DESIGN AND IMPLEMENTATION OF CLOCKED OPEN CORE PROTOCOL INTERFACES FOR INTELLECTUAL PROPERTY CORES AND ON-CHIP NETWORK FABRIC

by

Raghu Prasad Gudla

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

The University of Utah

May 2011

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

This thesis of **Raghu Prasad Gudla**

has been approved by the following supervisory committee members:

Kenneth S. Stevens, Chair **01/14/2011**
Date Approved

Alan L. Davis, Member **01/14/2011**
Date Approved

Erik L. Brunvand, Member **01/14/2011**
Date Approved

and by **Gianluca Lazzi**, Chair of the Department of **Electrical and Computer Engineering** and by Charles A. Wight, Dean of the Graduate School.

ABSTRACT

This thesis designs, implements, and evaluates modular *Open Core Protocol* (OCP) interfaces for *Intellectual Property* (IP) cores and *Network-on-Chip* (NoC) that reduces *System-On-Chip* (SoC) design time and enables research on different architectural sequencing control methods. To utilize the NoCs design time optimization feature at the boundaries, a standardized industry socket was required, which can address the SoC shorter time-to-market requirements, design issues, and also the subsequent reuse of developed IP cores. OCP is an open industry standard socket interface specification used in this research to enable the IP cores reusability across multiple SoC designs. This research work designs and implements clocked OCP interfaces between IP cores and On-Chip Network Fabric (NoC), in single- and multi-frequency clocked domains. The NoC interfaces between IP cores and on-chip network fabric are implemented using the standard network interface structure. It consists of back-end and front-end submodules corresponding to customized interfaces to IP cores or network fabric and OCP Master and Slave entities, respectively. A generic domain interface (DI) protocol is designed which acts as the bridge between back-end and front-end submodules for synchronization and data flow control.

Clocked OCP interfaces are synthesized, placed and routed using IBM's 65nm process technology. The implemented designs are verified for OCP compliance using SOLV (Sonics OCP Library for Verification). Finally, this thesis reports the performance metrics such as design target frequency of operation, latency, area, energy per transaction, and maximum bandwidth across network on-chip for single- and multi-frequency clocked designs.

CONTENTS

CHAPTERS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to convey my gratitude to all who made this thesis possible.

First and foremost, I want to thank my advisor Dr. Kenneth Stevens, whose guidance and sound advice has enabled me to accomplish this research. Above all, I appreciate his unflinching support, contributions of time and ideas in making my MS study experience productive and stimulating. Over this period, I have imbibed a few of his great qualities and will always look up to him as an excellent example of a professor.

I gratefully acknowledge Dr. Al Davis and Dr. Erik Brunvand for their supervision and valuable input during the initial stages of the research. I owe my deepest gratitude to the University of Utah, which has contributed in all possible ways to my success.

I thank the Open Core Protocol International Partnership (OCP-IP) organization and Sonics Corporation for providing OCP specifications, tools, and support. I would like to thank my advisor for supporting my research financially on behalf of the Semiconductor Research Corporation (SRC) and National Science Foundation (NSF).

I am indebted to thank especially my research group colleagues, Junbok You and Daniel Gebhardt, in realization of design and timely guidance during research accomplishment. I take this opportunity to thank my other research group colleagues, Krishnaji, Vikas, Shomit, Eliyah, and Yang Xu. I extend my thanks to all my friends in Salt Lake City and in India.

Lastly, and most importantly, I wish to thank my parents and entire extended family for their love and encouragement. Of course above all, I thank whole heartedly God for ubiquitous support in my life. For my parents who raised me, taught me, and loved me for what I am, I dedicate this thesis to them. Thank you.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Current technology trends, scaling, and with end users showing a marked preference for the smaller geometries of deep submicron processes forces a design style where multiple independent circuit implementations are integrated together into a single *System-On-Chip* (SoC). However, contemporary SoC designs have their own share of issues and challenges. The major challenges faced by a design engineer include the ever increasing complexity in modern SoC designs, reusability, time-to-market, communication between *Intellectual Property* (IP) cores, integration of different clocked domain IP cores, and global clock distributions on a chip. The design of standard *Network-on-Chip* (NoC) interfaces to SoC is pivotal in addressing design reusability, time-to-market, and integration of IP cores employing different clock domains (synchronous, elastic, and asynchronous).

I became motivated to take up this prospective research from knowledge learned through academic experience in *Very Large Scale Integrated Circuit* (VLSI) design, verification, and testing domains. This research is mainly targeted to provide an efficient solution to address SoC design challenges by building standard NoC interfaces using an industry standard socket interface, *Open Core Protocol* (OCP). Also listed below are a few motivating factors from an industrial perspective in the realization and implementation of this project:

a. **Levels of device integration lead us to SoC design style**

SoC provides the platform for integration of different architectural cores such as microprocessor chips, *application specific integrated circuit* (ASIC) chips, random access memory (RAM) chips, and peripherals on a single die. The major advantage of SoC design over custom design is its shorter time-to-market, but at the expense of performance and area. SoC designs help enable IP reusability when they utilize a standard communication interface. Employing a standard socket interface on a SoC enables reuse of the good designs with minimal modification to IP cores [1]. This project targets the shorter time-to-market feature of SoCs to build standard interfaces between IP cores at the expense of power, performance, and area.

b. **NoC is the best method of communication on SoC designs**

With the ever growing complexity in SoC designs and the need for better performance, an efficient communication medium is needed. SoC designs can use a NoC or on-chip bus as the on-chip communication medium between IP cores. Network-on-chip is an efficient communication medium compared to bus because of its advantages like the following [2]:

- Efficiency improvement in speed, bandwidth, area, and power consumption
- Supports concurrency - effective spatial reuse of resources
- Low latency
- Scalable bandwidth
- Modularity

c. **OCP a standard socket interface to address IP core reusability**

OCP is a standard core-centric protocol which addresses the IP cores reusability [3]. This not only allows independent IP core development without IP core interconnect network but also allows IP core development in parallel with a system design, reducing design time, design risk, and manufacturability costs.

d. **Design implementation supports different IP core architectures and enables research on NoC**

Building standard NoC interfaces using OCP between IP cores and on-chip network fabric not only supports different architectural designs but also gives a chance to research different NoC architectures by employing different clocking strategies. Commercially available IP cores with OCP are typically synchronous in nature, but the IP cores with wrapper OCP interfaces enables us to have clocked, elastic, or asynchronous interfaces which will support different IP core architectures across multiple SoC designs.

1.2 Related Work

Over the years, relentless advances in semiconductor fabrication and continuous increase in the complexity of modern SoC designs led to integration of more IP blocks into a chip. Figure 1.1 shows a typical SoC architecture design [4]. Hundreds of IPs are integrated into an SoC providing various functionalities including inter-IP core communications, networking, multimedia, storage, etc.

An open and flexible standard interface protocol such as bus-centric or core-centric protocol for IP cores and NoC is necessary to address design reusability, time-to-market, efficient on-chip intercommunication, SoC integration and verification. Semiconductor IP core designers are striving to ensure their IP can be used in the widest possible range of applications but integration of these IPs on SoC is an issue. SoC integration of third party IPs with different interface standards requires an external adapter or bridge to connect them to a standard protocol. Designing such adapters or bridges is not a difficult task but verification is an issue due to standard translation and compliance checks.

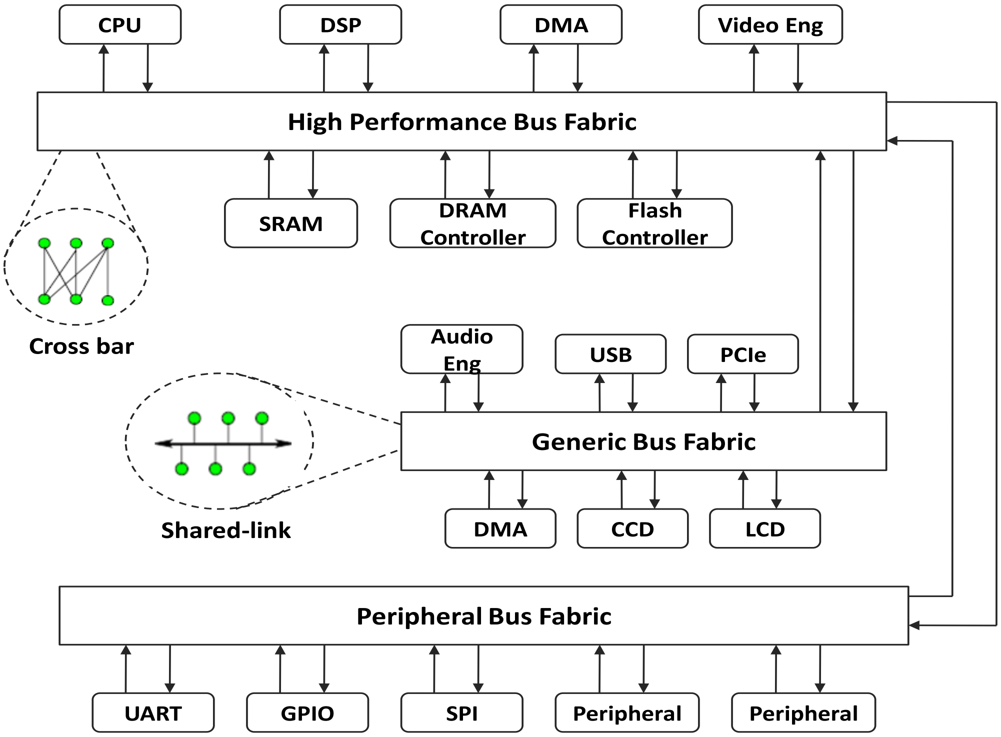


Figure 1.1. An SoC Architecture

1.2.1 Bus-centric Protocols

Bus-centric protocols define technology independent standard bus protocol methodologies for easy integration of IPs within an SoC [5] [6]. Bus protocols are based upon a printed circuit board style of interconnect structures that consist of hierarchical wire bundles and are proving to be ineffective communication for complex SoC designs. All the bus protocols strictly define an interblock data flow communication methodology. Also bus protocols typically do not support sideband control (Reset, Control, Status, Error, and Interrupt signals) and test signals (Scan, JTAG signals) which create a loss of features or performance on interfacing with another bus/core-centric protocol.

The *Advanced Microcontroller Bus Architecture* (AMBA) was developed by ARM Ltd. and is widely used as the on-chip communication bus standard in SoC designs [7]. AMBA is one solution to interface IP cores with each other on an SoC and also enables development of multiprocessor designs with large numbers of controllers and peripherals. The AMBA bus provides design reusability by defining a common backbone for SoC modules using AXI, AHB, APB, ASB, and ATB specifications, each targeted to meeting different requirements.

CoreConnect is an IBM microprocessor bus architecture specification for interconnecting IP cores and custom logic on SoC designs [8]. CoreConnect has similar bridging capabilities to that of AMBA bus architecture, allowing reuse of existing SoC cores (processor, system, and peripheral cores). The CoreConnect architecture provides three buses: a processor local bus (PLB) for connecting high performance peripherals (low latency), an on-chip peripheral bus (OPB) to connect the slower peripheral cores (reduces the traffic on PLB), and a device control register bus (DCR) designed to transfer data between the CPUs general purpose registers, and the DCR slave logic device control registers.

