POWER EFFICIENT MICRO-CODED INSTRUCTION BASED ARCHITECTURE FOR EXPRESS STANDARDS EXTENDED TO USB 3.0

Submitted by:

MUHAMMAD UMER
2008-NUST-MSCSE(S)-07

Supervised by:
Dr Nazar Abbas Saqib

Submitted to the Department of Electrical Engineering, School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Islamabad in partial fulfillment for the requirements of a M.S. Degree in Communication System Engineering,
January 2011

ABSTARCT

Emerging high speed communication protocols with ultra high bandwidth requirement demand efficient system architecture design to meet their specifications. Universal Serial Bus (USB) 3.0 supporting 4.8 Gbps, PCI express providing 5Gbps and Intel's forthcoming light peak supporting 10Gbps are some of the examples. Both streaming and non-streaming high-end digital devices such as HD cameras, webcams and mass storage devices employing these communication standards demand efficient system architecture that provides the required bandwidth with as little processing overhead (interrupts) for CPU as possible. Also with the substantial enhancement in digital systems, there is a demand of system architecture that consumes lesser power. The importance of low power consumption is due to the ever decreasing feature size of micro-electronic circuits, higher clock frequencies and larger die sizes; also because of the growing number of mobile and battery-operated systems. So, the key requirement is to minimize the interrupts, improve power consumption, reduce possible high clock speed requirements for embedded host controller, provide flexibility and all processing should be done through the centralized location.

Firstly, this thesis presents novel micro-coded instruction-based system architecture of USB 3.0. The micro-coded based design implicitly performs all DMA operations with minimum hardware resources and interrupts; consequently decreasing the architectural complexity, cost, area deployment and dynamic power consumption. Secondly, we propose clock gating in our novel micro-coded system architecture that decreases the dynamic power dissipation by stopping the clock switching activity of the unused functional blocks of the whole architecture. We have applied clock gating technique both on ASIC as well as on FPGA based technology. Experiments shows that by apply clock gating dynamic power consumption decreases up to 12%

with slight area overhead of 2% in ASIC technology; whereas in FPGA dynamic power reduced up to 15% with an 3% increase in area deployment. Furthermore, the designed system architecture increases flexibility as it will be easily extendible to forthcoming standards like PCI express and fiber optic

ACKNOWLEDGEMENT

All prayers and Praises to Almighty **Allah**; The most Beneficent and The most Merciful; Who has always blessed me with His blessings and made my path easier to get successes in my life.

First of all, I am thankful to my beloved family especially my parents for their immense care and love throughout my life. I am also thankful to all of my teachers since my childhood for always providing me the best guidance that a student can get. Among all those most respectful persons for me, I owe my deepest gratitude to my thesis advisor **Dr. Nazar Abbas Saqib** for his support, guidance and help not only in my MS research but also related to other issues, I faced during the time I worked with him on my thesis. One simply could not wish for a better or friendlier supervisor than him.

I am also highly obliged and grateful to **Mr Afzal Khan, Director Engineering, Palm Chip and Consultant USB3.0 Project**, for his unconditional and boundless support in completing my research work. He gave me number of ideas, and helped me time to time in number of ways. I am also grateful to my Thesis Co-advisor **Dr. Rehan Hafiz** for giving his precious time to me and also assisted me in my first paper draft. I am also thankful to my thesis Committee members **Dr. Osman Hassan and Mr. Nasir Mehmood** in helping me and sparing time for me

I also thank to my professional friends especially my CEFAR lab friends for always backing me in every step. I also acknowledge the scholarship committee of National University of Sciences and Technology (NUST) for their financial help during my MS course work. I also thankful to Dr. Arshad Ali and Dr. Nazar Abbas Saqib for hiring me as a **Professional Researcher & Developer** to financially support my M.S. thesis work.

Finally, I thank all those who helped me in my life, may Allah always blessed them with HIS blessings.

TO MY LOVING FAMILY

Table of Contents

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Importance of Efficient System Architecture

Emerging communication standards support ultra high data rates: Universal Serial Bus (USB) 3.0 supporting 4.8 Gbps, PCI express providing 5Gbps and Intel's forthcoming light peak supporting 10Gbps are some of the examples [1]. Both streaming and non-streaming high-end digital devices such as HD cameras & webcams, mass storage devices employing these communication standards therefore demand efficient system architecture for transfer of data to and from the internal peripheral interfaces, or from one location in the external memory to another or to and from system RAM to external FIFO which provides the required bandwidth with as little processing overhead for CPU as possible. Generally Direct Memory access (DMA) technique allows a peripheral to transmit data to or from system RAM without the intervention of CPU [2].

In the context of USB 3.0 system architecture; if DMA is chosen as a main controller, one needs to program the DMA registers for transactions whether these transactions are descriptor based or without descriptor based. Also to support multiple devices and to minimize interrupts in the whole architecture; there is a need of some sort of molded DMA architecture to handle multiple devices through descriptors and/or buffer or using multiple channel devices that will have the area as well as power plenty. Other option may be to use the high clock speed for the processor to service the multiple devices using single channel and set of registers in that channel but this option may not available in the embedded design where the CPU clock speed is generally running at few hundred MHz. Another disadvantage of using the explicit DMA approach is that development engineer get stuck with the notion of the interrupts and defining the DMA as separate entity i.e CPU interface and its registers are located in one place, whereas the DMA and its registers in other location. Also to support segmented memory, DMA controller also need scatter gather operation [2].

We propose a novel micro-coded instruction based architecture. In our designed architecture, the idea is to minimize the interrupts, get away from the possible high clock speed requirements for embedded host controller, get away from the fixed structures, provide flexibility and all processing should be done through the centralized location. To get away from the interrupt based architecture, a molded DMA scheme has been proposed by defining data structure for the transactions in the memory. This scheme reduces the clock speed as required for embedded system and gets to minimize the area of the overall design through providing support for multiple devices in memory data structures for the transactions. To achieve these goals, instruction based architecture has been designed. There are generalized instructions that can decode, encode, move, load, and store these memory data structure. Micro coded instruction based architecture provides support up to maximum permissible devices required by USB 3.0 specification. Hence interrupt based approach has been mitigated that reduce the overall power burden and overall clock speed that may require for supporting multiple devices especially in embedded design.

In this paper we propose a novel micro-coded instruction based architecture. In our designed architecture, the idea is to minimize the interrupts, get away from the possible high clock speed requirements for embedded host controller, get away from the fixed structures, provide flexibility and all processing should be done through the centralized location. To get away from the interrupt based architecture, a molded DMA scheme has been proposed by defining data structure for the transactions in the memory. This scheme reduces the clock speed as required for embedded system and gets to minimize the area of the overall design through providing support for multiple devices in memory data structures for the transactions. To achieve these goals, instruction based architecture has been designed. There are generalized instructions that can decode, encode, move, load, and store these memory data structure. Micro coded instruction based architecture provides support up to maximum permissible devices required by USB 3.0 specification. Hence interrupt based approach has been mitigated that reduce the overall power burden and overall clock speed that may require for supporting multiple devices especially in embedded design. Also, to minimize the power consumption of the whole architecture, root clock gating technique has been proposed. This technique shut-off the clock to the functional blocks that are not used at some particular clock cycles.

1.2 Motivations

Most of the system architecture present in literature survey has been designed by using direct memory access (DMA) like USB, PCI express, digital signal processing (DSP) application and so on. Traditionally, most of the DMA based system architecture has been designed by considering the effects of memory transaction speed, memory access latency, DMA write and DMA read effective throughput. But there is no significant work on area and power efficiency on DMA based system architectures. When we design any system architecture, there are some overheads of DMA with respect to computational complexity, area deployment and power consumption. Also the DMA based system architecture is application specific. It is not scalable to all forthcoming express standards. Also there is a demand of power optimizes design that will dissipate lesser power and up to some extent increases the battery life and make them portable.

To handle the above mention issues, there is a need to design power efficient system architecture for express standard USB 3.0 that removes the discrepancies of DMA. Also the designed system architecture will be scalable to forthcoming standards PCI express and Intel light peak.

1.3 Problem Statement and Thesis Objectives

The problem statement of this thesis can be stated as follows

“To design a scalable low power and area efficient system architecture for ultra speed data transaction extended to USB 3.0”

The main objectives of this thesis are as under

- To design an effective system architecture for express standards like USB 3.0

There are also some specific objectives of this thesis that are as under

- Study of existing DMA based system architectures
- Design/propose novel system architecture
- Implementation and simulation verification of system architecture
- Measurement of Performance parameters
- Comparison with and without clock gating

1.4 Thesis Contributions

The thesis exhibits the power efficient design of novel micro-coded instruction based system architecture extended to USB 3.0 protocol by considering the effects of area deployment and power overheads. Furthermore this thesis also demonstrates the effect of root clock gating to novel system architecture to make it more power efficient.

1.4.1 Problem Breakdown

We have divided the problem statement of our thesis into following discrete objectives.

- **Study of existing DMA based system architectures of express standards**

A number of system architectures have been designed over time to time for different express standards like PCI express, DSP applications and Input /output system buses. In these system architectures, efficiency of the system architecture has been shown by considering different performance parameters like DMA read/write effective throughput, memory access latency, memory transaction speed and other factors like area deployment and power consumption . Our goal is to study all related system architectures of express standards and how efficient their working with respect to different performance parameters is?

After a thorough and critical study of previously presented system architectures we will enlist the discrepancies and limitations of these system architectures. This list of discrepancies will set the goals of our research.

- **Design/propose novel system architecture**

Keeping in view the limitations of existing system architectures of express standards, we will design novel system architecture to mitigate the discrepancies of previous system architectures. First of all we will design area and power efficient micro-architecture of USB 3.0 protocol. Based on USB 3.0 communication protocol; we will make a high level data path and control logic of the whole system architecture.

- **Implementation and simulation verification of system architecture**

When micro-architecture of the whole system has been designed; we will write an RTL description of different modules based on the micro-architecture. We will design the control and data path logic of the whole communication protocol separately. After that we will implement our novel system architecture on simulation software and run different USB 3.0 test cases like bulk write transaction, bulk read transaction and so on. We will verify the data path and control logic of the whole design. In test cases, we will check all normal and erroneous cases of the communication protocol.

- **Measurement of Performance parameters**

When the whole system architecture has been implemented successfully and it is functionally verified on different simulation software; we will evaluate different performance parameters of the whole system architecture like toggle factor, no of gate counts and dynamic power dissipation. Based on the switching activity and number of gate counts of different modules, we will calculate the dynamic power consumption of the individual modules as well as overall system architecture.

- **Comparison with and without clock gating**

We propose power optimization in our whole system architecture; that makes the whole system architecture further power efficient. Root clock gating is an RTL power optimization technique that disables the clock to functional blocks that are not used during some particular clock cycles. We will calculate the dynamic power dissipation with and without root clock gating and provides a comparative analysis of the system architecture with and without root clock gating.

1.5 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 provides the background knowledge for Universal serial bus (USB) 3.0 communication protocol and general direct memory access (DMA) based system architecture flow. Chapter 3 provides contemporary system architectures of express standards; their contribution, strength and weakness. Chapter 4 provides the novel micro-coded instruction based system architecture. Chapter 5 provides description of the tools, software’s and methodology used in the experimentation of our novel system architecture. Chapter 6 provides the performance results of the designed system architecture with and without root clock gating; Chapter 7 is the final chapter of this thesis and provides conclusion and future directions.

CHAPTER 2

BACKGROUND

This chapter provides the necessary background knowledge of the universal serial bus (USB) 3.0 communication protocols so that the reader can have a better understanding of the USB 3.0 architecture and key concepts. It also briefly describes how the memory data transaction takes place between system software and input/output device using direct memory access. I also briefly describe the issues of DMA based system architectures for express standards like USB 3.0.

2.1 USB 3.0 (SuperSpeed) Communication Protocol

USB 3.0(SuperSpeed) provides broad compatibility, affordability and plug n play behavior. USB 3.0 is similar to earlier versions of USB in that it is a cable bus supporting data exchange between a host computer and a wide range of simultaneously accessible peripherals. The attached peripherals share bandwidth through a host scheduled protocol. The bus allows peripherals to be attached, configured, used and detached while the host and other peripherals are in operation. USB 3.0 utilizes a dual bus architecture that provides backward compatibility with USB 2.0[1].

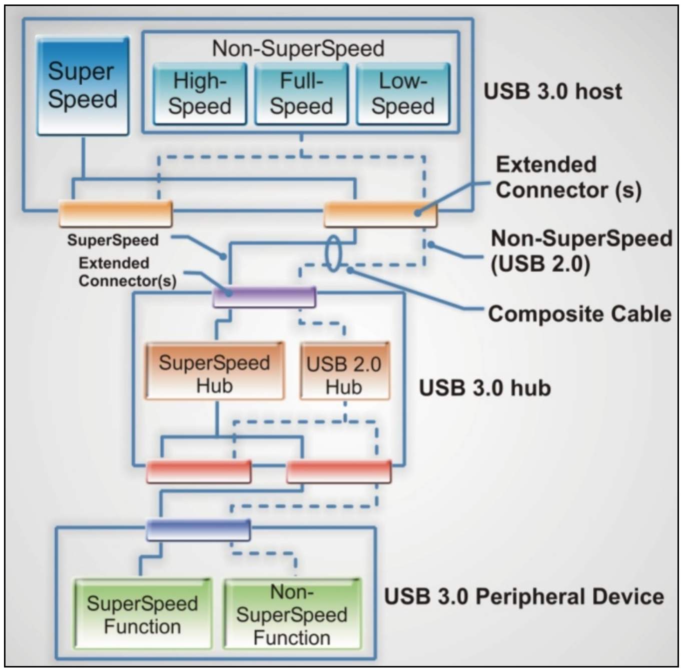


Figure 2.1: USB 3.0 Internal Architecture

USB 3.0 provides data rate up to 5 Gbps which is ten times more than the previous version of USB 2.0. There are dual simplex transmission channel present in USB 3.0; which means that there are separate paths for transmission and reception. In USB 3.0 data transfer is manage using endpoints. Endpoint is a uniquely addressable portion of a USB device that is either a source or sink of information. Endpoints are characterized by the type of data transfer supported by the protocol.

USB 3.0 is a layered communication architecture that is comprised of the following components [1]

- **USB 3.0 Interconnect:** USB 3.0 interconnect is the topology in which devices are connected to and communicate with the host.
- **Devices:** USB 3.0 devices are sources or sinks of information exchanges. They implemented the required device-end, USB 3.0 communications layers to accomplish

information exchanges between a driver on the host and a logical function (endpoint) on the device.

- **Host:** A USB 3.0 host is a source or sink of information. It implements the required host-end, USB 3.0 communications layers to accomplish information exchanges over the bus. It holds the USB 3.0 data activity schedule and management of the USB 3.0 bus and all devices connected to it.

