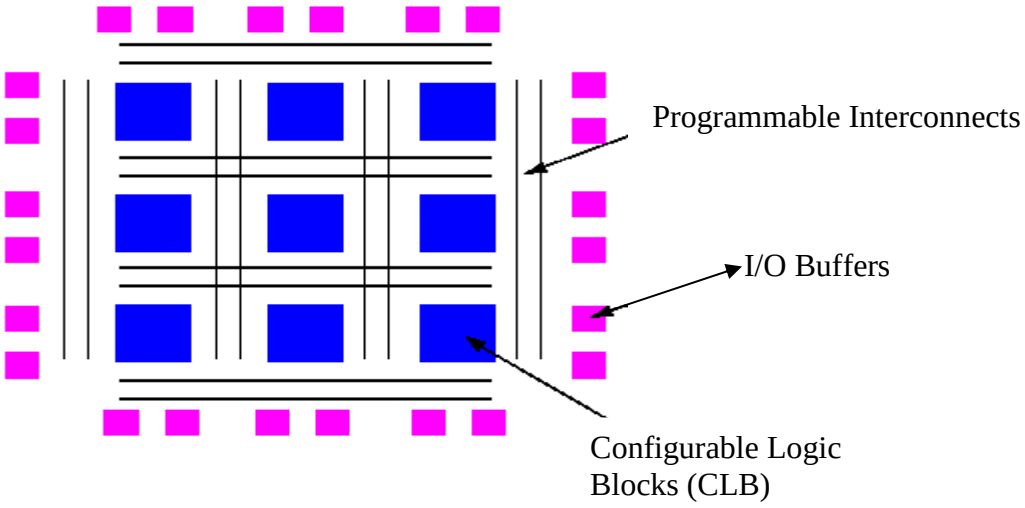


Figure 6.5[15] Basic Architecture of FPGA

So if we observe table 6.1, we can see that 81 slices have been used which contain the logic of the design and 88 I/O Buffers have been used i.e. the chip will require 88 I/O pins. Whereas if we observe table 6.2, we can see that the slice count increases to 105 as the logic gets more complicated but the number of I/O buffers reduces to 53, hence in this case comparatively less number of I/O pins will be required.

6.2.2 Synthesis Results on Design Complier

Synopsys DC tool version D-2010.03-SP1 is used for synthesis. The Working Environment is:

- Operating Condition Name : WCCOM
- Library : fsd0c_a_generic_core_ss1p08v125c
- Temperature : 125.00 0C
- Voltage : 1.08 V
- Interconnect Model : Wire load (worst case tree)
- Technology: 180 nm
- Timing Analysis Effort : Medium
- Power Analysis Effort : Low

a) Synthesis Results of PCI Target Device before Re-reuse Implementation

Table 6.3 shows the synthesis results of PCI Target Architecture before applying the re-reuse process.

Number of ports:	88
Number of nets:	212
Number of cells:	125
Number of references:	10
Combinational area:	6374.588379
Non Combinational area:	9624.394531
Total cell/Design area:	15998.976562
Total Dynamic Power	2.1969 mW

Table 6.3 Synthesis Results of PCI Target Device before Re-reuse Implementation

b) Synthesis Results of PCI Target Device after Re-reuse Implementation

Table 6.4 shows the synthesis results of PCI Target Architecture after applying the re-reuse process.

Number of ports:	53
Number of nets:	211
Number of cells:	98
Number of references:	11
Combinational area:	7049.546387
Non Combinational area:	11039.943359
Total cell/Design area:	18089.466797
Total Dynamic Power	2.3149 mW

Table 6.4 Synthesis Results of PCI Target Device after Re-reuse Implementation

c) Analysis

From table number 6.3 and table 6.4, we observe that the port count has been reduced from 88 to 53 but at the same time the logic design area has increased a bit and the total dynamic power has been increased by about 0.2 mW. Thus it is clear that the later design will save 35 I/O pins at the cost of increase in logic area and dynamic power

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

This chapter concludes what has been done in the thesis and what can be done in future.

7.1 Conclusion

The cost is an important factor in the chip design. The cost of a chip mainly includes the cost of design and the cost of manufacturing. Among them, the cost of manufacturing is chiefly determined by the amount of chips that be produced on a single wafer, i.e. depends on the area of the chip. Since size determines 95 % of a chip's cost, shrinking the size cuts the cost by that amount. One way to reduce the size of chip is to reduce the number of I/O pins on the chip which can be done by reusing the chip on time sharing basis.