Wishbone is an open source interconnection bus architecture from Silicore Corporation intended to interface IP cores with each other on an SoC and provide reusability by creating a common data exchange protocol between IP cores [9]. It is a simple one bus compact architecture for all applications. Because of its simplicity and flexibility, it is utilized in simple embedded controllers and high performance systems.

All three buses, AMBA, CoreConnect, and Wishbone, are fully synchronous in nature using the rising edge of the clock to drive and sample all signals [10]. The differences are in the supporting features of the specification depending on the choice of system buses used by the designer in case of AMBA and CoreConnect. A designer might face problems integrating different interfaces of the interconnects. Bridges or adapters are required to build a complete system in case of interconnect incompatibility. Wishbone connects all its IP cores to the same standard interface and a system designer can always have the flexibility to choose two wishbone interfaces for implementation in a microcontroller core, one for high-speed low-latency devices and one for low-speed, low-performance devices.

Bus-centric protocols are targeted for single unique application, and the interface circuitry is defined for that particular application. Any changes in the design application requires redesign of the arbitration logic and interface circuitry for the new application. Incompatibility between the chosen interconnect system and the bus-centric native protocol requires multiple bus bridges for communication, and limits the maximum utilization of IP capabilities [11]. Also, whenever there are differences in data and address presentation sequences between an IP core's native bus and target bus, the IP core's performance will likely suffer due to the bridge-on-a-bridge effect of having to correlate the signaling between the two disparate bus architectures. Also, one needs to compromise on the bridge gate count implementation which is likely to be higher. Selecting a bus-centric protocol as an IP core's native interface will ultimately limit its reusability compared to core-centric protocol. A socket can fulfill the reusability requirement virtually across any application, and the process can also be automated.

1.2.2 Core-centric Protocols

The solution to maximize IP core's reusability while still exploring the advantages of proven industry standard interfaces (sockets) is to adopt a well-specified core-centric protocol as an IP core's native interface [1] [6]. Sockets are universally accepted and are targeted to support virtually any application because of their design reusability and verification features. Also, sockets provide complete interface specification between IP cores and on-chip interconnect systems which enables the designers to independently develop individual SoC cores, reducing overall SoCs development time, effectively decreasing time-to-market.

Basically, the individual IP cores are designed and developed simultaneously by decoupling them from the system in which they reside. The final SoC is developed in parallel and the design time is reduced to that of the longest effort required in a single element design or the SoC integration. System designers are also benefited by not having to consider other diverse core protocols and delivery styles. Use of a standard IP core interface eliminates having to adapt each core for every SoC integration, and instead allows the system designer to focus on system level design issues. Also, since the IP cores are decoupled from the on-chip interconnect and from each other, it is easier to swap one core for another in meeting changing requirements.

For an IP core to be truly reusable, it must remain untouched as it moves from one system to another system and its interface must match continuously differing requirements of each systems interconnect. A standard IP core interface specification must be scalable and configurable to adapt to the wide range of requirements. Also, it must be able to capture the non-dataflow signals (such as reset, interrupts, error, and flow control signals) along with the dataflow signals. Following are a couple of core-centric protocols:

The *Open Core Protocol* (OCP) is an open standard, bus-independent protocol provided by Open Core Protocol-International Partnership (OCP-IP). It meets all the core centric requirements and is one of the protocols which unifies all the intercore communications including sideband control and test signals [3]. OCP defines a high performance, complete interface socket between IP cores facilitating design reuse, and also reduces design time, design risk, and manufacturing costs for SoC designs. By adopting OCP, IP cores can be developed independent of interconnect topology and implementation. Figure 1.2 shows IP cores with OCP interfaces connected with different interconnect topologies [4]. OCP supports very high-performance data transfer models ranging from simple request-grants through burst pipelined and tagging objects. OCP protocol compliance verification is one of the distinguishing feature from other protocols. OCP-IP not only provides the specification and its

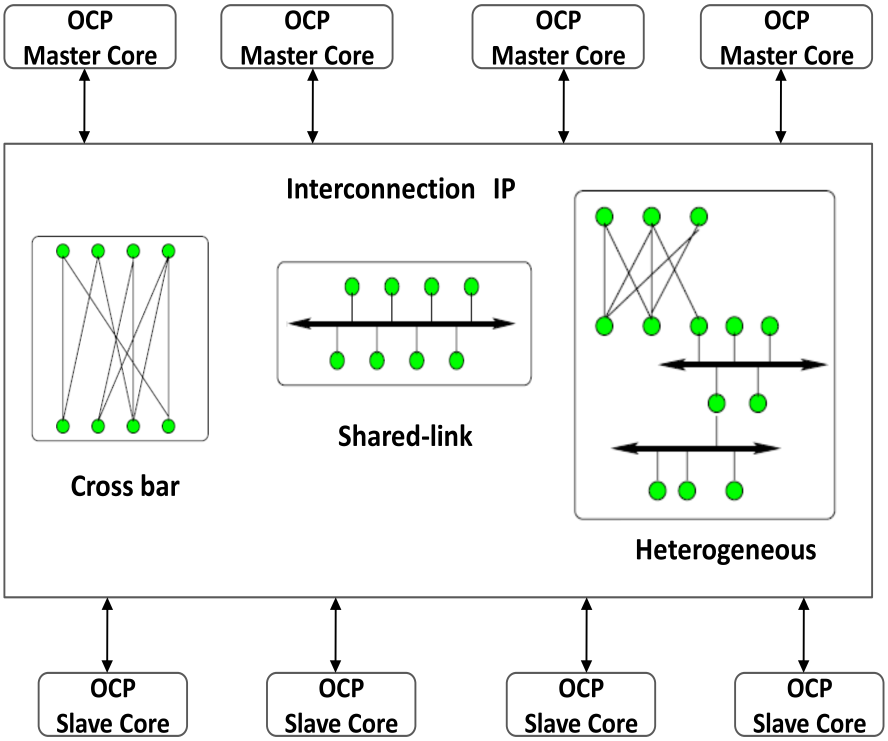


Figure 1.2. IP Cores with OCP Interfaces Using Hierarchical and Heterogeneous Interconnect System

member-driven evolution, but also industrial grade tools and services that ensure its members can rapidly confirm compliance and maximize their productivity.

IP cores using native OCP interfaces can easily communicate with any bus architecture or on-chip network fabric through simple bridge structures. Even with bus bridges, IP cores can utilize its maximum capabilities using OCP. If the chosen interconnect system cannot interface directly to OCP, the IP developer can design and develop bus bridges (wrapper interfaces) for common bus architectures that a customer may choose.

The *Virtual Component Interface* (VCI) is another core-centric protocol provided by Virtual Socket Interface alliance (VSIA) group [12]. VCI is similar to OCP in capability and philosophy. But VCI supports only dataflow aspects of core communications compared to OCP, which is a superset of VCI supporting configurable sideband control signaling and test signals.

1.3 Objectives

This research will design and implement clocked OCP interfaces between IP cores and on-chip network fabrics for single- and multifrequency (*Globally Asynchronous Locally Synchronous architectures* (GALS)) domains including the following:

- Design and implementation of customized back-ends to IP cores and NoC.
- Design and implementation of front-ends (OCP Master and Slave entities).
- Design and implementation of a generic Domain Interface (DI) module.
- Customize and build asynchronous dual clocked First-in-First-out memory structures (FIFOs) for the DI module used in GALS architectures.
- Determine the best placement of buffering and synchronization across the design path.
- Synthesize clocked designs with IBM's 65nm process technology using Synopsys Design Compiler (DC).
- Automatic Place and Route of clocked designs using Cadence SoC Encounter.
- Design validation and derivation of performance metrics for clocked designs in nonsplit and split operation modes.
- Perform power analysis on different clocked designs using Synopsys PrimeTime PX for deriving energy per transaction values.
- Build parameterized computational models for estimating the performance of clocked design configurations.
- Performance evaluation of clocked designs and NoC.
- Performance comparison between single- and multifrequency clocked designs.
- Verification of OCP compliant IP-cores using the SOLV component.

1.4 Contributions

Following are the key contributions from this research:

i. Designs and implements industry standard clocked OCP interfaces for single-frequency clocked domain and GALS architectures.

ii. A novel modular architecture is adopted that provides high design reusability, meeting SoC shorter time-to-market requirements, and simplifying design validation.

iii. The implemented OCP interfaces enable future research:
- Deriving elastic and asynchronous OCP implementations.
- Evaluating NoC performance from end-to-end.
- Research and comparison of different architectural sequencing control methods for IP cores and NoC (clocked, elastic, and asynchronous).

1.5 Thesis Overview

The rest of this thesis is organized into five chapters which give a detailed description of the research. Chapter 2 presents prerequisite information on current architectural design issues and challenges, and its solution with building of standard NoC interfaces using OCP. A brief description follows on OCP, including its operation, key features, and protocol signals.

Chapter 3 briefly describes the project design and its specifications. An overview of the design structure is presented, including the modularization of components across the design and network interface structure, synchronization and buffering, and data packetization and depacketization mechanisms.

Chapter 4 explains the design implementation in single- and multifrequency clocked domains. The design implementation includes a brief description about the steps involved in the implementation of the two clocked domains and the functionality of IP cores and NoC back-ends, OCP entities, and the DI module. Chapter 5 describes the verification methodology used to determine OCP compliance of IP cores.

Chapter 6 explains the different testing strategies and performance metrics used for comparison. The maximum target frequency in single-frequency and multifrequency clocked domains is determined. Other performance metrics such as latency, energy per flit, area, and maximum bandwidth across NoC are also tabulated. Different design parameters are varied to determine the design maximum and minimum limits on performance in case of design expansion, and worst and best scenarios.

Finally, Chapter 7 summarizes the research work, including the project scope, application, and areas of future research. The Appendix includes detailed information about desynchronization and elasticization of clocked NoC interfaces and the complete set of OCP protocol signals.