USB 3.0 Communication protocol has following communication layers [1]

1. Physical layer
2. Link layer
3. Protocol layer

2.1.1 Physical layer

The physical layer defines the PHY portion of a port and the physical connection between a downstream port and the upstream facing port on a device. The USB 3.0 physical connection is comprised of two differential data pairs, one transmit path and one receive path. The nominal signaling data rate is 5Gbps.

2.1.2 Link layer

Link layer provides device connect device removal, link power management, flow control, error control and inserting of framing sequences (i-e packet delimiters).

2.1.3 Protocol layer

USB 3.0 protocol layer defines “end-to-end” communication between host and device. USB 3.0 protocol provides application data information exchanges between a host and a device endpoint. It is a host-directed protocol, which means the host determines when application data is transferred between host and device.

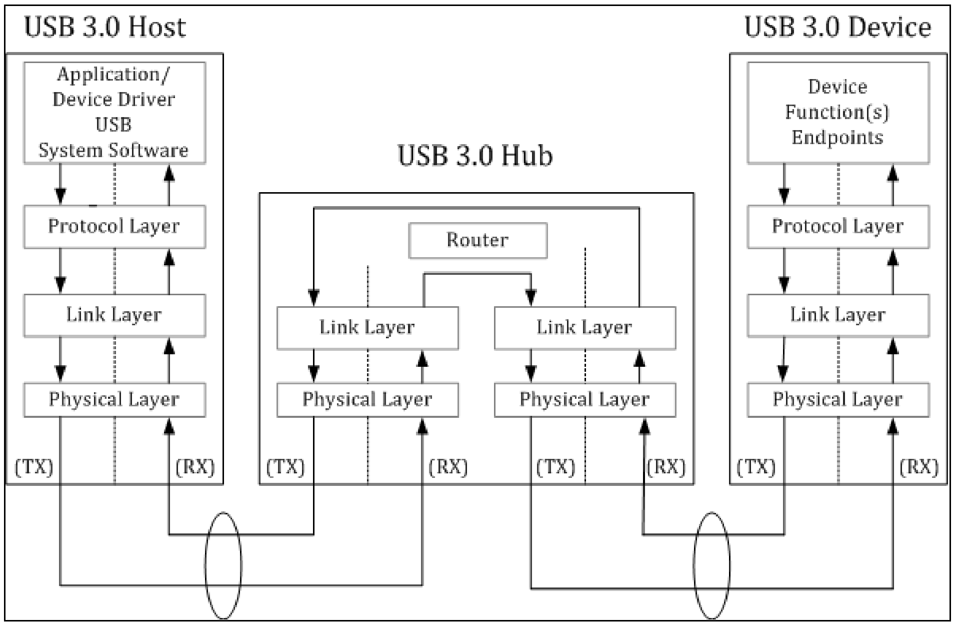


Figure 2.2: Communication Flow of USB 3.0

USB 3.0 protocol supports four types of data transaction modes: bulk, control, isochronous and interrupt. Bulk transfer mode is used when massive amount data is required to transmit between host and device. Interrupt transfer mode supports when an urgent data is required to be sending to the device through some interrupt. Isochronous transfer mode sends real time data between host and device/peripheral. Similarly, control transfer mode allows device initialization and configuration [1]. Endpoints transfer data in one direction so there are separate endpoints for transmit (Tx) and receive (Rx) transfer. The only exception is “default endpoint” that provides access to device configuration information. A maximum of 15 endpoints can be supported by each device [1]. USB 3.0 core register block includes all registers that are used in performing USB 3.0 functions inside the USB 3.0 core. Endpoint control register is used for endpoint configuration. It contains the information regarding transfer direction, transfer type, endpoint number, payload size etc [1].

2.2 Direct Memory Access (DMA)

DMA technique allows a peripheral to transmit data to or from the system RAM without the

CPU intervention. Many hardware systems use DMA including disk drive controllers, graphics cards, network cards and sound cards. There is a system bus interface between DMA controller, system CPU and system RAM that allows the data to be read or write between system RAM and local memory of connected peripheral [2]. System bus has a master as well as slave interface. Master Interface of system bus requests the grant of system bus. Slave interface is used to move data payload and device descriptors through system RAM. There is also a bus arbiter that grants the system bus to the DMA controller based on some priority for data transactions i-e device initialization and configuration has highest priority over transaction. Similarly periodic transactions have highest priority over non-periodic data transactions. DMA system architecture contains the following components [2]:

- System CPU
- System RAM
- DMA Controller
- I/O System Bus
- I/O peripheral device

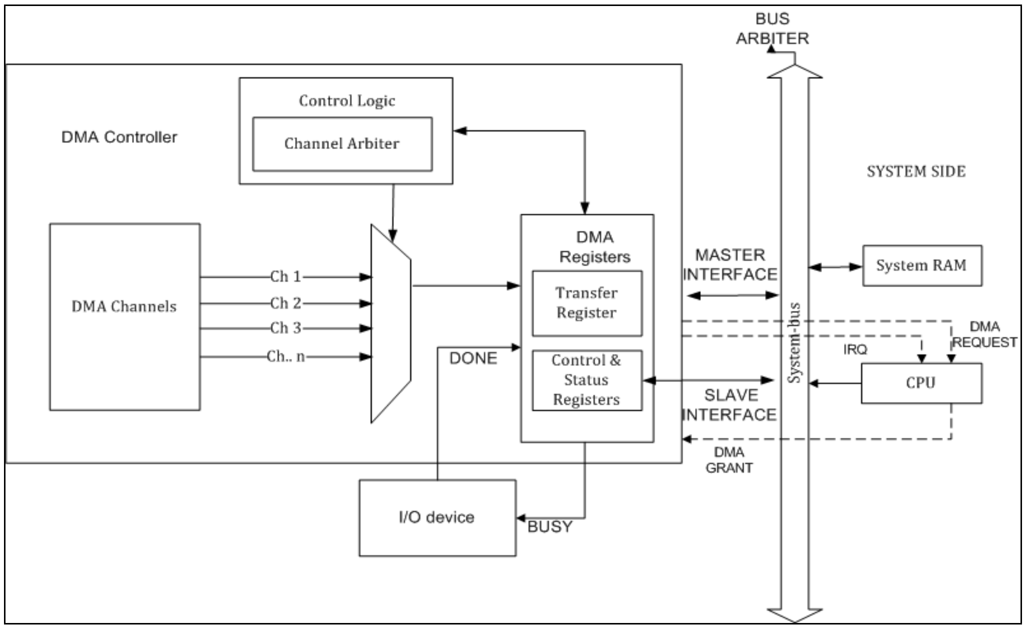


Figure 2.3: DMA Controller System Architecture

DMA registers contain control, status and transfer registers. Control registers contain the information related to device initialization and configurations. Transfer registers enclose the information related to particular transaction of specific device. Status registers contain the status of the current transaction. There are multiple channels in DMA controller that allows multiple devices/peripheral to communicate with the system software. Each channel is allocated to one particular device. Also each DMA channel has its own set of request lines. There is channel arbiter that selects the particular channel based on the DMA control and transfer register program by the system CPU. Also, each channel has its own set of interrupts controller and operational registers [2].

2.2.1 DMA Transfer

DMA transfer has three different distinct stages:

- Initialization Stage
- Transfer Stage
- Status Stage

2.2.2 Initialization Stage

During initialization stage, CPU programs the DMA control registers through slave interface of system bus. The configuration of these registers indicates that there are some memory data transactions or configuration need to be started. After that DMA controller sends DMA REQUEST signal to the CPU through slave interface. CPU then grants the system bus to the DMA controller and also set DMA GRANT signal. Now CPU does not communicate any more with the peripheral and waits a transaction completion interrupt from the DMA to communicate further with the peripheral [2].

2.2.3 Transfer Stage

In transfer stage, when the system bus is granted to DMA controller; it first programs the transfer registers of DMA controller and also through channel arbiter, choose the particular channel to which data is transmitted to the particular device. Then DMA controller set the BUSY signal for the device and sends the memory data from system RAM to the local memory of I/O device/peripheral. When the whole data is transmitted successfully to the device/peripheral, device acknowledge through DONE signal given to the DMA controller [2].

2.2.4 Status Stage

In status stage, DMA controller sends an interrupt completion request to the CPU through IRQ signal. Also the DMA releases the grant of system bus and waits other transactions initiated by the CPU [2].

2.3 Issues in DMA based System Architecture for USB 3.0

Since DMA interface is used for memory data transaction in different system architectures. But there are some overheads of DMA based system architecture for express standards like USB 3.0 some of these are as follows:

1. Explicit DMA registers are required for transactions
2. To handle multiple devices, there is a need of buffer or multiple DMA channels
3. High clock speed requirement to support multiple devices using single channel
4. Interrupt based system architecture in which CPU is dependent on DMA controller
5. To support segmented memory, DMA scatter gather complex operation is required

2.3.1 DMA Scatter Gather Operation

In system RAM, data transaction memory is not allocated continuously. It is segmented memory and there is a need of some extra logic to handle all the segments sequentially. In DMA based system architecture, DMA scatter gather complex operation is required to implement the whole hierarchy [2].

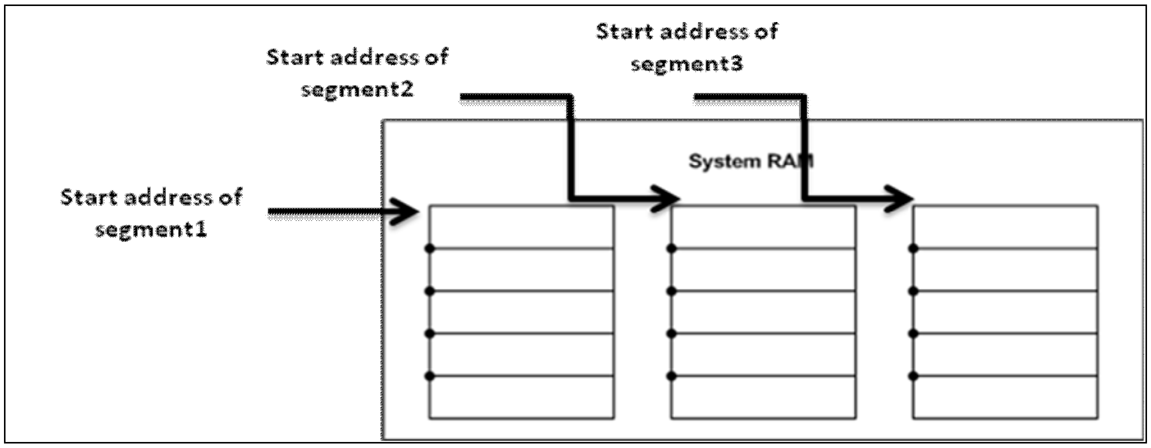

Start address of segment2
Start address of segment3
System RAM
Start address of segment1

Figure 2.4: DMA Scatter Gather Operation

In scatter gather operation, start addresses, size of each memory segment and sequence number of all segments are required. When the first segmented is read/write successfully, we will move to next memory segment using scatter/gather operation. So, this operation is computation complex and has some area as well as power overhead in any system architecture.

2.4 Performance Parameters of System Architecture

This section explains some metrics which can be used to evaluate the performance of the designed system architecture.

2.4.1 Dynamic Power Dissipation (P)

Dynamic power dissipation describes how much power is dissipated of a module or functional block due to switching activity of clock signal and data signals, operating frequency, voltage and number of gate counts. The dynamic power dissipation of any system architecture depends upon the following equation [11].

$$P(W) = n * C * V^2 * f \qquad \text{.................} \qquad (2.1)$$

Where n is the toggle factor or switching activity factor, C is the capacitance or the number of gate counts, V is the operating voltage and f is the operating frequency. The above relation shows that if we increase either n, C or f dynamic power dissipation increases linearly. Similarly, if we increase V, dynamic power dissipation increases exponentially [9][11].

2.4.2 Toggle factor (n)

Toggle factor is the switching activity of a module or any functional block. If the module or functional block toggles on every clock cycle, it means toggling will be 100% of that module. Generally, toggling of the functional block or module lies in between 40-60%. More the toggle factor more will be the dynamic power dissipation [14].

2.4.3 Capacitance (C)

Capacitance is the number of the gate count or area deployment of any module or functional block. It depends upon the logic, memory size and number of registers of the functional module or block. Higher the value of C, higher will be the number of dynamic power dissipation of the designed system architecture [14].

2.4.4 Operating Voltage (V)

Operating voltage is the specific voltage used in the whole designed architecture. Generally it is application specific like FPGA board or ASIC library. Higher the operating voltage, higher will be the dynamic power dissipation [14].

2.4.5 Operating frequency (f)

Operating frequency is the clock frequency that is used in the whole system architecture. It is also application specific like USB 3.0 has different operating frequency, PCI needs different operating frequency, and other express standards has its own operating frequency to meet the design requirement. Higher clock frequencies are not desirable as they increase the dynamic power dissipation [14].

CHAPTER 3

LITERATURE SURVEY

This chapter explains the state of the art designed system architectures of express standards like PCI express, DSP processors, I/O system buses and so on. Also this chapter explains clock gating power optimization techniques at register transfer level.

3.1 System Architecture

Nowadays, as technology advances there is demand of high data rate, lowest latency, minimizes area deployment and efficient power consumption [4]. To fulfill these demands, there is a need of efficient system architecture design. System architecture provides the control and data path flow to transmit data between different peripherals or between host and any particular device. Also to communicate with the system RAM there is a specific interface of system bus and controller required to interact between hardware and software [5]. We will explain the whole system architecture and its required components with details in chapter 5.

3.2 Related Work

Traditionally DMA based system architecture has been designed for different express standards. For architectural merits, USB 3.0 is becoming more and more popular in high speed communication [1]. However, no comprehensive work has been found on USB 3.0 system architecture. Generally, Peripheral Component Interconnect (PCI) Express system architecture has been designed using DMA as a main controller to access data payload from the system memory [4]. Although USB 3.0 communication protocol to some extent is related to PCI express communication protocol; so we may design DMA based system architecture of USB 3.0 but due to DMA limitations it will not be efficient system architecture [1] [4]. Following are the some state of the art system architecture that has been designed for express standards.

- DMA System Architecture for DSP Applications [3]:
- DMA Transaction Method for PCI Express Devices [4]:
- DMA Cache based system architecture for I/O system buses [5]

3.2.1 DMA System Architecture for DSP Applications [3]

This system architecture has been designed for DSP based applications. In this system architecture, the whole system architecture has been divided into several modules. For different applications, some of the modules are operational while other modules remain inactive [3]. So modularity of the design allows all DMA operations to be added or removed very easily. This system architecture is scalable for all DMA transactions [3].