	Before Re-ruse	After Re-reuse
Number of Slices:	81	105
Number of Slice Flip Flops:	89	119
Number of 4 input LUTs:	128	123
Number of IOs:	88	53
Number of bonded IOBs:	87	52
Number of GCLKs:	1	1

Table 7.1 FPGA Synthesis Comparison

So it can be clearly observed that after applying re-reuse technology, the I/O pin count has been reduced from 88 pins to 53 pins. We have been able to save 35 pins which means that the pin count has been reduced by approximately 40%. Though the logic design area has been increased but as the feature size has been decreasing very rapidly, an increase in gate count by few hundred can be incorporated very easily.

7.2 Future Scope

This thesis work has been emphasized on reducing the I/O pin count of the PCI Target Device by implementing re-reuse concept of PCI AD bus and C/BE bus. In future, work can be done on utilizing the saved pins for implementing other additional features that PCI bus offers such as 64 bit extension or interrupt handling.

REFERENCES

[1] Carl Hamacher, Zvonko Varnesic, Safwat Zaky, “Computer Organization” (fifth edition), McGraw-Hill, International Edition 2002.

[2] Prepared By : Tarak Modi Advisor: Dr. Wells. “Study Study of Advanced Bus Architecture (CPE 602)”, Summer’97.

[3] Doug Abbott, “PCI Bus Demystified” LL Technology Publishing (2000).

[4] YuXiao, Sch. of Comput. Sci. & Technol., Northwestern Potytechnical Univ., Xi'an, China, Computer Science and Electronics Engineering (ICCSEE), “The design of local bus interface unit based on PCI9054”, (2012) International Conference on 23-25 March 2012.

[5] Tom Shanley, Don Anderson, “PCI System Architecture (4th Edition)” Addison Wesley (1999).

[6] PCI Special Interest Group, “PCI local bus specification”, revision 2.2, 1998.

[7] Gan-min Zhou, Ming-Iun Gao, Yong-hua Hu, Hua-feng Cao, “A PCI Target Device with Re-reusing PCI AD Bus”, Solid-State and Integrated Circuits Technology, 2004. 7th International Conference on 18-21 Oct. 2004.

[8] Gui-shan Li,”PCI Local Bus Developer Manual”, Xian: Xidian University Press (1997).

[9] Wang K “Designing the MPCl05 PCI Bridge/Memory Controller”, IEEE 1995

[10] Xiu Jun Geng, ”The implementation of PCI expansion ROM for Raid Controller Card”, Apperceiving Computing and Intelligence Analysis, International Conference on 23-25 Oct. 2009.

[11] E. Solari, G. Willse, “PCI Hardware and Software. Architecture and Design”, Publisher: Annabooks (1996).

[12] J Mike Thompson "Mixed-signal FPGAs provide GREEN POWER". EE Times, 07-02-2007.

[13] Steve Kilts, “Advanced FPGA Design: Architecture, Implementation, and Optimization” John Wiley & Sons, 29-Jun-2007.

[14] Design Compiler User Guide v1999.10.

[15] Florent de Dinechin,Projet Arenaire, Lip Cnrs -Inria, “The Price of Routing in FPGAs”, Ecole Normale Superieure de Lyon, France.

[16] Liang Zhang, “Digital Circuit Design and Verilog HDL”, Beijing: Posts Telecom Press (2000).

[17] R.Locatelli, ”Master/Target PCI VHDL Core”, ESA 1999.

[18] Lucent Technologies, Bell Lab Innovations “ORCA Series FPGAs in PCI BUS Master (with Target) Applications“(2000)

[19] Xilinx Logic Core, ”PCI master & Slave Interfaces Version 2.0”, Oct. 1997.

[20] PCI Special Interest Group (http://www.pcisig.com).

[21] A. Abo Shosha, P. Reinhart, F. Rongen, " Reconfigurable PCI-BUS Interface (RPCI)" Central Laboratory for Electronics (ZEL), Research Centre Juelich (FZJ), D-52425, Germany.

[22] PCI Special Interest Group, "PCI Local Bus Specifications Revision 2.1", 1 June 1995.