CHAPTER 2

BACKGROUND

2.1 Existing Architectural Designs with OCP

The existing OCP interfaces for IP cores can be classified into two categories depending on how OCP is implemented with respect to the IP core functionality and communication interface. If the OCP interface is integrated as part of the IP core, we refer to it as *Native OCP interface*; otherwise, we call it *Wrapped OCP interface* [3]. In case of native OCP interfaces, the OCP interfaces are integrated as part of the IP core and are typically developed along with the new IP cores. Figure 2.1 illustrates IP cores with native OCP interfaces. In case of wrapped OCP interfaces, a wrapper (bridge) OCP interface is placed around the IP core's native interface to communicate to the outside world. The additional wrapper OCP interface basically maps the signals and values of the existing IP core to OCP compatible signals and values. Essentially, this wrapped interface is an exercise in protocol conversion between an IP core's existing protocol and OCP as it must implement flow control and at times executes commands not part of the base IP core functionality. The conversion logic can range from a few gates in complexity to a very complicated interface. However,

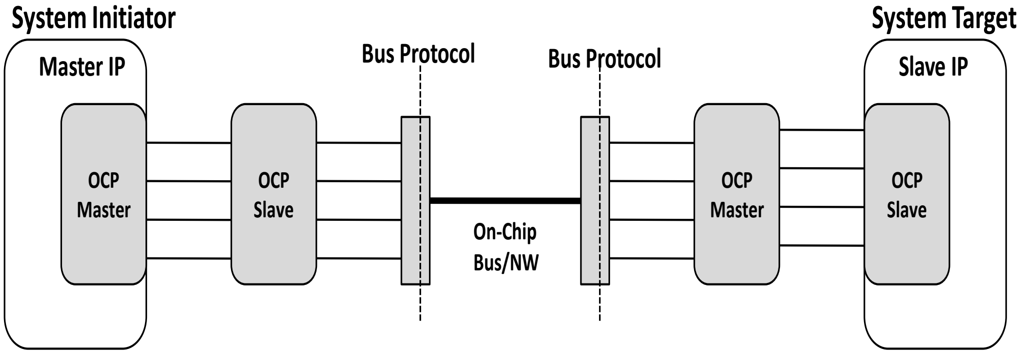


Figure 2.1. IP Cores with Native OCP Interfaces

from the viewpoint of any IP core or an on-chip network fabric, which uses OCP to communicate with the another IP core, the type of OCP interface (native or wrapped) does not matter; it is all just using the OCP protocol.

2.2 Building Standard NoC Interfaces Using OCP

This research provides the solution for interfacing different architectural IP cores on SoC by building standard NoC interfaces using OCP. Current architectural designs which utilize OCP for interfacing with other IP cores are synchronous in nature and typically OCP is integrated (native OCP) into the IP core. As OCP is a clocked protocol, integration of OCP onto IP cores which employ clocking strategies like elastic protocols and asynchronous handshaking have not been investigated. In order to explore elastic and asynchronous clocking strategies, this project will implement clocked wrapper OCP interfaces (OCP located outside the IP core). Figure 2.2 illustrates IP cores with clocked wrapper OCP interfaces. A significant portion of the research will address the issue of simplifying the interfacing of different architectural IP cores to OCP by the following:

a. Design and implement customized back-end interfaces to IP cores (such as pipelined processor bus and synchronous memory) and on-chip network fabrics.

b. Investigate the possibility of a generic OCP wrapper interface protocol, called the domain interface (DI).

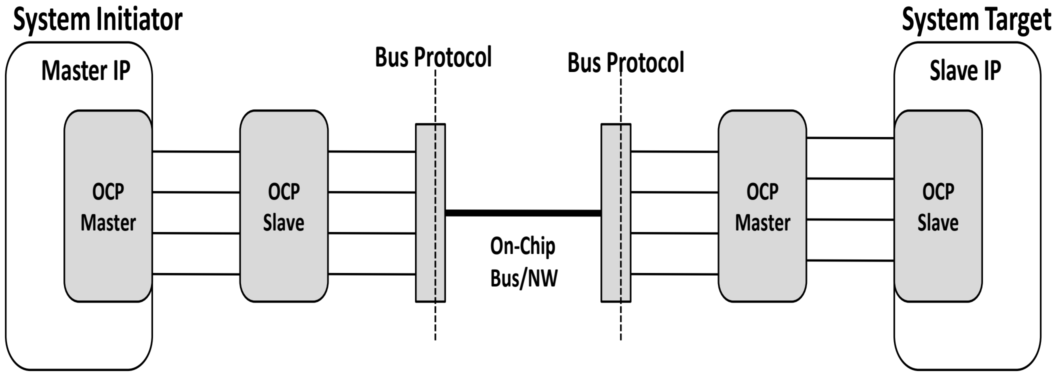


Figure 2.2. IP Cores with Clocked Wrapper OCP Interfaces

2.3 OCP Description

2.3.1 Introduction to OCP

The Open Core Protocol is an openly licensed, core-centric protocol standard, which defines a high performance, synchronous, bus independent configurable interface for communication between IP cores and NoC [3][1]. It is an efficient point-to-point connection standard and because of its configurability, scalability, and generality, it has been widely accepted from low-power to high-performance applications. It can be optimized to use only the necessary features required for communicating between any two components, which saves chip area. It dramatically improves reusability of IP cores independent of the architecture and design of the systems, which leads directly to a more predictable, productive SoC designs and also simplifies the system verification and testing. OCP consists of an aggregation of signals that aims to unify the communication among IP blocks, reducing the system design time significantly. It is comprised of a continuum of communication protocols that share a common definition for the whole system where it ensures dramatic time reduction of functional verification for any future releases of the system.

2.3.2 OCP Operation and Its Key Features

OCP defines a point-to-point interface between two communicating entities such as IP cores and bus interface modules (bus wrappers). One entity acts as the master of the OCP instance, and the other as the slave. Only the master can present commands and is the controlling entity [3]. The slave responds to commands presented to it, either by accepting data from the master, or presenting requested data to the master. For two entities to communicate in a peer-to-peer fashion, there needs to be two OCP instances connecting them, one where the first entity is a master, and one where the other entity is a slave.

The characteristics of the IP core determine whether the core needs master, slave, or both sides of the OCP. The bus wrapper interface modules must act as the complementary side of OCP for each connected entity. Depending on the direction and type of the signal (like MCmd:Master command) one can classify whether the module is a master or slave. Figure 2.3 shows a simple system containing an on-chip

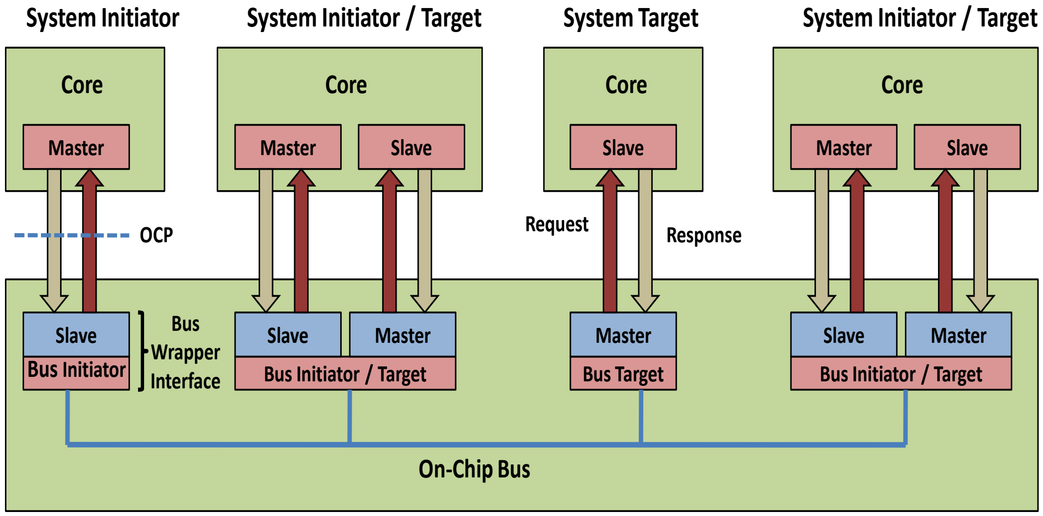


Figure 2.3. A Simple System with OCP between IP Cores and On-Chip Bus

bus and four IP core entities: one that is a system target, one that is a system initiator, and the other two that have both. Different OCP transactions are possible between system initiator and system target with optional response. In one OCP transaction type, the Master IP (system initiator) presents a request command without expecting a response from Slave IP. Another type expects a response from the Slave IP in the same path. A third case starts like the first, but the Slave IP issues a response using its OCP master interface to send a message back to the initiating IP.

Each transfer across this system occurs as follows. A system initiator (as the OCP master) presents command, control, and possibly data to its connected slave. The interface module presents the request across the on-chip bus system. OCP does not specify the embedded bus functionality. Instead, the interface designer converts the OCP request into an embedded bus transfer. The receiving bus interface module (as the OCP master) converts the embedded bus operation into a legal OCP command. The system target (OCP slave) receives the command and takes the requested action. Some of the OCP key features include the following [3]:

a. **Point-to-Point Synchronous Interface**:

   To simplify timing analysis, physical design, and general comprehension, OCP is composed of unidirectional signals driven with respect to, and sampled by, the rising edge of the OCP clock.

b. **Operational Commands**:

There are two basic commands, Read and Write, and five command extensions. Extensions include ReadExclusive, ReadLinked, WriteNonPost, WriteConditional, and Broadcast.

c. **Configurable Dataflow Signals (Address, Data, Control)**:

To increase transfer efficiencies, OCP supports a configurable data width to allow multiple bytes to be transferred simultaneously. OCP supports word sizes of power-of-two and non-power-of-two (byte addressable, word aligned) and also supports transfers of less than a full word of data by providing byte enable information that specifies which octets are to be transferred.

d. **Configurable Sideband Signals (Interrupts and Errors)**:

Different types of control signaling are required to coordinate data transfers (for instance, high-level flow control) or signal system events (such as interrupts). Many devices also require the ability to notify the system of errors that may be unrelated to address/data transfers. OCP refers to all such communication as sideband (or out-of-band) signaling, since it is not directly related to the protocol state machines of the dataflow portion of OCP.

e. **Pipelining Transfer Support**:

OCP allows pipelining of transfers to improve bandwidth and latency characteristics also known as *split transactions.* To support this feature, the return of read data and the provision of write data may be delayed after the presentation of the associated request.

f. **Burst Transfer Support**:

To provide high transfer efficiency, burst support is essential for many IP cores. A burst is a set of transfers that are linked together into a transaction having a defined address sequence and number of transfers.

g. **In-band Information**:

Using in-band signals, OCP can pass core-specific information along with the other information during transactions. In-band extensions exist for requests and responses, as well as read and write data. A typical use of in-band extensions is to pass cacheable information or data parity.

h. **Tagging or Out-of-Order Completion Support**:

   To support out-of-order responses and to commit writes out-of-order, tagging is used as long as the transactions target addresses are different. Without tags, a slave must return responses in the order that the requests were issued by the master. The tag links the response back to the original request.

i. **Multithreading and Connections**:

   Out-of-order request and response delivery can also be enabled using multiple threads. Concurrency is also supported at the expense of having an independent flow control for each thread, eliminating ordering restrictions for transactions on different threads. The notion of a thread is a local concept between a master and a slave communicating over OCP. Thread information is passed from initiator to target using connection identifiers. Connection information helps to identify the initiator and determine priorities or access permissions at the target.

2.3.3 OCP Signals and Encoding

OCP interfaces are synchronous, employing a single clock signal. Thus, all its signals are driven with respect to, and sampled by, the rising edge of the clock, except reset [3] [11]. OCP interface signals are divided into three categories: data flow, sideband (error, interrupt, flag, control, and status), and test signals (scan, JTAG). With the exception of the clock, all OCP signals are unidirectional and point-to-point. The rising edge of the OCP clock signal is used to sample other OCP signals to advance the state of the interface. The Clk and EnableClk signals are required inputs in both masters and slaves and they are driven by a third entity (neither the masters nor the slaves). When the EnableClk signal is not present, the OCP clock is simply the Clk signal.