This designed system architecture performs data transactions at much faster data rate as compare to conventional DMA system architectures. This system architecture also maximizes the usage of all interfaces of each peripheral [3]. There are some drawbacks of this system architecture. Firstly, scatter gather complex operation requires for non-segmented memory. Also this system architecture has some area as well as power overheads because of DMA controller. Secondly, this system architecture performs plenty of memory data transactions so there is an issue of bus contention that significantly reduces the effective bandwidth.

3.2.2 DMA Transaction Method for PCI Express Devices [4]

This system architecture has optimized PCI express transaction through improving both bus-efficiency and DMA efficiency. This system architecture reduces the issue of bus contention that reduces effective bandwidth of the PCI express as in DSP based system architecture [3]. In this system architecture authors has proposed an FSM based test method to improve data transaction speed [4].

This system architecture has improved the DMA write and DMA read transaction speed. This method is equally applicable to PXI-Express applications and so forth. This designed system architecture has some flaws. This method is only applicable for non-segmented memory (continuous). Scatter gather complex operation is required for non-segmented memory that increases area and make it complex [4].

3.2.3 DMA Cache based system architecture for I/O system buses [5]

In this system architecture the authors have presented two DMA cache techniques to store I/O data in dedicated on chip storage namely Decoupled DMA Cache (DDC), and Partition-Based DMA Cache (PBDC). Decoupled DMA Cache (DDC) contains DMA cache as buffer to store I/O data. Partition-Based DMA Cache (PBDC) does not contain DMA cache as buffer instead uses dynamically processor last level cache (LLC) as DMA cache. This scheme has reduced memory access latency up to 34.8% for DDC and 80% for PBDC. Similarly, DMA overhead reduces the overall efficiency of system architecture [5].

3.3 Power Optimization

Promising high speed communication protocols need power efficient system architectures. The importance of the low power has been come into sight from the last few years. If there is no constraints on power utilization then it will increases up to 1,000's of watts as shown in figure-1 so we have to limit this power consumption up to some extent. There are lots of applications like cell phones; laptops, personal digital assistant (PDA) and all wireless devices need power efficient system design. These application demands high battery life that make it reliable. Also low power design/devices have an ability to provide lots of features to different application. High power dissipation leads to expensive packaging and cooling systems. Furthermore high power dissipation will minimize the product life [8].

There are different levels of power optimization namely system level, algorithm level, register transfer level, gate level and transistor level. We have employed register transfer level power optimization technique in our novel system architecture. Register transfer level provides 20-30% reduction in dynamic power utilization [11]. There are numerous power optimization techniques that have been proposed in register transfer level (RTL).

Clock gating is the most fundamental and efficient technique at RTL. It is mostly used in clock sequential circuits [10]. There is some of the state of the art clock gating technique that has been previously employed. These are:

- Enable Function Based Clock Gating [15]
- RTL Clock Gating Based on ODCs [16]
- Zero Skew Clock Gating [17]
- RTL Clock Gating with Emphasis on Glitch Analysis and Reduction [18]
- Root Clock Gating [10]

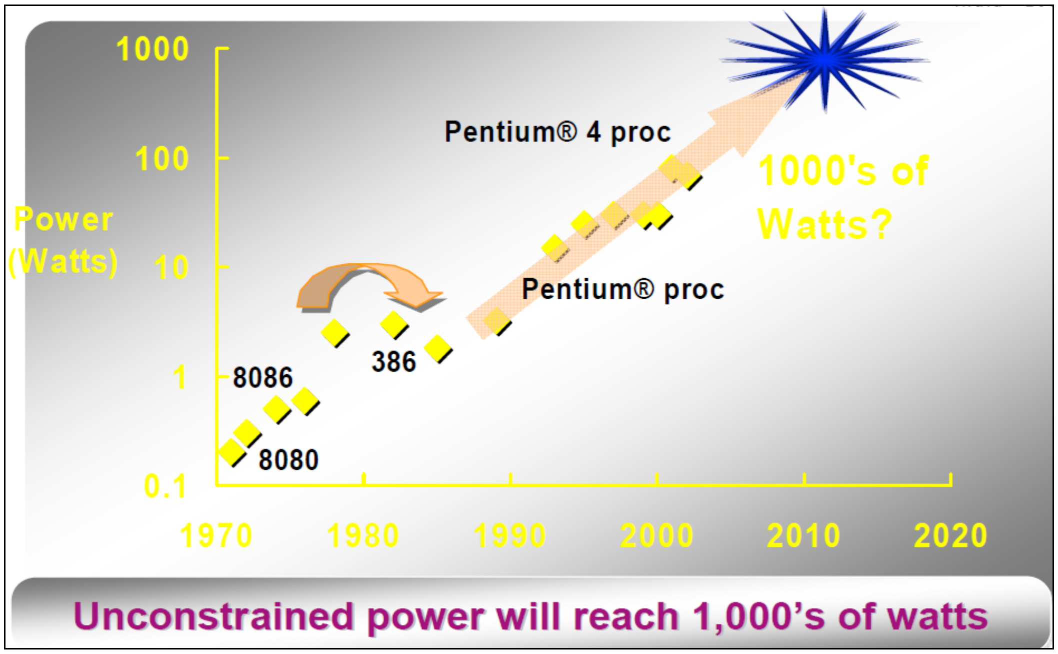


Figure 3.1: Importance of Power Dissipation

3.3.1 Enable Function Based Clock Gating [15]

This technique has provided the improved clock gating scheme as it is formulated to find the optimal set of enable function in the shared logic cone that leads to best power reduction on flip flops [15]. This technique added only small amount of area overhead and reduces dynamic power significantly. Due to clock gating circuitry itself, this technique shouldn’t be applied on small RTL circuit [15].

3.3.2 RTL Clock Gating Based on ODCs [16]

This technique provides effective clock-gating schemes by creating an activation function that can stop the clock to set of registers for a significant fraction of cycles; when the register outputs are unobservable [16]. This contribution provides an excellent algorithm for automatic detection of complex idle conditions. It is also very suitable for integration into industrial design tools [16]. ODC based approach shows a slight penalty on area and delay with respect to original circuit [16].

3.3.3 Zero Skew Clock Gating [17]

This technique minimizes the skew (delay) in clock tree synthesis of synchronous sequential circuits by adding redundant gates to synchronize clock [17].This novel clock tree design style to ensure that the logic gates at the same level are in the same type and significantly reduces the clock skew [17]. There is slight penalty of area and dynamic power consumption [17] on the whole system design.

3.3.4 RTL Clock Gating with Emphasis on Glitch Analysis and Reduction [18]

This technique reduces glitching power consumption in RTL circuits by minimizing the propagation of glitches through different blocks of the circuit. This technique is also used at register transfer level (RTL) that analyzes the generation and propagation of glitches for controller/data path circuits [18]. It is an automated power reduction technique of clock gating at RT level that avoids propagation of glitches on control signals [18].

3.3.5 Root Clock Gating [10]

This technique inserts gating logic at the root of the clock to save the power consumed by the clock network feeding these instances [10]. This is a novel solution for root gating optimization that further reduces the clock tree power [10]. The main disadvantage of this technique is that the fragmentation of the clock tree into several clocks gating domain makes skew balancing very difficult [10].In this technique, possible no of candidates for root gating grows exponentially with the no of clock gating logic in the design [10]

The main purpose to elaborate these clock gating techniques are that because we have designed customized clock gating technique based on USB 3.0 communication protocol and the characteristics of the designed system architecture. We will explain this customized clock gating technique in chapter 5

CHAPTER 4

EXPERIMENTAL SETUP

This chapter explains how we have designed our test cases; apparatus and tools that have been used to conduct implementation, functional verification, performance evaluation of the designed system architecture.

4.1 Selection of Tools

For implementation and verification purpose, both ASIC and FPGA based system tools have been used. In ASIC, we used UMC 90nm ASIC technology (FSDOA_A Core Cell) library and for FPGA, we used Xilinx Virtex-5 board [9] [13].

4.2 UMC 90nm ASIC Technology library

The power consumption characteristic of this specific library is 4.5nW/MHz/gate [13]. This means that if a single gate is operated on 1 MHz frequency; it will consume 4.5nW dynamic power. Operating voltage of the whole system design in this technology library is 1Volt. Since the USB 3.0 protocol demands 4Gbps data rate. So the operating frequency of the designed system architecture is 250MHz (equation 1). Operating frequency depends upon the expected throughput. Also the power consumption by using UMC 90 nm technology is:

$$P(nW) = n*4.5*C*V^2*f \quad \ldots\ldots\ldots\ldots \quad (4.1)$$

$$f\ (Mhz) = 1/expectedthroughput\ (Gbps) \quad \ldots\ldots\ldots\ldots\ldots \quad (4.2)$$

Where n is the toggle factor; C is capacitance or number of gate counts, V is the operating voltage; and f is the operating frequency.

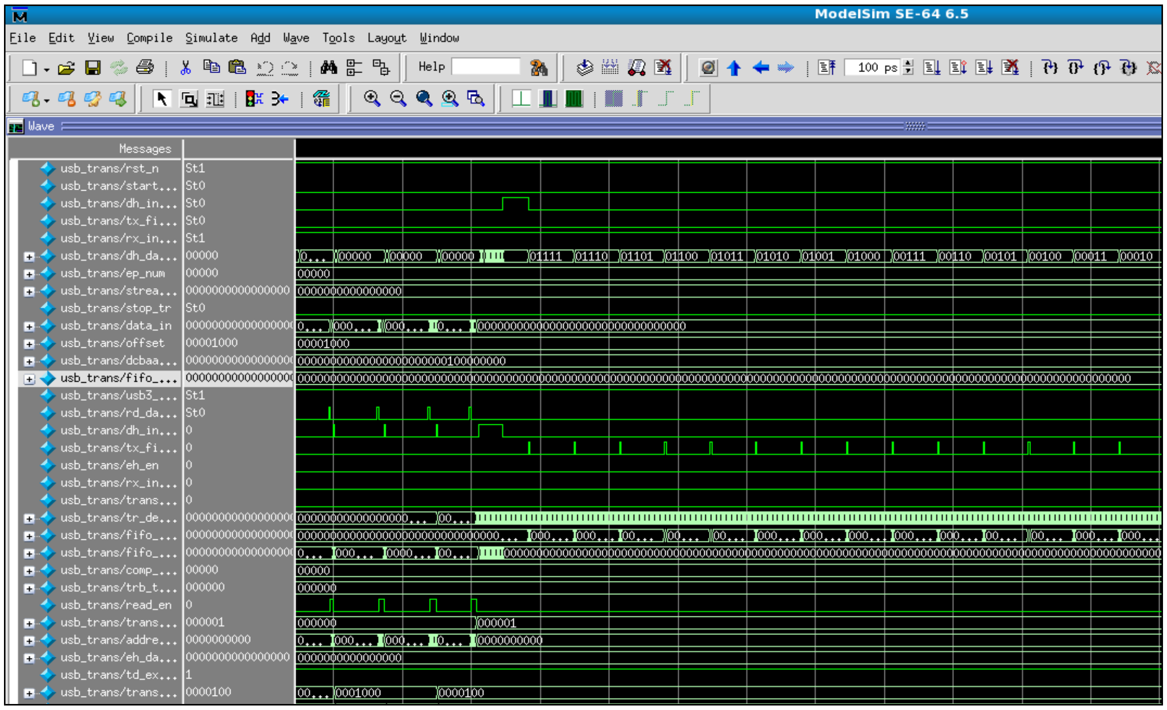
4.2.1 Tools for ASIC Technology

Functional verification of the designed system architecture has been performed in ModelSim 6.5 SE. we have run different test cases of the USB 3.0 communication protocol in modelsim and verify the data path of the whole system architecture. For area deployment estimation, we have used Cadence Encounter RTL Compiler. Similarly for power estimation, Cadence Encounter RTL Compiler has been used [14].

4.2.2 Area Deployment Estimation & Dynamic Power Estimation

Following steps have been performed to calculate the area deployment, number of gate counts and dynamic power consumption of different modules of USB 3.0 on ASIC tools [14].

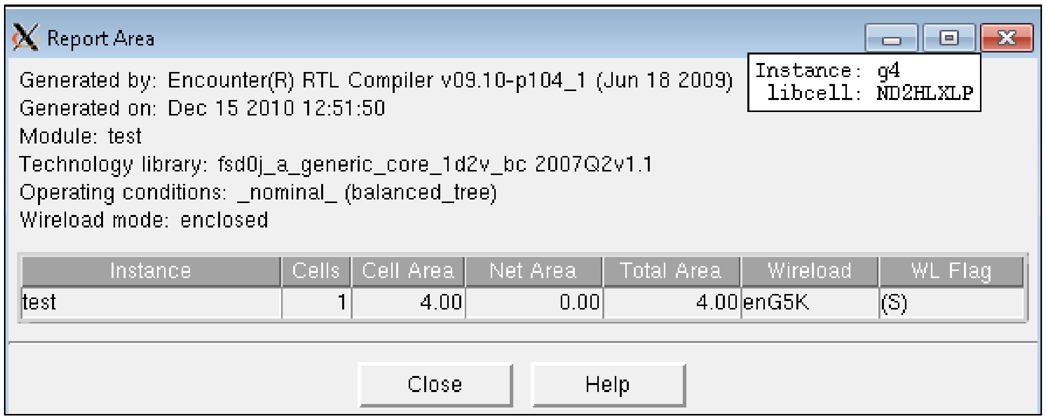
- Read an RTL of the specific module in Cadence RTL Compiler
- Add specific ASIC technology library (UMC 90nm)
- Synthesize the specific module


ModelSim SE-64 6.5
File Edit View Compile Simulate Add Wave Tools Layout Window
Help
100 ps
Wave
Messages

```
usb_trans/rst_n      St1
usb_trans/start...   St0
usb_trans/dh_in...   St0
usb_trans/tx_fi...   St0
usb_trans/rx_in...   St1
usb_trans/dh_da...   00000
usb_trans/ep_num     00000
usb_trans/strea...   0000000000000000
usb_trans/stop_tr    St0
usb_trans/data_in    0000000000000000
usb_trans/offset     00001000
usb_trans/dcbaa...   0000000000000000
usb_trans/fifo_...   0000000000000000
usb_trans/usb3_...   St1
usb_trans/rd_da...   St0
usb_trans/dh_in...   0
usb_trans/tx_fi...   0
usb_trans/eh_en      0
usb_trans/rx_in...   0
usb_trans/trans...   0
usb_trans/tr_de...   0000000000000000
usb_trans/fifo_...   0000000000000000
usb_trans/fifo_...   0000000000000000
usb_trans/comp_...   00000
usb_trans/trb_t...   000000
usb_trans/read_en    0
usb_trans/trans...   000001
usb_trans/addre...   0000000000
usb_trans/eh_da...   0000000000000000
usb_trans/td_ex...   1
usb_trans/trans...   0000100
```