2.3.3.1 Dataflow Signals

Dataflow signals consist of a set of signals, some of which are used for data transfers, while others are configured to support any additional communication requirements between the master and slave components. Dataflow signals can be divided into the following categories:

a. **Basic Signals**:

These include the clock, address, read and write data, transfer type, and handshaking/response signals between the master and the slave. Only the clock and transfer type signals (MCmd) are mandatory for an OCP interface, the remaining signals being optional. Table 2.1 lists the OCP basic signals. The widths of the address, read data, and write data are configurable, and not limited to being multiples of eight. The transfer type indicates the type of data transfer operation issued by a thread running on a master, and can be any one of the following:

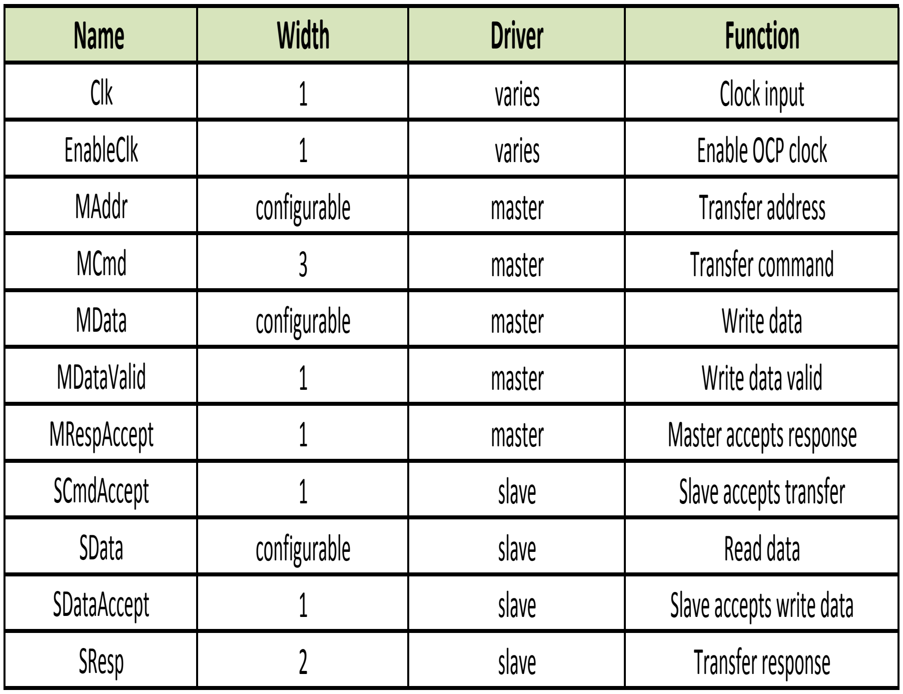
i. **Idle**: No operation is required to be performed.

ii. **Read**: Reads data from the addressed location in a slave.

iii. **Write**: Writes data to the addressed location in a slave.

iv. **Broadcast**: Writes data to the addressed location using MData field, which may be mapped to more than one slave in a system-dependent way. Broadcast clears the reservations on any conflicting addresses set by other threads.

Table 2.1. OCP Basic Dataflow Signals



Name	Width	Driver	Function
Clk	1	varies	Clock input
EnableClk	1	varies	Enable OCP clock
MAddr	configurable	master	Transfer address
MCmd	3	master	Transfer command
MData	configurable	master	Write data
MDataValid	1	master	Write data valid
MRespAccept	1	master	Master accepts response
SCmdAccept	1	slave	Slave accepts transfer
SData	configurable	slave	Read data
SDataAccept	1	slave	Slave accepts write data
SResp	2	slave	Transfer response

v. **Exclusive Read**: Reads from a location in a slave using SData field and locks it, preventing other masters from writing to the location (exclusive access). The location is unlocked after a write to it from the original master that caused the lock to be set.

vi. **Linked Read**: Reads data from the addressed location in a slave using SData field, and sets a reservation in a monitor for the corresponding thread, for the addressed location. Read or write requests from other masters to the reserved location are not blocked from proceeding, but may clear the reservation.

vii. **Nonposted write**: Writes data to the addressed location in a slave using MData field, unlocking the location if it was locked by an exclusive read, and clearing any reservations set by other threads.

viii. **Conditional write**: Only writes to the addressed location in a slave using MData field, if a reservation is set for the corresponding thread. Also clears all reservations on the location. If no reservation is present for the corresponding thread, no write is performed, no reservations are cleared, and a FAIL response is returned.

b. **Simple Extensions**:

These include signals to indicate the address region (e.g. register or memory), byte enables for partial transfers, and core-specific configurable signals that send additional information with the transfer request, read data, write data, and the response from the slave. Configurable signals can transmit information about data byte parity, error correction code values, FIFO full or empty status, and cacheable storage attributes. Simple extension signals include the following:

- **MAddrSpace**: Specifies the address space and is an extension of MAddr basic signal which is used to indicate the address region of transfer.
- **MByteEn**: Indicates which bytes of OCP word are part of the current transfer.
- **MDataByteEn**: Indicates which bytes of OCP word are part of the current write transfer.
- **MDataInfo**: Extra information is sent with the write data.

c. **Burst Extensions**:

These signals are used to support burst transfers which allows the grouping of multiple transfers that have a defined address relationship. Bursts can either include addressing information for each successive command (which simplifies the requirements for address sequencing/burst count processing in the slave), or include addressing information only once for the entire burst. Burst extension signals include the following:

- **MBurstLength**: This field indicates the number of transfers for a row of the burst and stays constant through the burst.
- **MBurstPrecise**: This field indicates whether the precise length of the burst is known at the start of burst.
- **MBurstSeq**: This field indicates sequences of addresses for requests in burst (Incrementing, Custom or user defined, Wrapped, Stream, Exclusive-OR and Unknown).
- **MBurstSingleReq**: This burst has a single request with multiple data transfers.
- **MDataLast**: This field indicates last write data in a burst.

d. **Tag Extensions**:

These signals are used to assign tags (or IDs) to OCP transfers to enable out-of-order responses and to indicate which transfers should be processed in order. Tag numbering begins at 0 and is sequential. The binary encoded TagID must carry a value less than the tag's parameter. Tag extension signals include the following:

- **MTagID**: This field indicates the request tag from the Master IP.
- **MTagInOrder**: This field indicates that the current request cannot be reordered with respect to other requests when this field is asserted.
- **STagID**: This field indicates the response tag from the Slave IP.
- **STagInOrder**: This field indicates that the current response cannot be reordered with respect to other requests when this field is asserted.

e. **Thread Extensions**:

These signals are used to assign IDs to threads in the master and slave, and for a component to indicate which threads are busy and unable to accept any new requests or responses. Thread numbering begins at 0 and is sequential. The binary encoded ThreadID must carry a value less than the thread's parameter. Thread extension signals include the following:

- **MConnID**: This field indicates the connection identifier.
- **MDataThreadID**: This field indicates the write data thread identifier.
- **MThreadBusy**: This field indicates master thread busy.
- **MThreadID**: This field indicates request thread identifier.
- **SThreadBusy**: This field indicates slave thread busy.
- **SThreadID**: This field indicates response thread identifier.

2.3.3.2 Sideband Signals

Sideband signals are optional OCP signals that are not part of the dataflow phases, and can change independent of the request/response flow (but are still synchronous to the rising edge of the clock). These signals are used to transmit control information such as interrupts, resets, errors, and other component specific information like core specific flags. They are also used to exchange status and control information between a component and the rest of the system using Control, ControlBusy, ControlWr, Status, StatusBusy, and StatusRd signals. All sideband signals are optional except for reset (active low). Either the MReset_n or the SReset_n signal must be present.

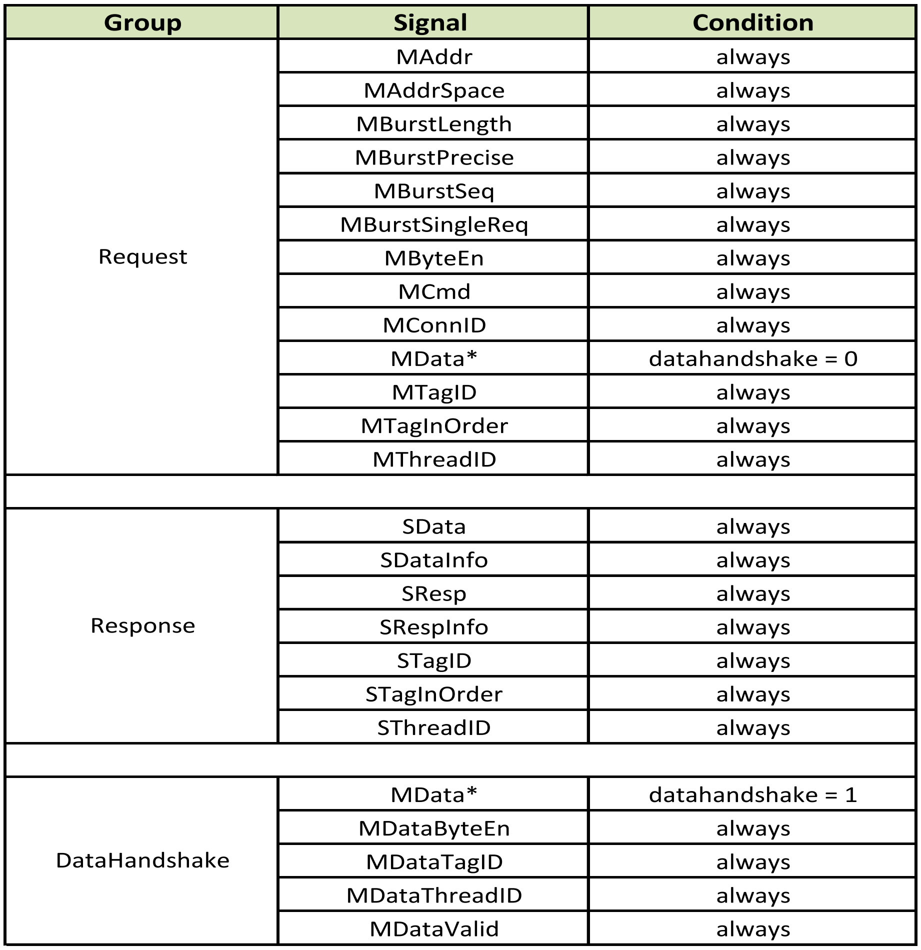
2.3.3.3 Test Signals

The OCP test signals are also a set of optional signals, and are responsible for supporting scan, clock control, and IEEE 1149.1 (JTAG). The scan interface signals include ScanCtrl, Scanin, and Scanout. Debug and test interface signals include TCK, TDI, TDO, TMS, and TRST_N.