Figure 4.1: Simulation of USB 3.0 modules in ModelSim

- In area report, read the no of gate counts being used by this specific module

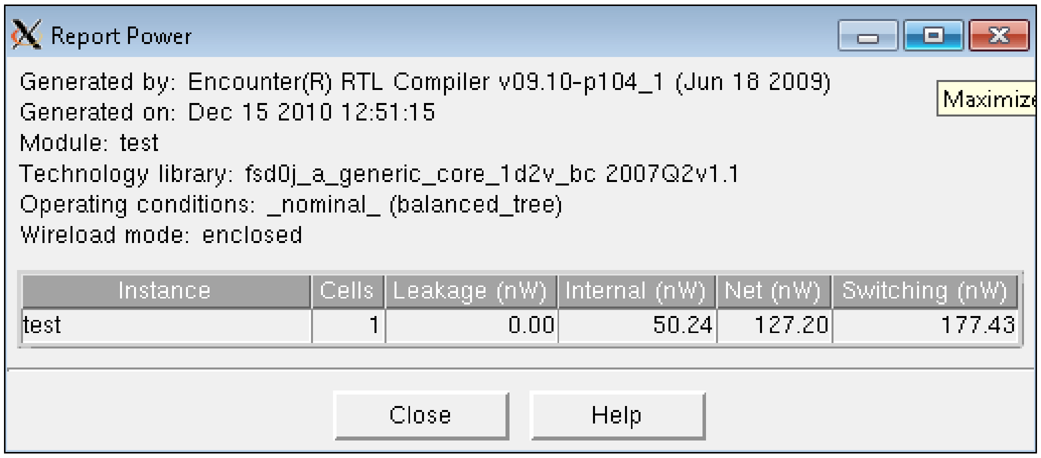


```
Generated by: Encounter(R) RTL Compiler v09.10-p104_1 (Jun 18 2009)
Generated on: Dec 15 2010 12:51:50
Module: test
Technology library: fsd0j_a_generic_core_1d2v_bc 2007Q2v1.1
Operating conditions: _nominal_ (balanced_tree)
Wireload mode: enclosed
```

Instance	Cells	Cell Area	Net Area	Total Area	Wireload	WL Flag
test	1	4.00	0.00	4.00	enG5K	(S)

Figure 4.2: Area Estimation on Cadence RTL Compiler

- In power report, read switching power of this specific module



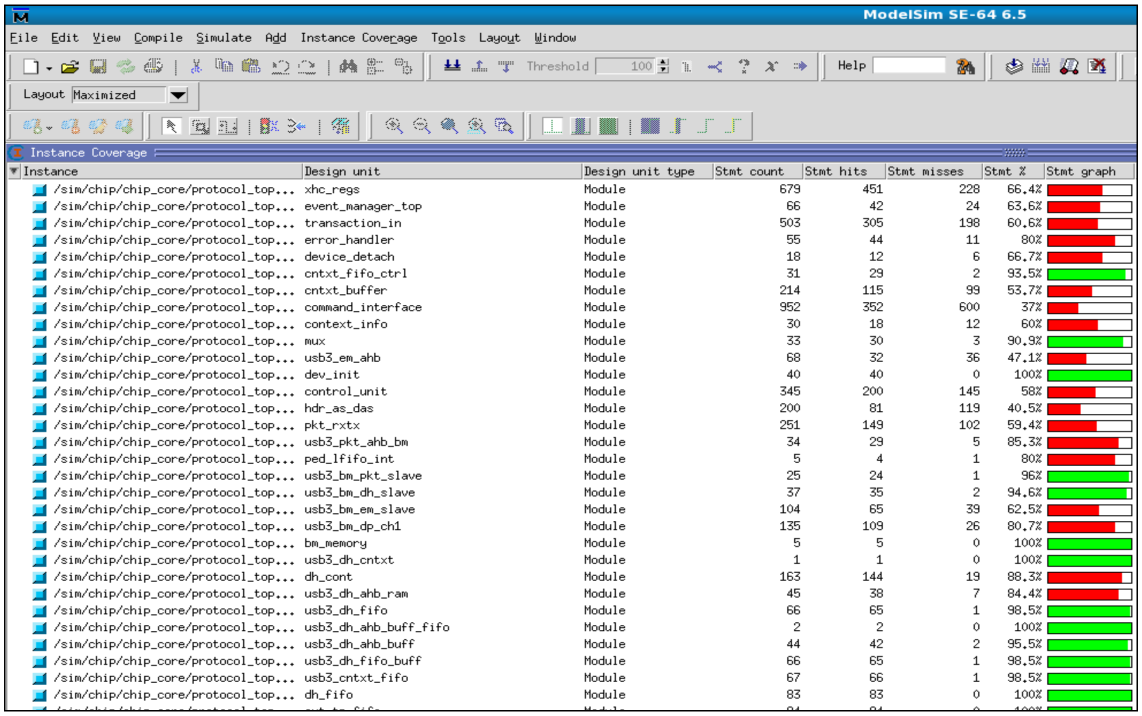
```
Generated by: Encounter(R) RTL Compiler v09.10-p104_1 (Jun 18 2009)
Generated on: Dec 15 2010 12:51:15
Module: test
Technology library: fsd0j_a_generic_core_1d2v_bc 2007Q2v1.1
Operating conditions: _nominal_ (balanced_tree)
Wireload mode: enclosed
```

Instance	Cells	Leakage (nW)	Internal (nW)	Net (nW)	Switching (nW)
test	1	0.00	50.24	127.20	177.43

Figure 4.3: Power Estimation on Cadence RTL Compiler

4.2.3 Toggle factor [14]

To calculate the toggle factor of the whole system architecture or specific module of the specific system architecture, Modelsim code coverage details has been used. Code coverage details include toggle coverage information that tells about the switching activity of the specific module or whole designed system architecture.


ModelSim SE-64 6.5
File Edit View Compile Simulate Add Instance Coverage Tools Layout Window
Threshold 100
Help
Layout Maximized
Instance Coverage

Instance	Design unit	Design unit type	Stmt count	Stmt hits	Stmt misses	Stmt %	Stmt graph
/sim/chip/chip_core/protocol_top...	xhc_regs	Module	679	451	228	66.4%	🟥
/sim/chip/chip_core/protocol_top...	event_manager_top	Module	66	42	24	63.6%	🟥
/sim/chip/chip_core/protocol_top...	transaction_in	Module	503	305	198	60.6%	🟥
/sim/chip/chip_core/protocol_top...	error_handler	Module	55	44	11	80%	🟥
/sim/chip/chip_core/protocol_top...	device_detach	Module	18	12	6	66.7%	🟥
/sim/chip/chip_core/protocol_top...	cntxt_fifo_ctrl	Module	31	29	2	93.5%	🟩
/sim/chip/chip_core/protocol_top...	cntxt_buffer	Module	214	115	99	53.7%	🟥
/sim/chip/chip_core/protocol_top...	command_interface	Module	952	352	600	37%	🟥
/sim/chip/chip_core/protocol_top...	context_info	Module	30	18	12	60%	🟥
/sim/chip/chip_core/protocol_top...	mux	Module	33	30	3	90.9%	🟩
/sim/chip/chip_core/protocol_top...	usb3_em_ahb	Module	68	32	36	47.1%	🟥
/sim/chip/chip_core/protocol_top...	dev_init	Module	40	40	0	100%	🟩
/sim/chip/chip_core/protocol_top...	control_unit	Module	345	200	145	58%	🟥
/sim/chip/chip_core/protocol_top...	hdr_as_das	Module	200	81	119	40.5%	🟥
/sim/chip/chip_core/protocol_top...	pkt_rxtx	Module	251	149	102	59.4%	🟥
/sim/chip/chip_core/protocol_top...	usb3_pkt_ahb_bm	Module	34	29	5	85.3%	🟥
/sim/chip/chip_core/protocol_top...	ped_lfifo_int	Module	5	4	1	80%	🟥
/sim/chip/chip_core/protocol_top...	usb3_bm_pkt_slave	Module	25	24	1	96%	🟩
/sim/chip/chip_core/protocol_top...	usb3_bm_dh_slave	Module	37	35	2	94.6%	🟩
/sim/chip/chip_core/protocol_top...	usb3_bm_em_slave	Module	104	65	39	62.5%	🟥
/sim/chip/chip_core/protocol_top...	usb3_bm_dp_ch1	Module	135	109	26	80.7%	🟥
/sim/chip/chip_core/protocol_top...	bm_memory	Module	5	5	0	100%	🟩
/sim/chip/chip_core/protocol_top...	usb3_dh_cntxt	Module	1	1	0	100%	🟩
/sim/chip/chip_core/protocol_top...	dh_cont	Module	163	144	19	88.3%	🟥
/sim/chip/chip_core/protocol_top...	usb3_dh_ahb_ram	Module	45	38	7	84.4%	🟥
/sim/chip/chip_core/protocol_top...	usb3_dh_fifo	Module	66	65	1	98.5%	🟩
/sim/chip/chip_core/protocol_top...	usb3_dh_ahb_buff_fifo	Module	2	2	0	100%	🟩
/sim/chip/chip_core/protocol_top...	usb3_dh_ahb_buff	Module	44	42	2	95.5%	🟩
/sim/chip/chip_core/protocol_top...	usb3_dh_fifo_buff	Module	66	65	1	98.5%	🟩
/sim/chip/chip_core/protocol_top...	usb3_cntxt_fifo	Module	67	66	1	98.5%	🟩
/sim/chip/chip_core/protocol_top...	dh_fifo	Module	83	83	0	100%	🟩

Figure 4.4: Toggle Coverage in ModelSim

4.3 Xilinx Virtex -5 FPGA [9]

In FPGA based implementation, we leveraged the architectural features of Virtex-5 FPGA. It has divided the FPGA chip internally into clock zones for clock distribution. So we can cut off the clock to whole clock region instead of a single FF. This leads to better power reduction. Clock is controlled through Global Clock Buffers (there are 32 global clock buffers in virtex-5 FPGA) as in figure

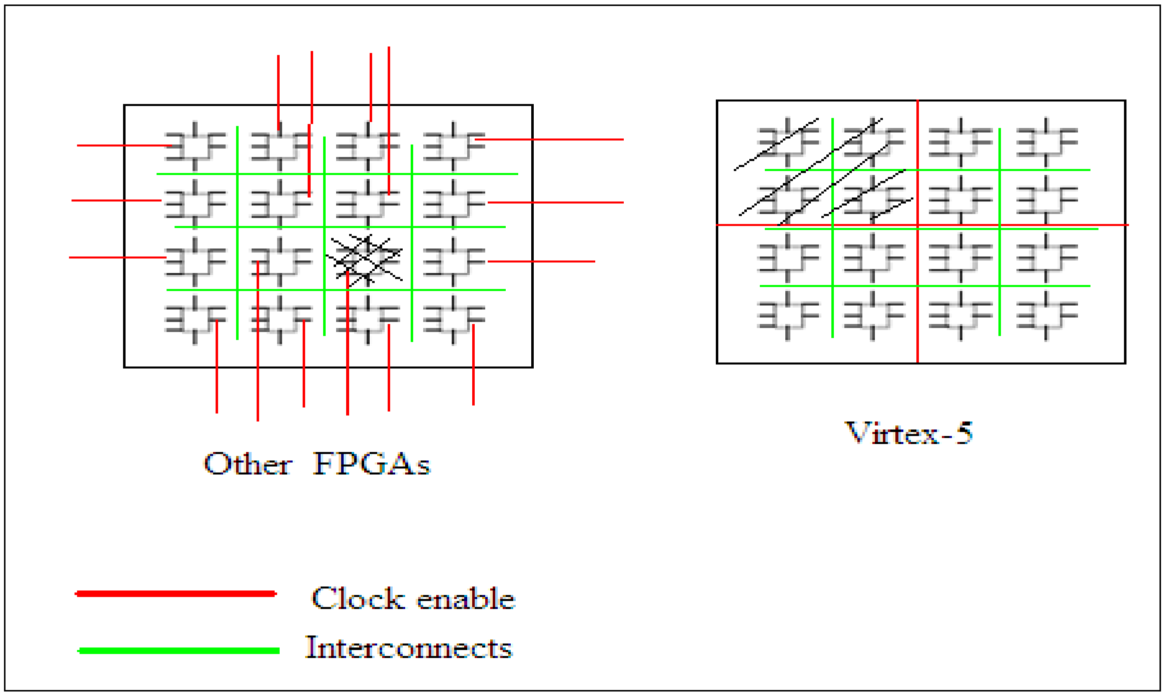


Figure 4.5: Xilinx Virtex -5 Internal Architecture

4.3.1 Tools for FPGA Technology

In FPGA, we have used ModelSim6.5 SE and Xilinx ISE 11 for functional verification of data path of the specific functional block or module. For area estimation, Xilinx design summary report gives information about the no of slices and no of clock buffer used. For dynamic power estimation we have used Xilinx power analyzer.

4.3.2 Methodology

Following are the major steps that need to be performed for area and dynamic power consumption on FPGA [9].

- Using Xilinx ISE software to perform synthesis of the functional module
- Target device selected is Virtex-5
- Simulation of the circuit is performed using ModelSim or Xilinx ISE simulator software
- Simulation step generate VCD file (Value Change Dump), also called simulation file
- Each simulation runs the circuit for a certain no. of iterations

- Implement the synthesize module using Xilinx ISE (Translate, Map, place and Route)
    - Power analysis is done using XPA (XPower Analyzer)
    - XPA takes as input the simulation file generated during simulation and the synthesized version of the circuit
    - Based on the device power characteristics and signal transitions (from simulation file) it calculates avg. power consumed

Note: When we apply clock gating using FPGA virtex 5, then we have to enable clock gating buffers explicitly.

Step 1: Before Start Power Analysis

- First of all, synthesize your ISE project
- Add timing constraints (.UCF file)
- Implement your ISE project. After implementation, ngd output file is built that are used for power analysis.