2.3.4 OCP Signal Directions and Groups

Depending on the module instance acting as a master or slave, the direction of request/response/datahandshake signals are defined and control signals are defined depending on the module acting as a system or core. Interface types to each module is defined depending on OCP entity and connected system. If a module acts as an OCP master and also a system, it is designated as system master. Some of the OCP signals are grouped together depending upon the active state of the signals at the same time. Dataflow signals are classified into three groups: request, response, and datahandshake signals. The handshake and response signals are optional and can be configured depending on the IP core's communication requirements. Table 2.2 lists the OCP signal groups.

Table 2.2. OCP Signal Groups



Group	Signal	Condition
Request	MAddr	always
	MAddrSpace	always
	MBurstLength	always
	MBurstPrecise	always
	MBurstSeq	always
	MBurstSingleReq	always
	MByteEn	always
	MCmd	always
	MConnID	always
	MData*	datahandshake = 0
	MTagID	always
	MTagInOrder	always
	MThreadID	always
Response	SData	always
	SDataInfo	always
	SResp	always
	SRespInfo	always
	STagID	always
	STagInOrder	always
	SThreadID	always
DataHandshake	MData*	datahandshake = 1
	MDataByteEn	always
	MDataTagID	always
	MDataThreadID	always
	MDataValid	always

2.4 Prior Relevant Research Work

As part of my thesis study, I implemented a portion of OCP in the advanced VLSI course, which helped me learn about a subset of OCP, the importance of OCP interfaces in SoC designs, and its functionality. For the class project, I considered two IP cores (Master as CPU and Slave as Memory system) with native OCP interfaces. The on-chip communication medium between the two IP cores was assumed to be a simple buffering unit with some latency in the path. Using a subset of OCP signals (dataflow signals), simple memory read and write transactions between the two IP cores was implemented. The modules were developed using VHDL. Figure 2.4 shows the block diagram used in this study.

2.5 Design Development, Implementation, and Testing

Building modern digital integrated circuits is a complex process and requires powerful Electronics Design Automation (EDA) and Computer Aided Design (CAD) tools in the design development and implementation [13]. Figure 2.5 illustrates the steps required in this project implementation [14]. The implementation flow follows the standard ASIC design steps which includes specification, front-end, and back-end phases. The front-end phase is comprised of the development of schematics or register transfer level (RTL) Verilog code, and functionality testing of synthesized netlist. This project developments RTL Verilog code in the front-end phase. The back-end phase is comprised of Automatic Place and Route (APR) of the physical design (synthesized

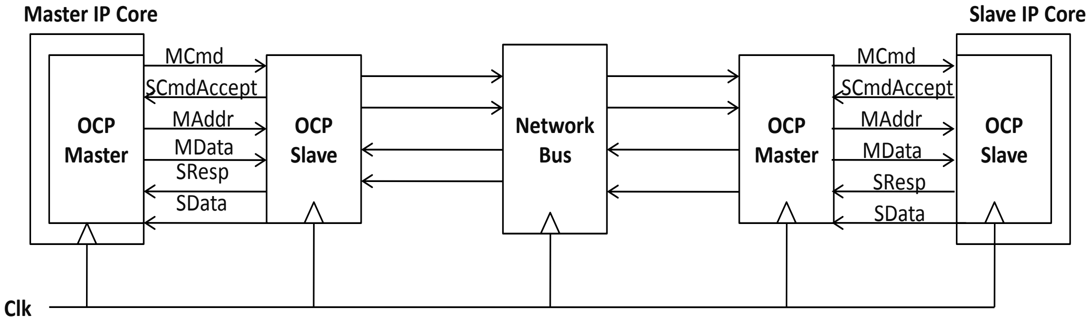


Figure 2.4. IP Cores with Native OCP Interfaces Implementation

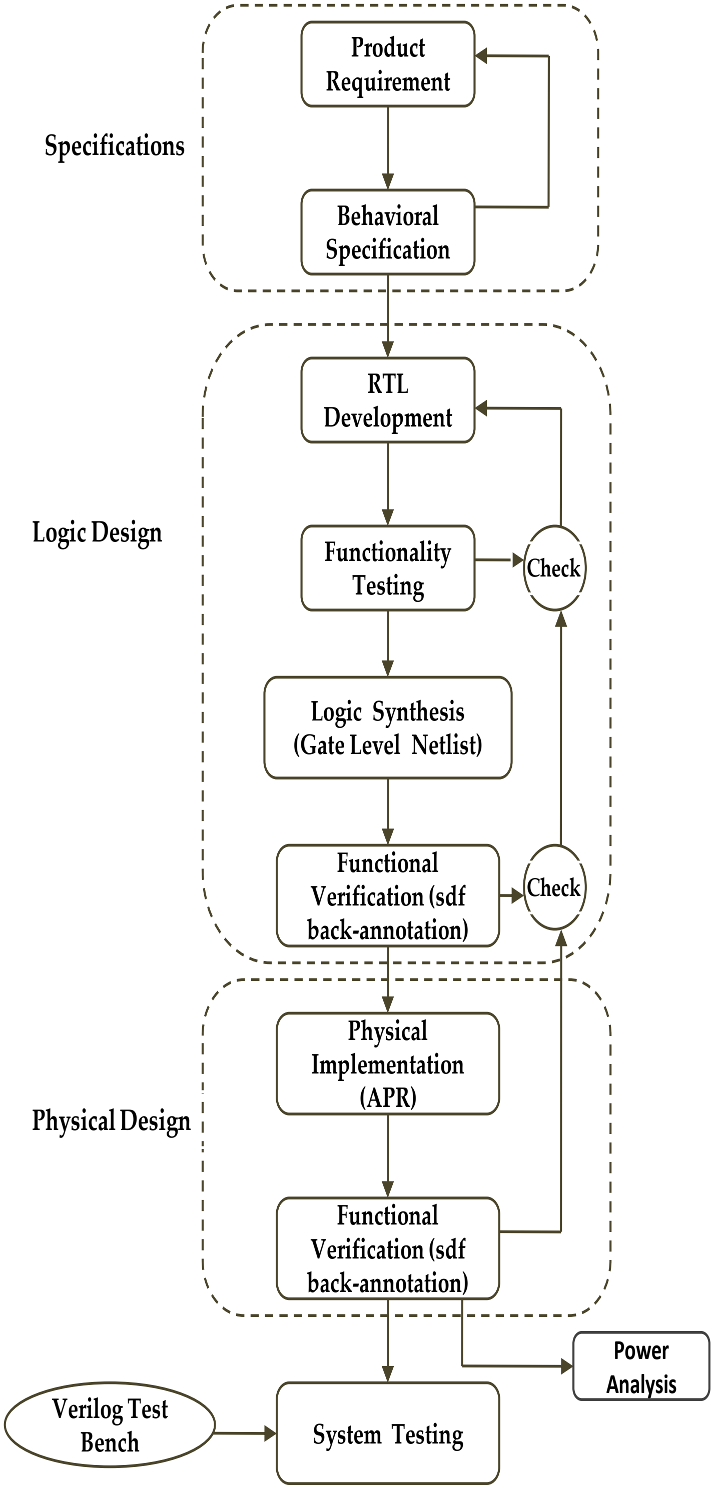


Figure 2.5. Design Implementation Flow

structural Verilog netlist), and functionality verification. Finally, the routed design is tested using a Verilog test bench where different performance metrics are determined.

2.5.1 RTL Development Using Verilog HDL

Verilog HDL is one of the most widely used hardware description languages (HDL) in the development of modern ASICs. It is used to describe the design, verification, and implementation of digital circuits [15]. HDLs differ from other programming languages in describing signal propagation times and their dependencies (sensitivity). RTL employs a constrained HDL format using gate level, transistor level, and behavioral level descriptions, and provides an efficient way of synthesizing any given design. Synthesis productivity advantages have propelled HDL technology into a central role for digital design compared to schematic methodologies because of its features to provide high speed, low power, and the ability to synthesize circuitry. HDL also provides a simulation and debugging environment in validating a design's intended functionality. This project uses Modelsim [16] and NC-Verilog [17] simulators for simulating RTL and synthesized netlists.

2.5.2 Logic Synthesis and Automatic Place and Route

Logic synthesis is the process of converting a RTL description of design into an optimized gate-level representation (structural Verilog) using cells from a standard cell library. The generated structural Verilog is expected to provide the same functionality with extracted delays assigned to each cell from the technology process library. Logic synthesis can generate a ddc file (binary database file for further processing), sdf file (standard delay format file to back-annotate the simulations with extracted timings from cells), sdc file (standard design constraint file for physical implementation of the circuit), and pow file (reports the required power for the design). This project uses IBM's 65nm process technology library and Synopsys Design Compiler tool for synthesizing the design.

Place and Route is the process of converting a structural Verilog description into a physical circuit (layout with pins and metal routing) which involves floorplan synthesis depending on the aspect ratio and cell utilization, power grid and clock tree synthesis, placement of the standard cells, and routing the wiring connections between them. During the placement of standard cells, an abstract view of the cells is used since it has the physical information about the size and shape of the cells, the connection points of the cells (pins), and routing abstractions in those cells. This project uses Cadence SoC Encounter to automatic place and route the design. SoC Encounter requires standard cell characterized file (liberty format file), cell abstract information (.lef file), structural Verilog, and sdc files from Synopsys DC Compiler. Placing and routing the design involves floor planning, power planning, placement of standard cells, pre- and postclock tree synthesis, and final routing. The final routed design is optimized and geometry and connectivity are verified. The Place and Route process generates design exchange format (DEF) file (used to read layout back in layout editor), structural Verilog (final placed and routed circuit), sdf file (standard delay format file to back-annotate the simulations with extracted timings from cells and interconnect), spef file (reports the extracted information on parasitics used in power analysis) and pow_report file (reports leakage power, internal power, and switching power of the design).

2.5.3 Design Testing Environment

This project design uses an emulated pipelined RISC CPU as the Master IP core [18], a synchronous memory block (ROM and RAM) as the Slave IP core, and a NoC built from synchronous routers [19]. A Verilog test bench is used to drive the device under test (DUT) inputs and monitors the outputs. Figure 2.6 illustrates the design testing environment. The test bench generates the required traffic stimulus for the project design testing in both single- and multifrequency clocked domains. In the multifrequency clocked domain, the test bench generates three asynchronous clocks for Master IP, Slave IP, and NoC, respectively.

The test bench is customized to generate pipelined CPU traffic compatible with synchronous memory. Figure 2.7 shows the customized 32-bit CPU address format used in this project implementation. The 32-bit logical address generated from Master IP consists of routing information, physical address, transaction type, operation mode, and burst data.