Step 2: Open Xilinx Power Analyzer

Open XPA in one of these ways:

1) In ISE, in the Processes tab, double-click the **XPower Analyzer** process, located under **Place & Route**

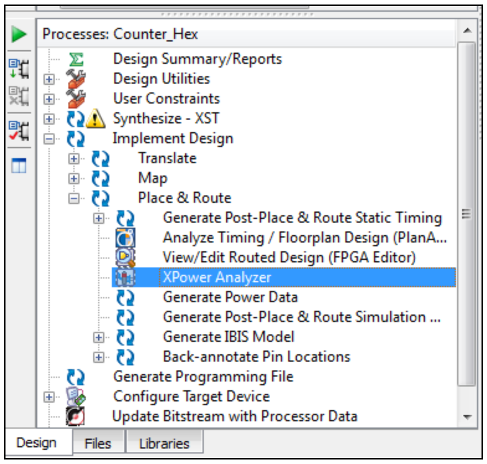

Processes: Counter_Hex
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Translate
Map
Place & Route
Generate Post-Place & Route Static Timing
Analyze Timing / Floorplan Design (PlanA...
View/Edit Routed Design (FPGA Editor)
XPower Analyzer
Generate Power Data
Generate Post-Place & Route Simulation ...
Generate IBIS Model
Back-annotate Pin Locations
Generate Programming File
Configure Target Device
Update Bitstream with Processor Data
Design
Files
Libraries

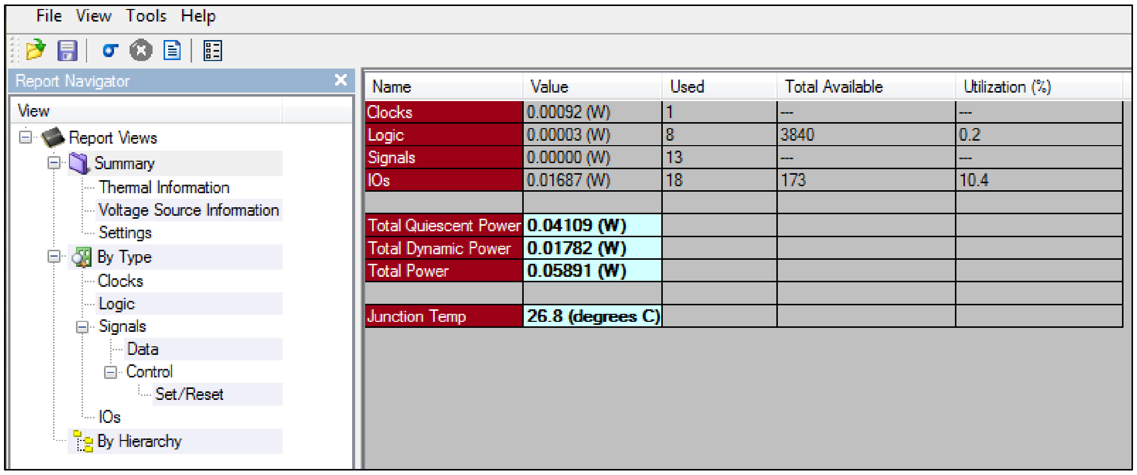
Figure 4.6: Start XPower Analyzer

When you open XPA from ISE, you are analyzing a design that has already been fully placed and routed.

OR

1) Open XPA standalone. To do so:

- Select **Start** > **Run**.
- At the command prompt, type **xpa** and press **Enter**.



Name	Value	Used	Total Available	Utilization (%)
Clocks	0.00092 (W)	1	---	---
Logic	0.00003 (W)	8	3840	0.2
Signals	0.00000 (W)	13	---	---
IOs	0.01687 (W)	18	173	10.4
Total Quiescent Power	0.04109 (W)			
Total Dynamic Power	0.01782 (W)			
Total Power	0.05891 (W)			
Junction Temp	26.8 (degrees C)			

Figure 4.7: XPower Analyzer Window

Step 3: Open Design

In XPA, open the design files in this way:

- Select **File > Open Design**
- In the Open Design dialog box, select the files to use for virtex-5

  usb3_em.ncd (Post-Place and Route NCD file)

  usb3_em.pcf (Physical Constraints File)

  usb3_em.saif (Stimulus file from Simulation)

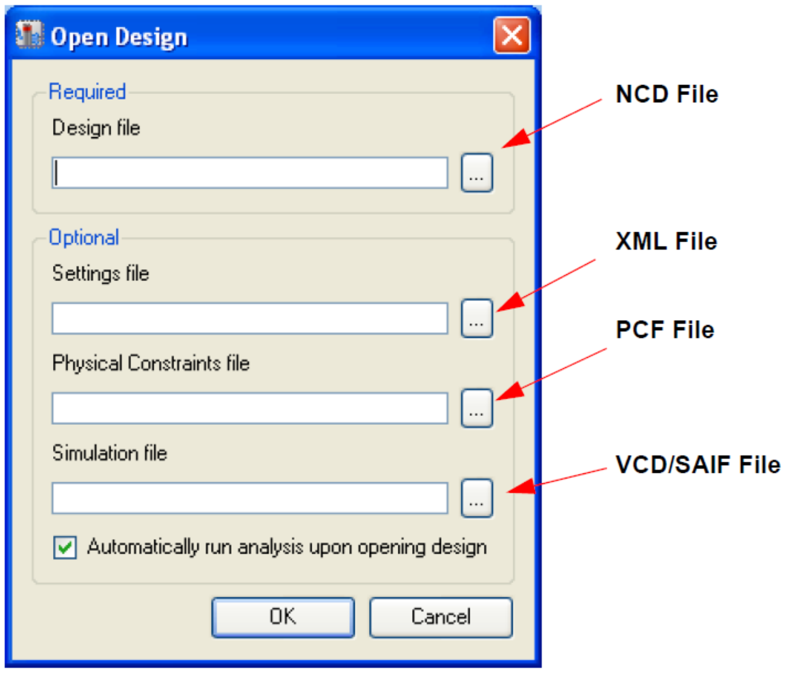


Figure 4.8: Open Design Window in XPower Analyzer

Step 4: View Summary Report

Once XPA is completely opened and has analyzed the design using the specified input files, these report views are displayed:

- The **Summary** view displays all the important voltage rail information.
- The **Thermal Information** view shows thermal information to be used in the device power calculation, and allows you to set the relevant thermal setting to match your hardware and environment

Step 5: Settings

- For a design in which no VCD or SAIF stimulus file is available yet, you can use the **Settings** view to enter estimated activity rates for all different design elements
- Every time a setting or a value is changed or updated in XPA, you must then update the power analysis (**Tools > Update Power Analysis**)
    - Clocks – Shows power for each clock tree in the design.
    - Signals – Shows activity rate and power for the design routing elements.
    - Logic – Shows activity rate and power for the design slice elements (for example, Look-up table (LUT), and flip-flop).
    - IO – Shows activity rate and power for the different I/Os in the design.
    - Dedicated blocks – Shows activity rate and power for the different blocks in the design (for example, block RAM, DSP, Transceiver, and Processor)

Step 6: Power Report by Type

- The By Type views break down the power estimation into specific groups

  In the **clock view**, make sure all clock frequencies have been correctly extracted from the physical constraints file (PCF).

  If the clocks in your design were properly constrained in the UCF (user constraints File) before implementation then frequencies should be correctly extracted from the PCF (physical constraints File) in XPA

Similarly logic, signals and I/Os tabs describe the power consumption of these resources.

Step 7: Add Switching Activity Information or Value Change dump file (VCD)

- Add SAIF information file or VCD file to analyze the actual and more accurate power consumption of your design

I. Add test-bench in your design

II. Run your test-bench

III. Then add SAIF in ISIM command prompt

```
***********************************
 saif open –scope uut –allnets
run 10000ns
saif close
**************************************
```

- Or if you can add VCD in ISIM command prompt

```
************************************
vcd dumpfile vvv.vcd
vcd dumpvars –m /uut
vcd dumpon
run 10000ns
vcd dumpoff
vcd dumpflush
*************************************
```

- Now after adding switching activity information file, you will get an accurate power consumption of your design.

Similarly, similar sequences of steps need to be performed for other modules of USB 3.0 system architecture i-e Data handler switch, packet encoder decoder and so on.

CHAPTER 5

EFFICIENT DESIGN OF NOVEL SYSTEM ARCHITECTURE OF USB 3.0

In this chapter, first we have described our efficient designed novel system architecture, its control and data path flow and how it works. Secondly we have explained the clock gating power optimization technique on our novel system architecture.

5.1 Proposed System Architecture

Microcode is simply a micro-instruction that executes different parts of a more complex instruction. Microcode instructions translate machine instructions into sequences of detailed circuit-level operations. It helps to separate the machine instructions from the underlying electronics so that instructions can be designed and altered more freely. It also makes it feasible to build complex multi-step instructions while still reducing the complexity of the electronic circuitry compared to other methods. For communication standard protocols; we propose to break complex data transactions for example bulk transfer transaction to simpler more generic micro-coded instructions. These smaller instructions reduces the area deployment as there is no need of separate memory or buffer to store memory data transaction parameters instead a centralized data buffer is used. As area requirement has been reduced; so dynamic power dissipation is also decreases. These micro-coded instructions contain the information related to transfer type, transfer direction, endpoint number, size of payload and addresses (source and destination). In our USB 3.0 micro-coded system architecture, there is an additional power manager functional block that controls the root clock of all others functional blocks. Figure-3 shows high level system architecture of USB 3.0. The system architecture discussed in this paper is comprised of the following components: Event Manager, Data Handler (DH), Packet Encoder Decoder (PED), Data Buffer, Advance High Performance System Bus (AHB) and Link Layer

FIFO.

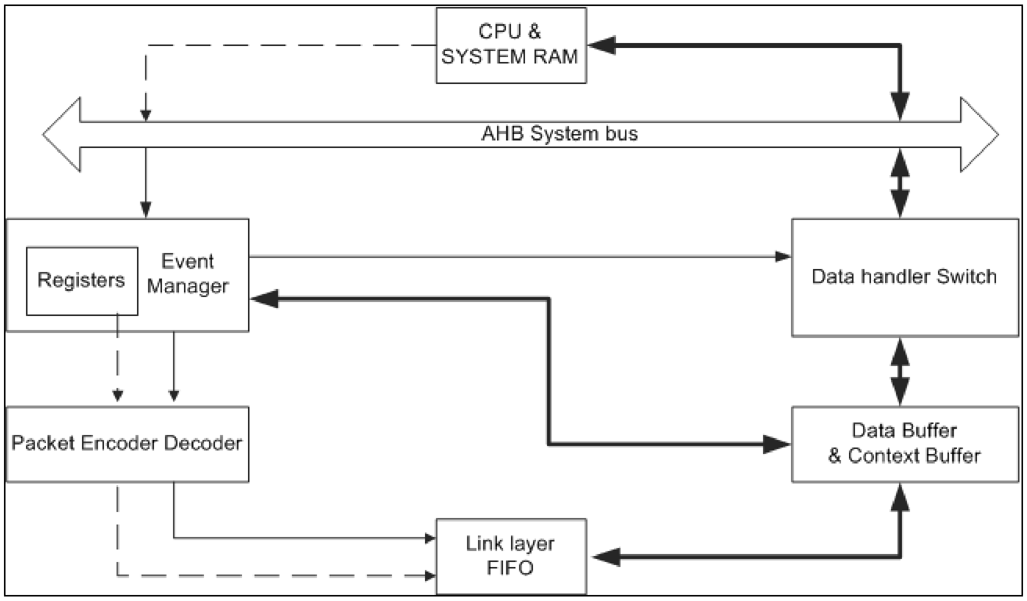


Figure 5.1: High level system architecture of USB 3.0

5.2 Event Manager

In our proposed architecture, Event manager functional block acts as a main controller. It is responsible for generating instructions for other functional block of the architecture depending upon the register configurations. Advanced high performance Bus (AMBA-AHB) is selected as the inter-connect bus. AHB offers the bandwidth that is necessary for transferring data to or from system memory of large capacity. In order to achieve high performance, AHB bus supports burst transfers, split transactions, pipelined operations, and data widths of 64 or 128 bits. We are not supporting the split transactions and other features. Instead we are supporting the customize version of the AHB bus.CPU programs the registers of event manager through system bus using slave interface. Event manager also informs the CPU; whenever the whole transaction is executed successfully or if any error condition occurs in running transaction. Event manager comprised of the following major sub-blocks as shown in figure 4. These are:

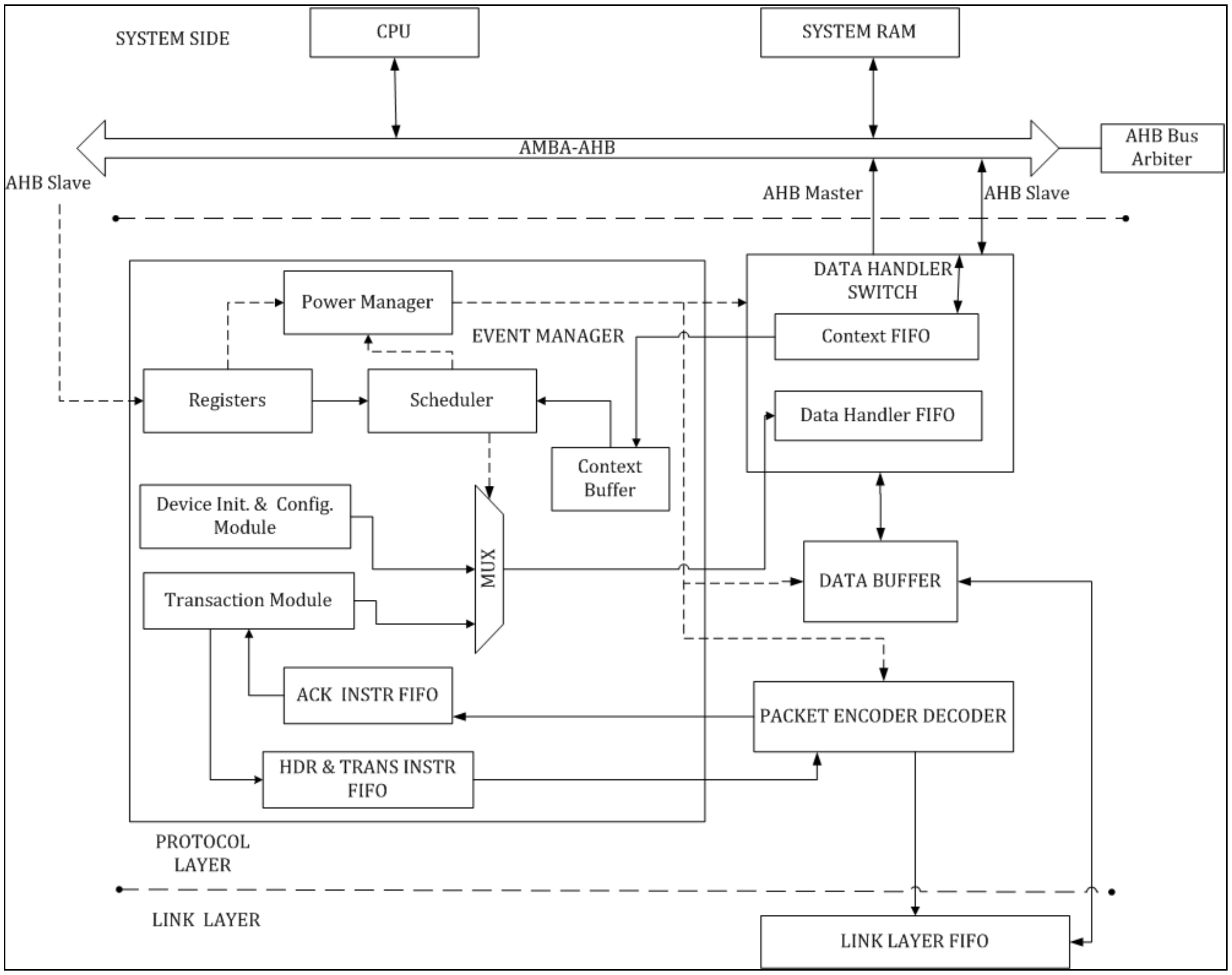


Figure 5.2: Micro Coded Instruction based USB 3.0 System Architecture

- Instruction FIFO's, (HDR & TRANS INSTR FIFO, ACK INSTR FIFO and DH INSTR FIFO),
- Event Manager Registers,
- Scheduler (Device initialization or configuration FSM and Transaction FSM)
- Power Manager

5.2.1 Instruction FIFO's

Three different types of micro coded instructions are designed based on their functionality.