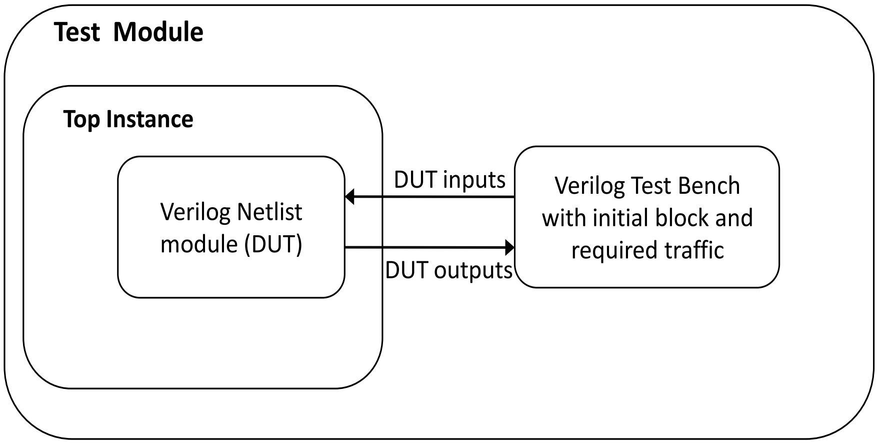

Test Module
Top Instance
Verilog Netlist module (DUT)
DUT inputs
DUT outputs
Verilog Test Bench with initial block and required traffic

Figure 2.6. Design Testing Environment Using Verilog Test Bench

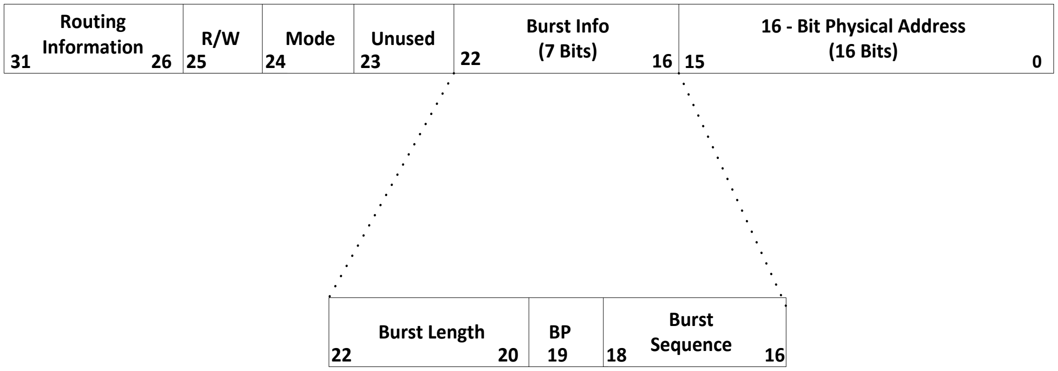


Figure 2.7. Pipelined CPU 32-Bit Address Format

2.5.4 Design Metrics Computation

With technology advancement, power management is one of the major design challenges for deep submicron technologies where timing, power, and signal integrity are interrelated. Power analysis is a crucial aspect in a design flow since it can affect packaging, cooling decisions, device battery life, and cheap reliability. PrimeTime PX (PTPX) is an accurate power analysis tool that includes timing interdependencies for power dissipation that can be used at various stages in the design flow [20]. It also performs static timing analysis and signal integrity analysis.

PrimeTime computes the total power dissipated in a design from dynamic and static power dissipation components. Dynamic power is used during switching of the transistors and short circuit of power rails. Static power dissipation or leakage power occurs when the devices are at steady state. PrimeTime power analysis can be done in a vector-free flow (independent of switching activity) or RTL VCD flow (using switching activity from simulations). In this project, a more accurate RTL VCD flow methodology is chosen for power analysis. Figure 2.8 demonstrates the PrimeTime Power analysis RTL VCD flow used in this project. RTL VCD flow requires the following steps to be executed:

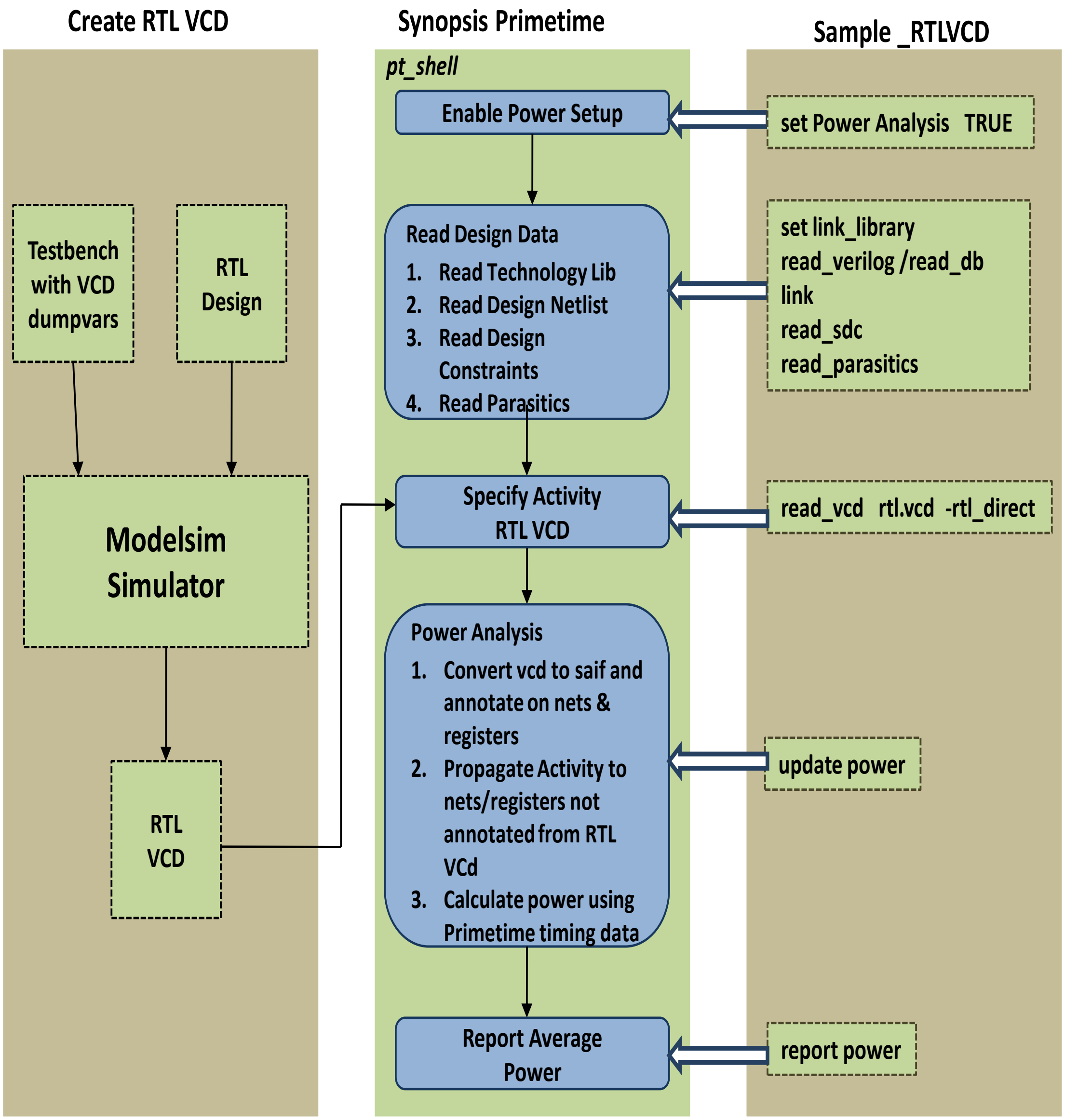


Figure 2.8. PrimeTime (PX) Power Analysis RTL VCD Flow

a. Create a VCD file from Modelsim simulation of the final physically routed design.

b. Enable the power analysis setup and read in the technology library (tech.lib) file, final routed structural Verilog file, design constraints (sdc) file, and parasitics file (spef).

c. Read the VCD file into PrimeTime and specify the activity.

d. Convert the VCD file into SAIF (switching activity interchange format) file and annotate on nets and registers.

e. Propagate activity to nets/registers not annotated from RTL VCD.

f. Calculate the power using PrimeTime timing data.

g. Report average power, timing constraints, and switching activity.

This project reports energy required per each transaction in nonsplit and split modes of operation as one of the performance metrics. Energy per transaction is derived from total average power and the total test time. Energy is the capacity to do work over the time.

$$Total\,Energy \Leftarrow Average\,Power \times Test\,time \tag{2.1}$$

$$Energy/Transaction \Leftarrow (Total\,Energy)/No.\,of\,Transactions \tag{2.2}$$

2.6 Validation of OCP Complaint IP cores

The OCP-IP organization provides the CoreCreator II tool for its members to verify OCP compliant design implementations [21] [11]. Figure 2.9 shows the setup of the OCP CoreCreator tool. The CoreCreator tool automates the tasks of building, simulating, verifying, and packaging OCP compatible cores. It can be used with both traditional Verilog and VHDL test bench environments to create directed tests for OCP designs. The OCP-IP organization provides the Verification IP entities and SOLV (Sonics OCP library for verification) for debugging tools. Debugging tools include an OCP checker to ensure protocol compliance, a performance analyzer to measure system performance, and a disassembler, which helps to view the behavior of OCP traffic. This tool is used in validating this project design flow. One restriction to this work is that the tools are limited to test and verify clocked interface designs.

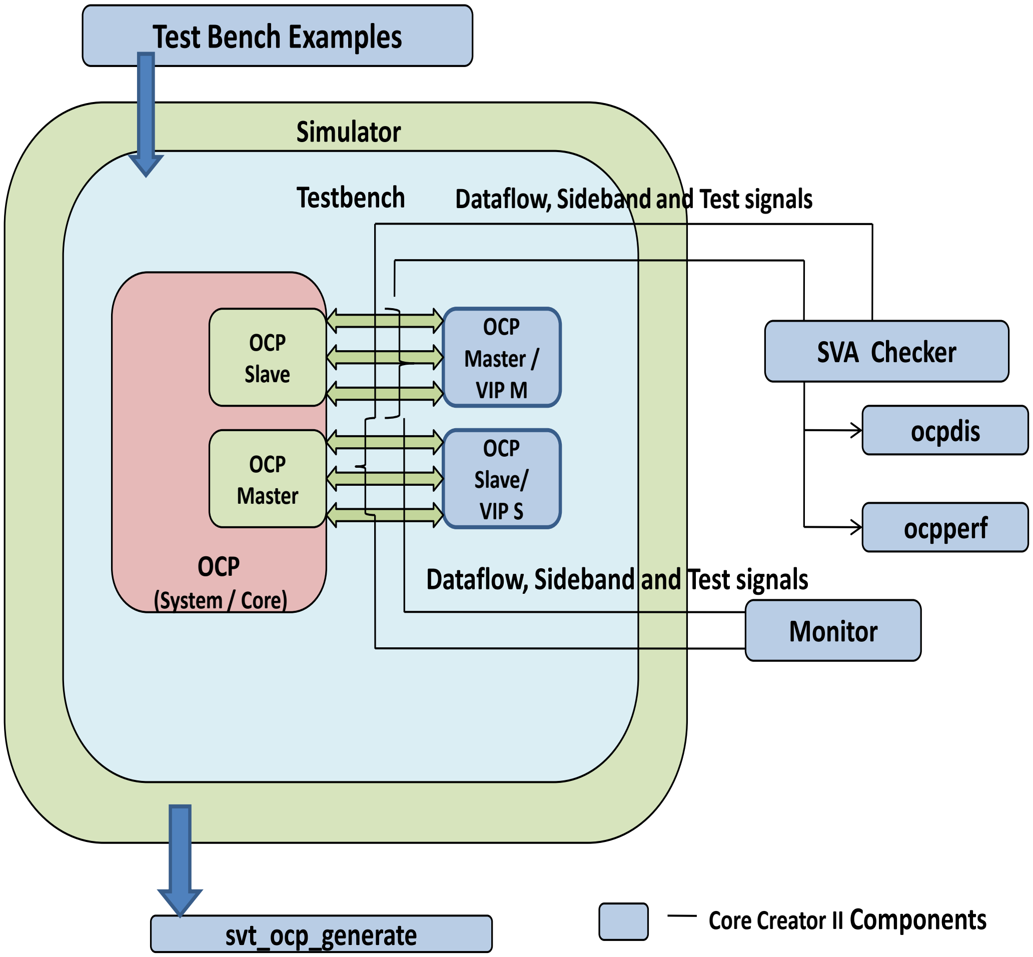


Figure 2.9. CoreCreator tool for Validating OCP Compliant IP Cores

Once different design interfaces are built, we can test the clocked architecture design implementation with the core creator tool for its performance to verify OCP compliance. Using the CoreCreator tool, we can measure the master/slave core metrics like issue rate (throughput), maximum number of operations outstanding (pipelining support), and the effect of burst support on issue rate and unloaded latency for each operation (only for slave core). Future work will translate the OCP compliant clocked design into elastic and asynchronous implementations. If this translation modifies the timing protocol, but remains faithful to the OCP protocol behavior, then we assume these versions will retain their OCP compliance.

CHAPTER 3

DESIGN DESCRIPTION AND SPECIFICATIONS

3.1 Design Description

OCP is one of the viable core-centric solutions to address contemporary SoC design implementation requirements. Commercially existing native OCP interfaces (integrated into IP Cores) are fully synchronous in nature which limits an IP core's capabilities to interface with other clocking control methodologies. In order to explore other clocking control methods, this research study designs and builds modular clocked wrapper OCP interfaces (OCP located outside the IP core) for existing IP cores. Figure 3.1 illustrates the project design structure.

This research develops a new approach to increase modularity, improve reliability, and reduce design time to interface different IPs to the OCP socket. This consists of splitting the design into common shared components and custom back-ends that are specific to the IP core. The common components consist of OCP master and slave components and a domain interface (DI) module. The DI module is used to synchronize mutually asynchronous clocked domains and dataflow control. These will be described in more detail in this chapter.

3.1.1 Design Structure

3.1.1.1 Modular Components Across Design

This project will build clocked NoC interfaces which enables multifrequency clocked designs to utilize these interfaces for standard communication. Later, as an extension to this research work, asynchronous and elastic NoC interfaces will be derived based on clocked NoC interfaces. As part of this project implementation, on a high level, it will design and implement clocked OCP interfaces between IP cores and on-chip network

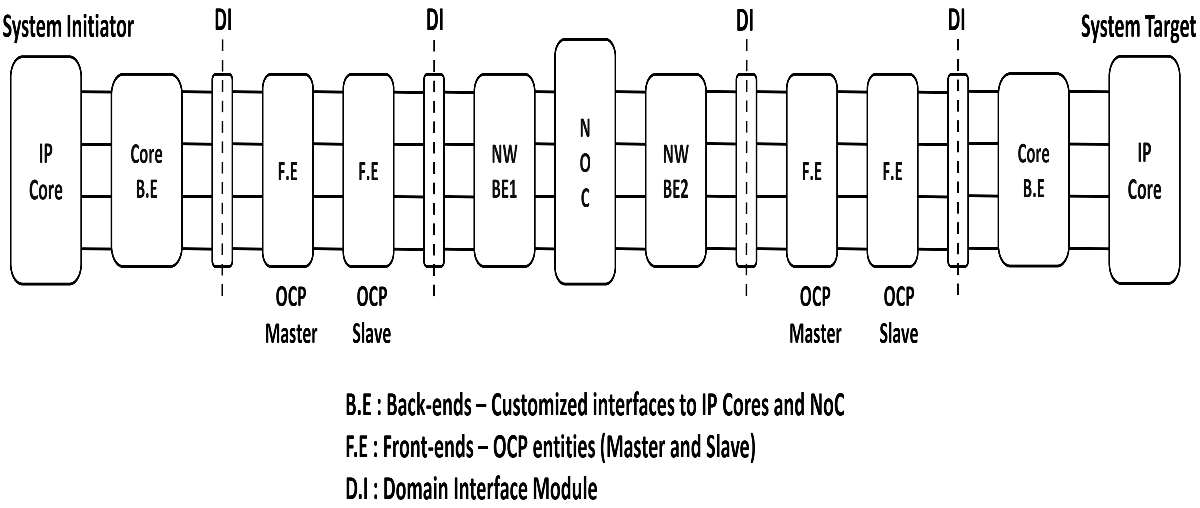

System Initiator
DI
DI
DI
DI
System Target
IP Core
Core B.E
F.E
F.E
NW BE1
N O C
NW BE2
F.E
F.E
Core B.E
IP Core
OCP Master
OCP Slave
OCP Master
OCP Slave
B.E : Back-ends – Customized interfaces to IP Cores and NoC
F.E : Front-ends – OCP entities (Master and Slave)
D.I : Domain Interface Module

Figure 3.1. Design Structure with Customized Back-ends, Front-ends and DI Module

fabric for single- and multifrequency clocked domains. Figure 3.1 illustrates the communication between two IP cores over an on-chip network fabric using customized back-end modules interfacing to the IP cores and NoC, common front-end (OCP entities), and domain interface modules.

In providing efficient and modular NoC interfaces, this project will build customized components which can be reused. In Figure 3.2, the dotted regions 1 and 2 represent the same design. The only difference is that the OCP master entity and DI module is communicating with an IP core back-end interface in the first dotted region, and in the second dotted region, it is communicating with a network fabric back-end interface. This is the same for dotted region 2.

The key point to note here is that a single design for the front-end modules and DI module will communicate with any back-end IP core or network fabric back-end interfaces. This improves modularity and simplifies design validation. The design and implementation includes the following:

a. Designing the customized back-ends to IP cores and network fabric.

b. Designing the OCP master and slave entities (front-end interfaces) and DI module.

c. Implementing the DI module with asynchronous FIFOs for GALS architectures.

d. Determining the proper placement of buffering and synchronizers across the design path (IP core-network fabric-IP core) to improve the design performance.

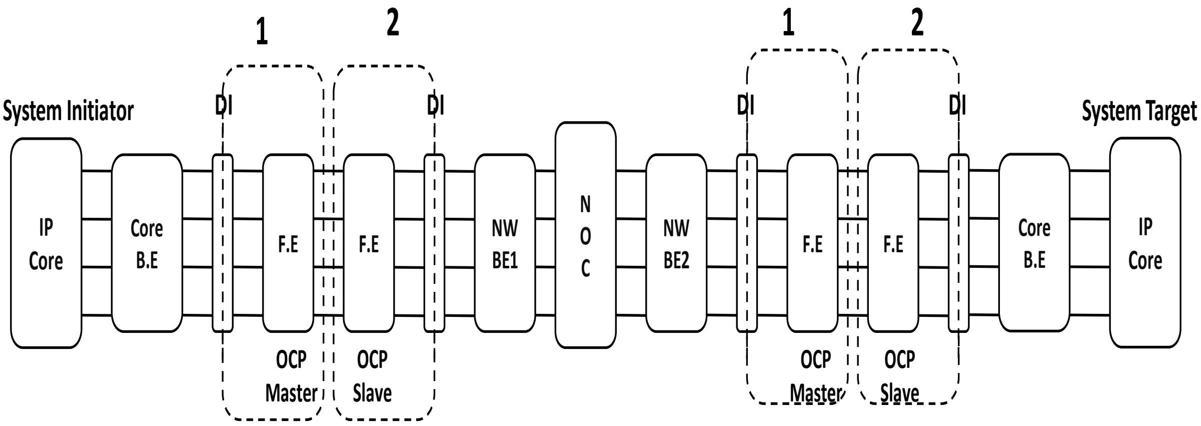


Figure 3.2. Design Structure with Modular Components Across Design Path

Also depending on the IP core architectures, the interface between OCP master and slave entities (front-end modules) can be varied based on design requirements. The design structure in Figure 3.2 enables the study of different clocking methodologies. For example, mutually asynchronous clocked IP cores can be interfaced with clocked on-chip network fabrics.

3.1.1.2 Synchronization and Buffering Placement

Communication between different architectural IP cores and on-chip network fabric in a GALS SoC requires synchronizers and buffering to mitigate metastability and uncertainty in timing [22]. Determining the proper placement of synchronizers and buffering to support multifrequency clocked domains is one of the major tasks in this study, since performance can be degraded or improved depending on the placement across the design path. Asynchronous dual clocked pointer FIFOs are employed for synchronization and dataflow control. Synchronization and buffering schemes will not be employed across OCP (master and slave) entities [23]. OCP is a point to point interface, and the data transfer between the two entities (master and slave) should use the same clock since request-acknowledgments are done with mutual consent. This also simplifies the validation of the OCP protocol. With this assumption, Figure 3.3 illustrates the possible locations where synchronizers and buffering can be employed for effective communication.

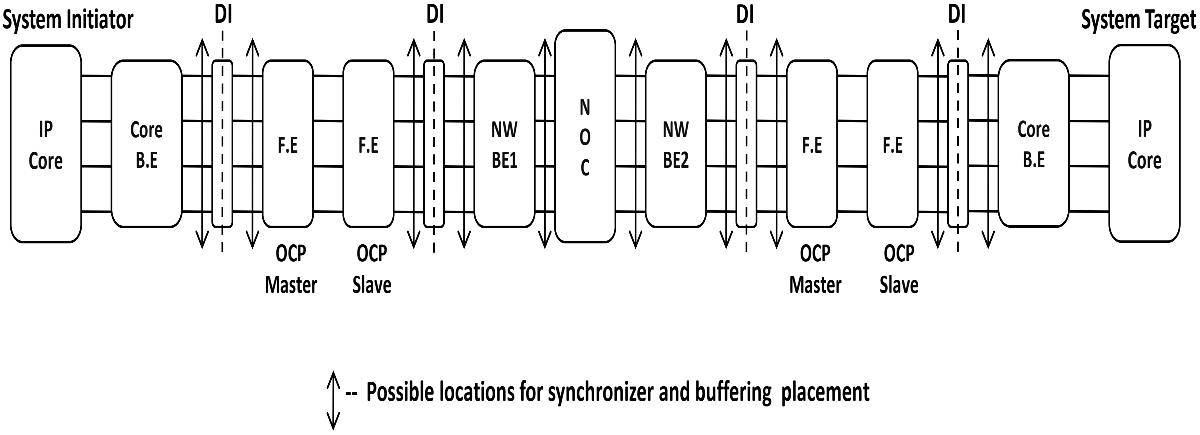


Figure 3.3. Synchronization and Buffering Placement Across Design Path

3.1.2 Network Interface Structure

Figure 3.4 shows the structural view of the network interface, consisting of front-end and back-end submodules and a domain interface in between [24]. Typically, the network front-end modules can be implemented using a standard point-to-point protocol allowing IP core resusability across multiple SoC designs, and the back-end interface to IP cores are implemented using existing protocols such as AMBA AXI, IBM CoreConnect, and OpenCores Wishbone [10].