Data Handler Instructions (DH): DH instructions are responsible to read or write context data from system RAM. These instructions consist of information required for describing the requested memory transactions.

Header Packet & Transaction Instruction (HDR & TRANS): HDR & TRANS instructions contain header packet & transaction related information i-e device address, endpoint number, payload size, transfer type, direction of transfer(host to device or device to host) and so on. These instructions are used by packet encoder decoder functional block in USB 3.0 micro architecture.

Acknowledgment Instructions (ACK): ACK instructions contain the status of the particular transaction to handle acknowledgement and error cases.

Separate instruction FIFOs (DH instruction FIFO, ACK instruction FIFO and HDR & TRANS instruction FIFO) are provided in the architecture to allow pipelining.

5.2.2 Event Manager Registers

In control & event registers of event manager, there are n+1 door bell registers. Where n is the number of devices that micro-coded instruction based architecture supports. There are two different types of doorbell registers; host controller doorbell registers and device context doorbell registers. Host controller doorbell registers contains the information regarding the device initialization & configuration; whereas device context doorbell registers contains the information related to particular endpoint of the device. System software rings the particular doorbell register by writing specific register values. Based upon these register values, event manager initiates particular transaction. Also there are some status registers of event manager that informs the system CPU about the status of device initialization & configurations. Each bit in this register corresponds to an individual status of the device i-e device attach detach, device reset, device enabled and device power management states (idle or suspended states).

5.2.3 Scheduler

Control & event registers of the Event Manager are configured by CPU via AHB bus through AHB slave interface. A scheduler, based upon the contents of the door bell registers (host controller or device context) and priority of the individual endpoints of the device, enables device initialization and configuration FSM or transaction FSM.

Device initialization and configuration FSM has highest priority. This module is responsible for initialization and configuration of the device. In USB 3.0 communication protocol; event manager waits the device attach signal of any particular device from the link layer FIFO. This signal is set by link layer upon detection of device to the USB port. System software rings host controller doorbell register. Scheduler gives control to device initialization or configuration FSM. First of all, this FSM reset & enabled the device by configuring status registers. This FSM writes event transaction in system RAM that informs the system software that device has been reset and also enabled. Event transaction informs system CPU about the status of the running transaction. After that this module configures the device address through control transaction. This module after complete initialization & configuration makes device in running state. Now device is ready to perform any particular transaction. In case of any error, device state has been changed from running to disable state and event manager removes its priority from the scheduling list.

Whenever device context doorbell register rings, scheduler assigns priority to the individual endpoint of device based upon periodic or aperiodic. Periodic endpoints have highest priority and have been allocated 80 percent of the total available bandwidth and the remaining 20 percent bandwidth is allocated to the aperiodic endpoints. Aperidic endpoints perform bulk or control transaction; whereas periodic endpoints perform interrupt and isochronous transactions of USB

3.0 protocol.

When system software rings the device context doorbell registers variably; scheduler decides which endpoint has highest priority and gives control to transaction module to serve the particular endpoint of the device. At any time, there are multiple doorbell registers for multiple devices or transfer rings that cannot be supported through the set of predefine registers. Our micro coded architecture are automatically store the un-serviced doorbell registers into the context buffer and services (load) them accordingly based on scheduler as required. A Context Buffer in the event manager keeps updated configurations of the entire endpoints of the attached USB 3.0 devices. Transaction module writes the micro-coded instructions in DH INSTR FIFO to read context data from system RAM. These parameters contain the information regarding any particular transaction (Bulk, Interrupt, isochronous or control) data length transferred to the device, address of the data payload to be read from the system RAM and also direction of the particular transaction (host to device or device to host). After extracting context parameters; event manager will start particular transaction indicated by context parameters. If the direction of transaction is OUT (host to device);then event manager writes data handler instructions in DH INSTR FIFO to get data packet payload from system RAM to data buffer. Concurrently, it writes HDR & TRANS instructions in HDR \& TRANS INSTR FIFO for PED. After receiving acknowledgement in ACK INSTR FIFO; it writes DH instruction in DH INSTR FIFO for event transaction. Event transaction informs system CPU about the status of the current transaction. After that scheduler decides which transaction has highest priority and serve it accordingly.

5.3 Data Handler

Data handler acts like a switch that enables when event manager write DH instructions in DH INSTR FIFO. Data handler is responsible to load or store data payload and transaction descriptors from system RAM to the Data Buffer or vice versa. In our proposed architecture, data structure is defined in memory that contains the context parameters of all the attached USB devices. There are some generalized instructions (DH instructions) that perform data transactions. Data handler have AHB master and slave interface that communicates with the system CPU and is responsible to load or store data payload or transaction descriptors from system RAM. Data Handler has a context FIFO. Context FIFO contains the configuration information of the specific endpoint related with the current transaction. It also specifies the memory location of the data payload in the system RAM that need to be transmitted or received.

Data handler has four different interfaces. These are:

Data Buffer to System RAM (DBUFF-RAM) Interface

System RAM to Data Buffer (RAM-DBUFF) Interface

Context Buffer to System RAM (CFIFO-RAM) Interface

System RAM to Context Buffer (RAM-CFIFO) Interface

There are some specific bits in DH instruction that informs the data handler to enable any particular interface.

5.3.1 RAM-CFIFO Interface

This interface is used to move context parameters from system RAM to the context FIFO. Event manager informs data handler through DH instruction to enable this particular interface. These parameters contain the information regarding particular transaction i-e transfer type (bulk, interrupt, isochronous or control), transfer direction (read or write), data payload size, device address, specific endpoint of the device and so on.

5.3.2 DBUFF-RAM Interface

DBUFF-RAM Interface moves data payload from data buffer to system RAM. Event manager performs read transaction after decoding context parameters coming from system RAM. Event manager writes DH instruction in DH INSTR FIFO to move data payload from data buffer to system RAM through data handler.

5.3.3 RAM-DBUFF Interface

During write transaction initiated by event manager, data handler load data payload from system RAM to data buffer through this interface. The size of data payload and transfer direction is added in DH instruction. Data handler decodes DH instruction and performs data transaction.

5.3.4 CFIFO-RAM Interface

This interface informs the CPU about the status of the particular transaction (write, read or device configuration & initialization) through event transaction. Event manager writes event data in context FIFO of data handler and also DH instruction in DH INSTR FIFO to execute event transaction.

5.4 Packet Encoder Decoder Block (PED)

Packet encoder decoder functional unit executing packet by packet transaction to or from the link layer FIFO using some specialize designed instructions driven through event manager. Packet encoder decoder reads HDR & TRANS instructions and decodes them to execute particular transaction. Packet encoder decoder makes header packet and data packet. Data packet contains both data packet header and data packet payload. Data packet payload is appended with the data packet header that comes from the data buffer. Packet encoder decoder also adds sequence number to each data packet. The header packet and data packet is send to the link layer FIFO. On receiving the acknowledgment packets from link layer FIFO, Packet Encoding/Decoding Module generates the ACK instruction. This instruction informs event manager the status of the current transaction.

5.5 Data Buffer

Data buffer contains data packet payload of data packet header. During write transaction, data payload comes in data buffer through data handler RAM-DBUFF Interface. Similarly, for read transaction data payload from device comes through PED AHB slave interface. There are separate memory locations in data buffer for write transactions data payload and read transactions data payload.

5.6 Link Layer FIFO

Link layer FIFO provides an interface between link layer and protocol layer of USB 3.0 system architecture. For write transactions, packet encoder decoder functional block writes header packet and data packet in link layer FIFO. After that Link layer FIFO sends these packets to the link layer. When link layer sends packets to the device; it responds with acknowledgements in form of packets. Link layer writes these ACK packets in Link layer FIFO that is read by protocol layer. Packet encoder decoder read these packets and generates ACK instructions for event manager. ACK packets contain status of the transmitted packet. Similarly, for read transaction, Link layer writes header packet and data packet in link layer FIFO and protocol layer acknowledges the transmitted packets through link layer FIFO.

5.7 Power Management in System Architecture

Clock gating is a technique in which we can cut off the clock to a sequential circuit or a part of it, while its functionality is not required. Consequently, there is lesser clock and other signals switching activity in the system architecture and lesser power is consumed. Power manager functional block performs root clock gating to each of functional block of the whole system architecture. Before applying clock gating, there is need to evaluate efficient enabling logic with slight area overhead and complexity. Based upon the registers configuration of the event manager and data path logic of the whole system architecture; Power manager enables certain functional blocks while shut-off other functional blocks because their processing are not required. Power manager contains an FSM based design that can control all data path logic of other functional blocks of the whole system architecture as shown in Figure-5 and Figure-6.

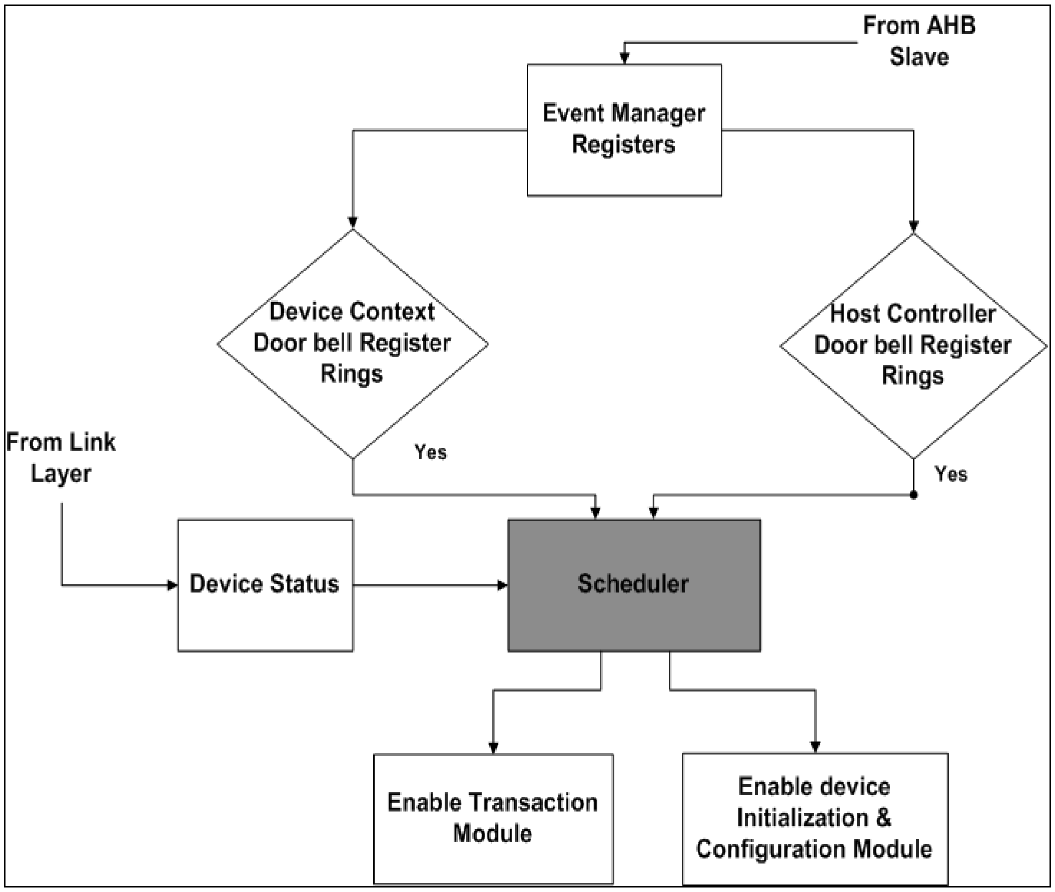
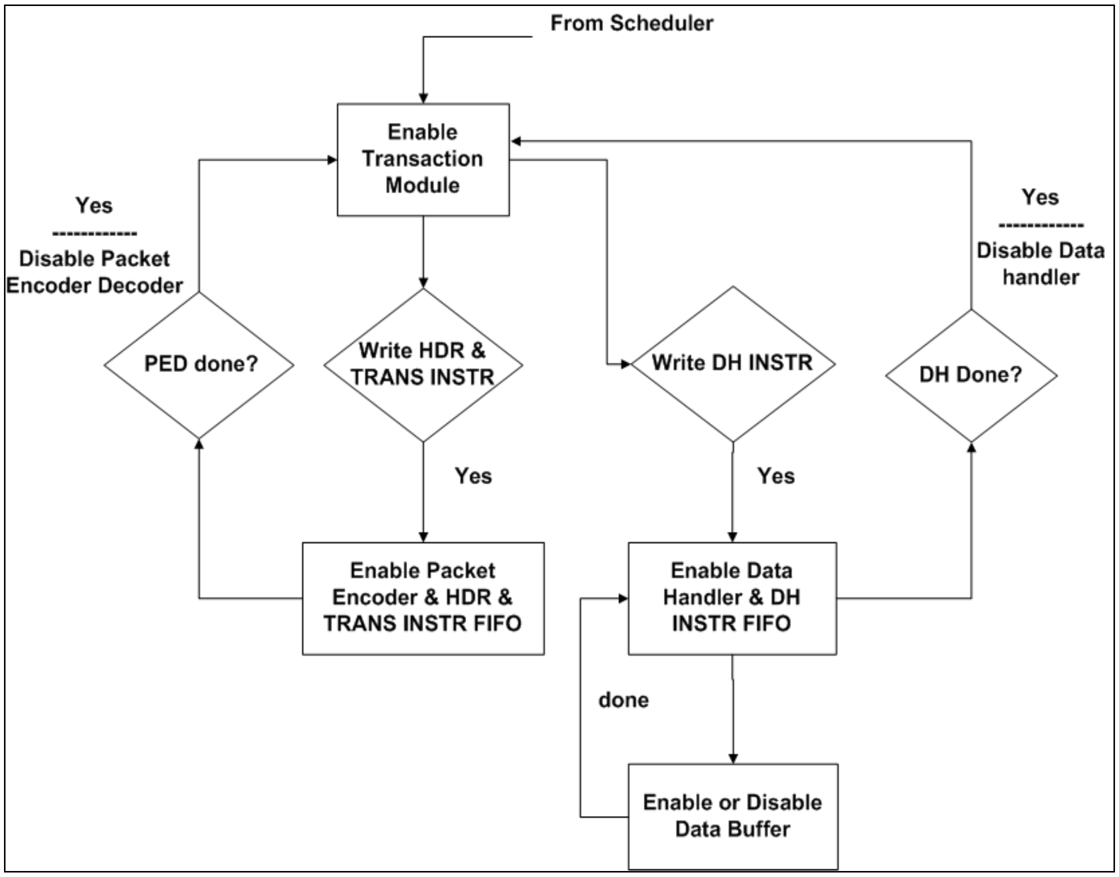

From AHB Slave
Event Manager Registers
Device Context Door bell Register Rings
Host Controller Door bell Register Rings
Yes
Yes
From Link Layer
Device Status
Scheduler
Enable Transaction Module
Enable device Initialization & Configuration Module

Figure 5.3: Clock Gating Pseudo-code I


From Scheduler
Enable Transaction Module
Yes

Disable Packet Encoder Decoder
Yes

Disable Data handler
PED done?
Write HDR & TRANS INSTR
Write DH INSTR
DH Done?
Yes
Yes
Enable Packet Encoder & HDR & TRANS INSTR FIFO
Enable Data Handler & DH INSTR FIFO
done
Enable or Disable Data Buffer

Figure 5.4: Clock Gating Pseudo-code II

Based on the event manager register configurations and scheduling; event manager enables either device initialization and transaction module. If the host controller doorbell register rings event manager enables device initialization and configuration module. When device context doorbell rings; transaction module is enabled. Now transaction module enables packet encoder decoder (PED), HDR & TRANS INSTR FIFO, DH INSTR FIFO and data handler functional block. When HDR & TRANS INSTR is needed to be written in HDR & TRANS INSTR FIFO; transaction module enables clock of packet encoder decoder functional block. When PED done signal is received from PED function module; transaction module disable the clock of packet encoder decoder functional block. Similarly, when DH INSTR is required in DH INSTR FIFO for memory data transaction; transaction module enables clock of the data handler module. When

DH done is received from data handler; transaction module disables the clock signal of data handler functional module. Also, data handler enables the data buffer clock signal; when there is a need to store or load data in data buffer. When the whole data transaction is executed successfully, data handler disables the data buffer. Device initialization and configuration module employ the similar logic of as utilize by transaction module to enable or disable all the functional blocks of the system architecture.

5.8 Micro-Coded Instructions

5.8.1 Header Packet & Transaction Instruction (HDR TRANS INSTR)

HDR & TRANS instructions contain header packet & transaction related information i-e device address, endpoint number, payload size, transfer type, direction of transfer (host to device or device to host) and so on. These instructions are used by packet encoder decoder functional block in USB 3.0 micro architecture.HDR TRANS INSTR format has been shown in table 1.

device_address	routing_string		opcode_tx
sequence_number	direction	transfer_type	ep_num
data_payload_address	data_payload_size		
Reserved	max_packet_size		

Table 1: HDR TRANS INSTR FORMAT

opcode_tx specifies the particular instruction associated with transfer type. **routing_string** specifies the exact port where device has been attached. **device_address** specifies the actual address of the particular device. **ep_num** indicates the particular interface on a device that is either a source or sink of information**.** **transfer_type** defines the meticulous transaction type i-e bulk, interrupt , isochronous or control. **direction** bits show that either to move data from host to device or from device to host. **Sequence number** shows particular sequence number of the header packet. **Data_payload_size** shows the size of data payload that will be appended at the end of the data packet header. **Data_payload_address** contains the address of the data buffer from where we move data to the link layer FIFO. **max_packet_size** defines the max packet size of the particular header packet.

5.8.2 Acknowledgment Instructions (ACK INSTR)

ACK instructions contain the status of the particular transaction to handle acknowledgement and error cases. ACK INSTR format has the following fields shown in table 2;

device_address	routing_string		opcode_rx
status	direction	transfer_type	ep_num
reserved	rx_data_payload_size		
reserved			

Table 2: ACK INSTR FORMAT

opcode_rx specifies the particular instruction associated with transfer type. routing_string specifies the exact port where device has been attached. device_address specifies the actual address of the particular device. ep_num indicates the particular interface on a device that is either a source or sink of information. transfer_type defines the meticulous transaction type i-e bulk, interrupt , isochronous or control. direction bits show that either to move data from host to device or from device to host. Status bits either the running transaction is completed successfully or not. rx_data_payload_size shows the data payload size that has been acknowledge successfully.

5.8.3 Data Handler Instructions (DH INSTR)

DH instructions are responsible to read or write context data from system RAM. These instructions consist of information required for describing the requested memory transactions. DH-INSTR format has the following fields shown in table-3;

Burst Size indicates the number of data words (32 bit in width) that is read from system RAM. Transfer type indicates the particular transaction type. Direction bits indicate the particular direction of data transfer. Destination address is the address where we place the data packet payload or context parameters coming from system RAM. Similarly, source address contains the address of system RAM from where we get data packet payload or context parameters.

burst_size	transfer_type	direction
Destination Address		
Source Address		
Reserved		

Table 3: DH INSTR FORMAT

5.8.4 How the Micro-coded Instruction Works

In our proposed system architecture, micro-coded instructions move data payload between system RAM and link layer FIFO or vice versa.

5.8.4.1 Bulk-OUT-1K Transaction

Following sequence of steps has to be performed for bulk OUT transaction to move 1K data payload between system RAM and link layer FIFO.

Step1: Reading Context Parameters from System RAM

When the doorbell rings from system CPU, event manager writes DH instruction in DH-INSTR-FIFO to read context parameters from system RAM. In this step, DH instruction parameters contain burst size “00 or 01” depending upon the corresponding no of Dwords in context parameters. In this step, transfer type field is don’t care because transfer type will be determine after decoding these context parameters. direction field in this step will be “10” indicating context parameters will be move from system RAM to the context FIFO as shown in table 4. This instruction read context parameters from system RAM as indicated by source address

(32’h100) and place them context FIFO so the destination address in this instruction will be don’t care. The detail sequence of this instruction is shown in table-5

Burst Size	**Transfer type**	**Direction**	**Destination Address**	**Source Address**
00: 4 Dwords 01: 8 Dwords 10: 16 Dwords 11: Unspecified	00: Control 01: Isochronous 10: Interrupt 11: Bulk	00: RAM- Data Buffer 01: Data Buffer- RAM 10: RAM-Context FIFO 11: Context FIFO- RAM	-	-

Table 4: DH INSTR PARAMETERS

2’b00	Don’t care(00)	2’b10
Don’t Care		
32’h0100		
Reserved		

Table 5: DH instruction for reading Context parameters

Step 2: Decode Context Parameters & Write DH-INSTR for data payload

When the context parameters come from system RAM, event manager will decode these context parameters. These context parameters contain data payload size, particular transaction type and direction of that particular transaction. If it is OUT transaction (i-e Bulk OUT), event manager writes DH instructions in DH-INSTR-FIFO to move data packet payload from system RAM to the data buffer. In these DH instructions, burst size is “10” i-e 16 Dwords and direction is “00” indicating data payload will be move from system RAM to the data buffer as specified by the source and destination addresses. Data handler places the data payload as specified by the destination address.

2’b10	2’b11	2’b00
32’h1		
32’h0200		
Reserved		

Table 6: DH instruction for writing data payload in Data buffer

Step 3: Write HDR TRANS INSTR

Concurrently, event manager writes BULK-OUT transaction in HDR TRANS INSTR FIFO for header packet. This instruction contains four Dwords (32 bits each). These Dwords contains the information regarding particular transaction type, endpoint number, device address and size of data payload that will be appended after the header packet as shown in table 1. The detail bit sequence of HDR TRANS INSTR for Bulk-OUT-1K has shown in table-7

7’h1	20’h0		5’h5
5’h0	1’h0	2’h3	5’h2
16’h1	16’h400		
Reserved	16’h400		

Table 7: HDR TRANS INSTR for BULK-OUT-1K

Step 4: Decode HDR TRANS INSTR and Transmit Data packet

PED decodes HDR TRANS instruction and produce data packet header of the data packet. PED sends data packet header to the header part of the link layer FIFO. Similarly, PED will also move data payload of data packet from data buffer to the payload part of the link layer FIFO through AHB slave interface.

Step 5: PED waits ACK packets and generate ACK Instruction

Now PED waits an ACK packet that comes from the link layer. After receiving an ACK packet, PED generates an ACK instruction that will be written in ACK instruction FIFO. ACK instruction format is shown in table 2. ACK instruction contains the status of the particular transaction i-e success or error etc. The detail bit sequence of ACK INSTR for BULK-OUT-1K is shown in table-8

7’h1	20’h0		5’h1
3’h0	1’h0	2’h3	5’h2
reserved	16’h400		
reserved			

Table 8: ACK INSTR for BULK-OUT-1K

Step 6: Event Manager writes event transaction

Now Event manager initiates an event transaction that informs the host (CPU) the status of the current transaction. Event manager writes event transaction parameters in context FIFO. After that event manager writes DH-INSTR in DH-INSTR-FIFO. In this instruction, burst size is “00” and direction is “11”that informs the data handler that event transaction parameters contains four Dwords (32 bit each) and data payload is to be written from context FIFO to the system RAM. The table-9 shows detail bit sequence of this DH Instruction.

2’b00	Don’t care(00)	2’b11
Context FIFO(32’h0)		
32’h1		
Reserved		

Table 9: DH INSTR for Event transaction

5.8.4.2 Bulk-IN-1K Transaction

Following sequence of steps has to be performed for bulk IN transaction to move 1K data payload between system RAM and link layer FIFO.

Step1: Reading Context Parameters from System RAM

When the doorbell rings from system CPU, event manager writes DH instruction in DH-INSTR-FIFO to read context parameters from system RAM. In this step, DH instruction parameters contain burst size “00 or 01” depending upon the corresponding no of Dwords in context parameters. In this step, transfer type field is don’t care because transfer type will be determine after decoding these context parameters. direction field in this step will be “10” indicating context parameters will be move from system RAM to the context FIFO as shown in table 4. This instruction read context parameters from system RAM as indicated by source address (32’h100) and place them context FIFO so the destination address in this instruction will be don’t care. The detail sequence of this instruction is shown in table-5

Step 2: Decode Context Parameters & Write HDR TRANS INSTR for header Packet

When the context parameters come from system RAM, event manager will decode these context parameters. These context parameters contain data payload size, particular transaction type and direction of that particular transaction. If it is IN transaction (i-e Bulk IN), event manager writes HDR TRANS INSTR for PED to make header packet and send to the link layer FIFO. The detail bit sequence is shown in table -10

7’h1	20’h0		5’h8
5’h0	1’h1	2’h3	5’h4
16’h1	16’h400		
Reserved	16’h400		

Table 10: HDR TRANS INSTR for BULK-IN-1K

Step 3: Decode HDR TRANS INSTR and make Header Packet

PED decodes HDR TRANS Instruction and make header packet and send it to the link layer FIFO.

Step 4: PED waits ACK packets and generate ACK Instruction

Now PED waits an ACK packet that comes from the link layer. After receiving an ACK packet, PED generates an ACK instruction that will be written in ACK instruction FIFO. ACK instruction format is shown in table-2. ACK instruction contains the status of the particular transaction i-e success or error etc. the detail bit sequence of this specific instruction is shown in table-11

7’h1	20’h0		5’h2
3’h0	1’h0	2’h3	5’h4
reserved	16’h400		
reserved			

Table 11: ACK INSTR for BULK-IN-1K

Step 5: Event manager writes DH-INSTR for data payload:

When the ACK instruction comes from system RAM, event manager will decode instruction parameters. Event manager checks the status of the transaction and payload size that has been written in data buffer. Event manager writes DH-INSTR in DH instruction FIFO. In these DH instructions, burst size is “10” i-e 16 Dwords and direction is “01” indicating data payload will be move from data buffer to the system RAM as specified by the source and destination addresses. The detail bit sequence of this particular instruction is shown in table-12

2’b10	2’b11	2’b01
32’h10		
32’h0220		
Reserved		

Table 12: DH-INSTR for BULK-IN-1K

Step 6: Event Manager writes event transaction

Now Event manager initiates an event transaction that informs the host (CPU) the status of the current transaction. Event manager writes event transaction parameters in context FIFO. After that event manager writes DH-INSTR in DH-INSTR-FIFO. In this instruction, burst size is “00” and direction is “11”that informs the data handler that event transaction parameters contains four Dwords (32 bit each) and data payload is to be written from context FIFO to the system RAM as shown in table-4. The detail bit sequence of event transaction DH instruction is shown in table-9.

CHAPTER 6

DISCUSSION & RESULTS

This chapter has described the performance parameters of the designed system architecture and has explained how it performed better in terms of area deployment, power consumption, interrupt handling, complexity, flexibility and throughput. Secondly, we have estimated some performance parameters of the designed system architecture like area deployment, toggle factor (switching activity), and dynamic power consumption with and without root clock gating on ASIC technology (UMC 90nm) as well as on FPGA Xilinx Virtex-5.

6.1 Discussion on Performance Parameters

In our novel system architecture, all the configuration and data transactions has been executed by using multiple small micro-coded instructions that removes the requirement of extra buffer or RAM to save the transactions parameters. These small micro-coded instructions include pipelining in the whole architecture that improves the overall efficiency of system architecture. These micro-coded instructions require multiple clock cycles to perform particular transaction but to limit the clock speed we increase the data bus width. Also in this whole architecture, DMA implicit functionality such as scatter and gather, descriptor fetches are implicitly done using the memory data structure and through the micro-coded instructions where the instructions are executed automatically based on fetches from memory and their decoding. These tasks have been accomplished by using event manager and data handler. Also there is no need of separate DMA registers for memory data transactions parameters. Therefore the area has been deployed efficiently in the whole architecture. Since area deployment has been reduced so the overall power consumption of the whole architecture has been reduced.

Interrupt based approach has been mitigated that reduces the clock speed requirement in the whole architecture. Generally interrupts halt the execution of the running transaction and perform particular task indicated by the specific interrupt. Also to store the contents and state of the running transaction there is a need of some local memory. To handle interrupts, there is a particular event transaction that has been introduced. This transaction writes the status of the current transaction in memory. System CPU reads the status of all transaction from system RAM through polling.

The system architecture that has been designed is fully pipelined. There are different FIFO's that are initiated in the whole system architecture that is working alone. Also the transmission and reception data path will perform transactions concurrently and independently. If there is some congestion or overflow occurs in any side of the data path either transmission or reception; the other side executes transactions separately. Since the operating frequency of USB 3.0 protocol designed architecture is 250 MHz. The data bus of whole system architecture is 32 bit wide. To move data from system RAM to the device, AHB system bus has been used. If CPU grants AHB system bus 60% of the time; then theoretical data rate or expected throughput of one transmission path is 4800 Mbits/sec (250 * 0.6*32); and for two way traffic it will be much greater than required data rate 4000Mbits/sec (4Gbps). But due to some other overheads of erroneous transaction and non-availability of AHB system bus expected data rate will decrease but it will still meet the data rate requirement of USB 3.0 protocol. We are working on the assessment of actual throughput or data rate that our designed novel system architecture will meet.

This architecture provides flexibility, robustness and centrality to the architecture because it has been realize that usage of these micro coded instructions will be a step towards universal serial bus controller (USC). We believe that we can come with one universal serial controller for all buses such as PCI express and high performance buses due to their similarity with USB; and also through approach of defining instructions in the memory using the data structures that will be utilize the micro coded instructions

6.2 Performance Parameters Evaluation on ASIC Technology (UMC 90nm)

Performance of designed USB 3.0 system architecture depends upon the number of gate counts, toggle factor and the power consumption of the whole architecture. To compute power consumption, we use UMC 90nm ASIC technology library (FSDOA_A Core Cell).. An implementation of the described architecture was included on Modelsim SE 6.5 of Mentor Graphics Tools and on Cadence Encounter RTL Compiler.

We have evaluated number of gate counts of different modules of the proposed architecture and its estimated power consumption. The toggle factor calculated from code coverage details of Modelsim SE 6.5. The table-1 shows that on average toggle factor of the whole novel architecture is 52.7 percent. When we apply root clock gating on our novel architecture, toggle factor minimizes from 52.7% to 44.6%. Similarly, no of gate counts of different modules of USB 3.0 have evaluated. In table-1, results show that by applying clock gating through power manager 3,548 gates are added. Also the dynamic power consumption by introducing root clock gating decreases from 102.78 to 90.25. Our results in table-1 and figure -1 shows that by applying root clock gating, power consumption reduces up to 12% with approximately 2% increase in number of gate counts.

USB 3.0 Modules	No of Gate Count (gates)		Toggle factor (%)		Power Consumption (mW)	
	Without Root Clock Gating	With Root Clock Gating	Without Root Clock Gating	With Root Clock Gating	Without Root Clock Gating	With Root Clock Gating
Event Manager	98,403	98,403	0.538	0.437	57.35	48.37
Data Handler	26,324	26,324	0.574	0.461	17.01	13.65
Packet Encoder Decoder	9,536	9,536	0.483	0.452	5.18	4.85
Data Buffer	31,337	31,337	0.543	0.424	19.14	14.95
Link Layer FIFO	9,756	9,756	0.554	0.468	6.08	5.14
Power Manager	-	3,548	-	0.572	-	2.28
Overall System Architecture	1,75356	1,78904	0.521	0.4484	102.78	90.25

Table 13: Performance Parameters of USB 3.0 Modules

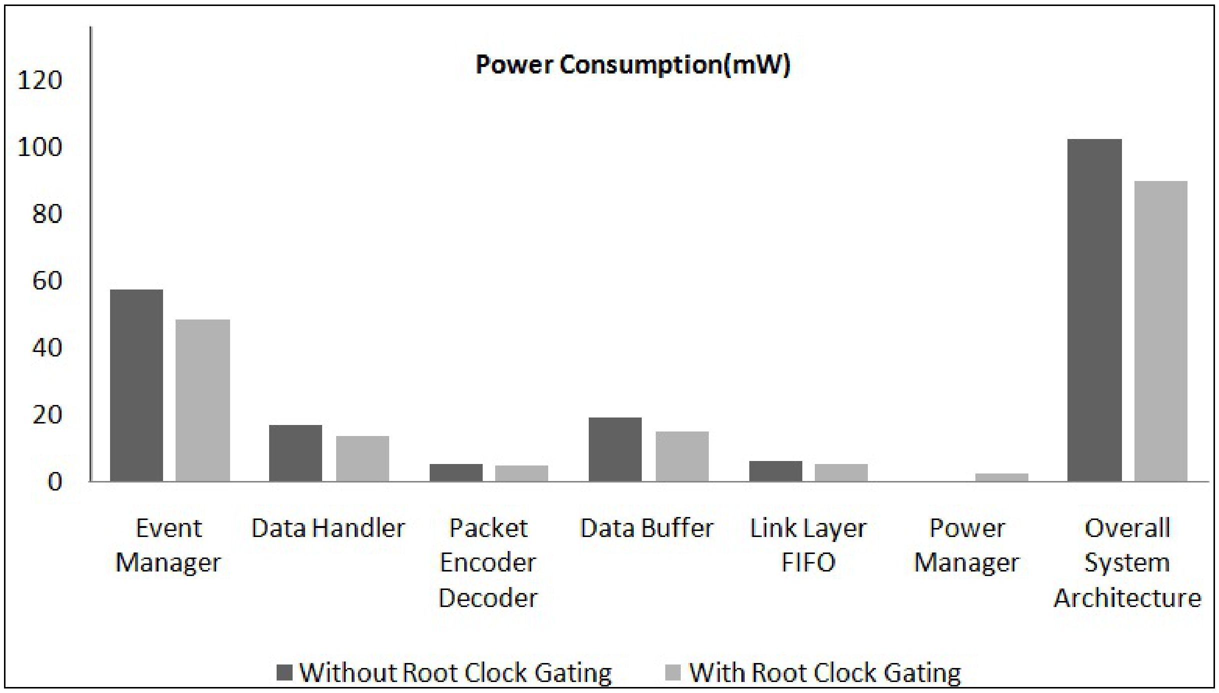


Figure 6.1: Power Dissipation in USB 3.0 System Architecture on ASIC

6.3 Performance Parameters Evaluation on FPGA (Xilinx Virtex-5)

To evaluate the performance parameters (dynamic power and area utilization) of USB 3.0 modules; Xilinx Virtex-5 FPGA has been used. It has divided the FPGA chip internally into clock zones for clock distribution. Clock is controlled through Global Clock Buffers (there are 32 global clock buffers in virtex-5 FPGA).

We have tested three main modules of event manager on FPGA namely; event manager (USB3_EM), data handler (USB3_DH) and control unit of packet encoder decoder (USB3_PED_CU). We have run different test cases of USB 3.0 transactions i-e BULK_1K, INTERRUPT_1K, BULK_4K and INTERRUPT_4K with and without clock gating. BULK_1K and INTERRUPT_1K test case indicate that we have sent 1024 bytes data payload to the device and device sends an acknowledgement of the transaction after receiving the whole data payload. Similarly, BULK_4K test case indicates that we have sent 4096 bytes data payload to the device without waiting an acknowledgement from the device. At the other end device has sent an acknowledgement after receiving 1024 bytes data payload successfully. For 4096 bytes payload it has send four acknowledgments. In INTERRUPT_4K, there is slight change of data flow. In this test case, 1024 bytes data payload has sent from host (system side) to the device, and after receiving an acknowledgment from the device will sent next 1024 bytes data payload and so on.

We have run these test cases for USB 3.0 modules and have estimated results in terms dynamic power dissipation and area deployment.

USB 3.0 Modules Without Clock Gating				
Modules		**Dynamic Power(mW)**	**Number of Occupied Slices**	**No of Clock Buffer**
USB3_EM	Bulk_1K	41.04	866 Slices	1 BUFG
	Bulk_4K	43.57	866 Slices	1 BUFG
	Intrp_1K	41.04	866 Slices	1 BUFG
	Intrp_4K	44.72	866 Slices	1 BUFG
USB3_DH	Bulk_1K	82.25	542 Slices	1 BUFG
	Bulk_4K	86.81	542 Slices	1 BUFG
	Intrp_1K	82.25	542 Slices	1 BUFG
	Intrp_4K	85.78	542 Slices	1 BUFG
USB3_PED_CU	Bulk_1K	18.56	141 Slices	3 BUFG
	Bulk_4K	20.28	141 Slices	3 BUFG
	Intrp_1K	18.56	141 Slices	3 BUFG
	Intrp_4K	21.84	141 Slices	3 BUFG

Table 14: Performance Parameters without Clock Gating on FPGA

USB 3.0 Modules With Clock Gating				
Modules		**Dynamic Power(mW)**	**Number of Occupied Slices**	**No of Clock Buffer**
USB3_EM	Bulk_1K	34.68	894 Slices	1 BUFG 5 BUFGCTRLs
	Bulk_4K	36.91	894 Slices	1 BUFG 5 BUFGCTRLs
	Intrp_1K	34.68	894 Slices	1 BUFG 5 BUFGCTRLs
	Intrp_4K	37.21	894 Slices	1 BUFG 5 BUFGCTRLs
USB3_DH	Bulk_1K	68.88	543 Slices	1 BUFG 3 BUFGCTRLs

	Bulk_4K	72.12	543 Slices	1 BUFG 3 BUFGCTRLs
	Intrp_1K	68.88	543 Slices	1 BUFG 3 BUFGCTRLs
	Intrp_4K	71.45	543 Slices	1 BUFG 3 BUFGCTRLs
USB3_PED_CU	Bulk_1K	16.31	2.62485	3 BUFG 8 BUFGCTRLs
	Bulk_4K	17.36	2.6259	3 BUFG 8 BUFGCTRLs
	Intrp_1K	16.31	2.62485	3 BUFG 8 BUFGCTRLs
	Intrp_4K	18.87	2.62642	3 BUFG 8 BUFGCTRLs

Table 15: Performance Parameter with Clock Gating on FPGA

Modules		**% Improvement in dynamic power**
USB3_EM	Bulk_1K	15.5%
	Bulk_4K	15.28%
	Intrp_1K	15.5%
	Intrp_4K	16.79%
USB3_DH	Bulk_1K	16.25%
	Bulk_4K	16.92%
	Intrp_1K	16.25%
	Intrp_4K	16.63%
USB3_PED_CU	Bulk_1K	12.12%
	Bulk_4K	14.4%
	Intrp_1K	12.12%
	Intrp_4K	13.6%

Table 16: Percentage Improvement in dynamic Power Dissipation on FPGA

In table 3 and table 4, it has been shown that for different test cases of USB 3.0 protocol; dynamic power consumption has been reduced significantly with slight increase in area deployment. Table-4 shows that approximately on average 15% dynamic power has been reduce with just 3% increase in area overhead. Similarly, the number of clock buffer increases depending upon the clock regions that has been enabled during particular test cases with and without clock gating.

Figure 6.2: Graphical Description of Dynamic Power Dissipation on FPGA

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

We have proposed a novel power efficient, area efficient interrupt free system architecture of USB 3.0 protocol. We have implemented and explained how to achieve better throughput by breaking a bigger task into smaller instructions. Also we have discussed different performance parameters of the system architecture like area, power, complexity, flexibility and throughput. Firstly, we employed different modules of our designed architecture and got results in terms of number of gate count, toggle factor and power consumption on ASIC technology (UMC 90nm) with and without root clock gating. Secondly, we have evaluated results using Virtex-5 FPGA with and without clock gating. Experiments show that on average power consumption has been reduced up to 12% with a slight increase in area of approximately 2% in ASIC technology, Similarly, dynamic power consumption has reduced 15% with an area deployment decreases up to 3% in Xilinx Virtex-5 FPGA. The efficient design of micro-coded instruction based architecture results into simpler hardware with better performance due to less CPU interrupts. Furthermore, the generic nature of micro-coded instructions adds flexibility to adapt the design for some other high speed standards like PCI Express and Intel light peak.

7.2 Future Work

The future work in this area could be to extend this system architecture to other high speed express standards like PCI express and Intel light peak. Also in USB 3.0 context system architecture, we could apply root clock gating to the link layer modules and further optimize the power consumption. Also, we are working on other performance parameters of the designed system architecture like effective throughput.

REFERENCES

[1] "Universal Serial Bus (USB) 3.0 Specifications", Hewlett Package Company, Intel Corporation, Microsoft Corporation, NEC Corporation, ST-NXP Wireless and Texas Instruments, version 1.0,November 2008

[2] A.F. Harvey, "DMA Fundamentals on Various PC Platforms" National Instruments Corporation, 1994

[3] Comisky, D, Agarwala, S, Fuoco, C," A scalable high-performance DMA architecture for DSP applications" International Conference on Computer Design (2000)

[4] Peng Yu, Li Bo, Liu datong,Peng Xiyuan " A high Speed DMA Transaction Method for PCI Express Devices" IEEE International Conference of Circuits and system on testing Diagnosis (ICTD 2009)

[5] Dan Tang, Yungang Bao, Weiwu Hu, Mingyu Chen" DMA Cache: Using On-Chip Storage to Architecturally Separate I/O Data from CPU Data for Improving I/O Performance" IEEE High performance Computer Architecture (HPCA-10)

[6] Lucas Kreger-Stickles, Mark Oskin," Microcoded Architectures for Ion-Trap Quantum Computers" International Symposium on Computer Architecture (ISCA 2008)

[7] "AMBA Specifications", ARM limited, version 2.0, August 1999

[8] Qiang Wang, Subodh Gupta, Jason Anderson"Clock Power Reduction for Virtex-5 FPGAs" ACM FPGA 2009

[9] www.xilinx.com/xilinx vertix-5 FPGA data sheet

[10] Qi Wang & Sumit Roy," Power Minimization by Root Clock Gating" IEEE proceedings of the ASP\-DAC 2003

[11] Qi Wang, Sumit Roy, "RTL Power Optimization with Gate-level Accuracy". International

Conference on Computer Aided Design (ICCAD'03)

[12] Gareth Morris,” AMBA based DMA controller specification” version 1.0,March 2001

[13]www.faraday.com.tw/AIP/doc/90nm/SP/FSD0A_A_GENERIC_CORE_ProdBrief_v1.0.pd

[14] Modelsim SE 6.5 user Manual

[15] Juanjuan Chen, Xing Wei, Yunjian Jiang, Qiang Zhou(2009)” Improve Clock Gating through Power-Optimal Enable Function Selection” ACM Design Automation Conference

[16] Pietro Babighian , Luca benini, and Enrico Macii (2005) “A Scalable Algorithm for RTL Insertion of Gated Clocks Based on ODCs Computation” IEEE Transaction on Computer Aided Design of Integrated circuits and Systems Vol. 24 No. 1

[17] Chia-Ming Chang, Shih_Hsu Huang, Yuan-Kai Ho,Jia-Zong lin, Hsin-Po Wang,Yu -Sheng Lu (2008).”Type –Matching Clock Tree for Zero Skew Clock Gating” ACM Design Automation Conference

[18] Anand Raghunathan, Sujit Dey and niraj K. Jha (1999). “RTL power optimization with emphasis on glitch analysis and reduction” IEEE Transactions on Computer Aided Design of Integrated circuits and Systems Vol. 18, No. 8