In this project, back-end modules implement customized logic for the IP core and NoC to convert their native signals into the DI protocol. The front-end modules are implemented using OCP. A generic DI will be designed which acts as a bridge between back-end modules and OCP entities to provide buffering and flow control across the design path.

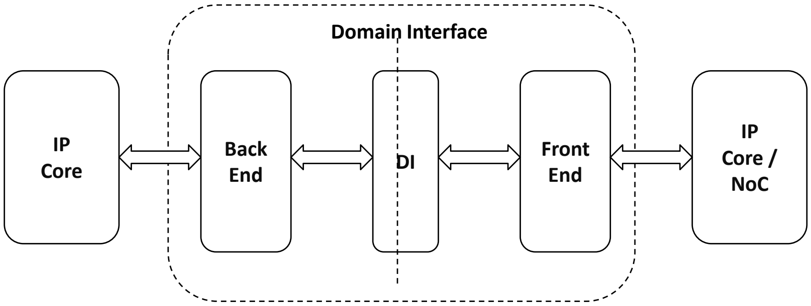


Figure 3.4. Standard Network Interface Structure

3.1.2.1 Back-end and Front-end Interfaces

The back-end submodules are custom designs for each specific IP core and NoC that interface them to the domain interface protocol, which in turn communicates with the OCP entities (master or slave). All communication occurs using a compile time configurable address, data, and control information format. Master IP core back-ends provide the functionality to generate source routing information from the IP core's logical address, which are used for sending and reception of packets to the correct destination over on-chip network fabric.

Network back-ends are customized to provide packetization and depacketization functionalities. The Slave IP core end of the network back-end stores the received source routing bits, tagging, and transaction information when a OCP protocol requires a response from the slave back to the requested Master IP core. This is used to compose the response network packet. The NoC front-end submodules are implemented using OCP and can act as either an OCP master or slave entity. The front-end modules are interfaced together to form the OCP point-to-point interface. The OCP entities are modular in nature to provide reusability across the design.

3.1.2.2 Domain Interface

The domain interface module is designed basically to synchronize the transactions across different clock regimes and control dataflow using buffering. Also, it pipelines the incoming data from IP cores and maps the existing IP cores signals and values to OCP compatible signals and values using registers (FIFOs). The same DI will be reused across the path (IP core-NoC-IP core) and its buffering capacity is defined by the IP core communication requirements and NoC bandwidth. The tradeoff of employing an intermediate DI protocol is that it may add some latency as part of the conversion between IP core signals to OCP compatible signals. This conversion usually does not add significant latency but increases design modularity. In the case of the multifrequency clocked domain, the DI module adds significant latency in request and response paths because it becomes the point of synchronization between clock domains. Synchronization also increases power dissipation, area, and reduces maximum bandwidth due to the employment of FIFOs.

3.1.3 Synchronization and Buffering

Interfacing different architectural IP cores can introduce metastability and uncertainty in timing during the transactions between two mutually asynchronous clock domains [22]. Synchronization and buggering are employed to provide safe synchronization between IP cores and the NoC. Buffering is not only used for supporting multifrequency clocked domains but also increases the throughput and reduces the overhead at peak times (traffic congestion). Determining the proper placement of synchronizers and buffering is one of the research goals in this project implementation, since performance can be degraded or improved depending on the placing across the design path (IP core-NoC-IP core).

3.1.3.1 Asynchronous Clocked Pointer FIFOs

Asynchronous FIFOs are employed to safely pass data between mutually asynchronous clocked domains. Data are written to a buffer (FIFO Memory) from one clock domain and the data are read out in another clock domain, where the two clock domains are asynchronous to each other [23] [25]. Figure 3.5 illustrates an asynchronous pointer FIFO design [23]. This design consists of FIFO memory, gray code pointers, two-flop synchronizer modules, and modules to generate stall signals into two clock domains (write full and read empty signals). FIFO memory (buffer), is accessed by both clock domains and the buffer can be instantiated using a synchronous dual-port RAM or created using a 2-D register array. Gray code pointers are used to generate write pointer, read pointer, and write_full and read_empty signals depending on the access of FIFO Memory. The two-flop synchronizer modules are used to synchronize write and read clock domains by synchronizing read pointer into the write clock domain (using write_full signal) and by synchronizing write pointer into the read clock domain (using read_empty signal).

Operation: The write pointer always points to the next word to be written on the FIFO Memory. After the FIFO write operation, the write pointer is incremented to the next location to be written. Similarly, the read pointer always points to the current FIFO address to be read, and after the read operation, it is updated to point to the next location. On reset, both the pointers are set to zero which asserts

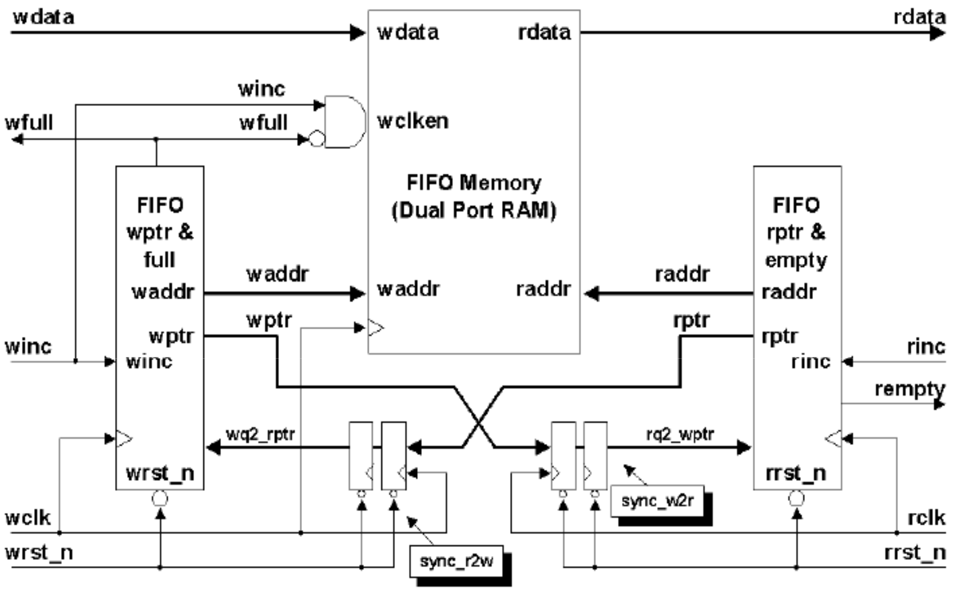


Figure 3.5. Asynchronous Dual Clocked Pointer FIFOs

read_empty signal to high, pointing the read pointer to invalid data (since the FIFO Memory is empty and the empty flag is asserted). When the first data word is written to the FIFO memory, the write pointer increments and points to next location and de-asserts the empty flag. The read pointer, which is still pointing to the first valid data location, reads out the data onto output port (RDATA). If the receiver clock is fast and if it does not require two write clock cycles for synchronization and reading out data, it can lead to both pointers pointing to same address, indicating that the FIFO is empty. A FIFO empty condition happens in case of a reset operation and when the read pointer catches up to the write pointer. A FIFO full condition (write_full signal is asserted) happens when both read and write pointers address to the same location, but in this case, the write pointer has wrapped around (faster) and caught up to the read pointer.

This design adds an extra bit to each pointer to distinguish between full and empty conditions. When the write pointer reaches the maximum FIFO address, the write pointer increments the unused most significant bit (MSB) while resetting the rest of the bits to zero, indicating that the FIFO has been wrapped once. The same occurs for the read pointer. The MSBs of two pointers will now determine if the FIFO is either full or empty.

3.1.4 Network-on-Chip Using Synchronous Routers

In this research, a NoC is employed as the communication medium between IP cores. A 3-port synchronous router designed by fellow research students is used as the NoC. Figure 3.6 illustrates the 3-port clocked elastic router architecture [19]. Each router consists of three switch and merge modules. Each switch and merge module has the buffering capability using one set latches at the input and output ports. The switch module guides the incoming data to one of the outgoing ports and the merge module arbitrates between two input requests to an output port.

Routers used in this design employ simple source routing, single-flit packets, and low latency paths [19] [26]. Each packet consists of a header containing the source routing information and the data field. Packets are switched from the input port to one of the output ports through a simple demultiplexer using the most-significant routing bit. The address bits are rotated each time and the next routing bit (MSB) controls the switching for the output packet. In this design, routers are configured at support 6 source routing bits and 66 bits of data.

3.1.4.1 Data Packets Switching Technique

Employing on-chip networks on an SoC is motivated from its novel solutions to support concurrent data transfers from the same resource, data restoration, and platform (modular components) to build reusable IPs [27]. The choice of packet switching

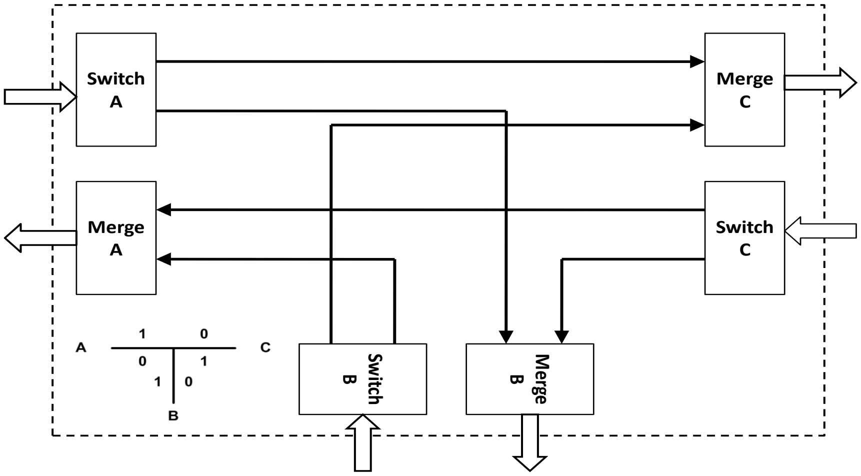


Figure 3.6. 3-Port Synchronous Router Design

technique for the on-chip network fabric is important to gain better performance. Different switching techniques are available such as store-and-forward, virtual-cut-through, and wormhole [24]. Each switching technique has different performance properties depending on the requirements and hardware resources.

Store-and-forward is used in this design to route packets from one router to the next. Store-and-forward routing analyzes packets passing through and therefore does content-aware packet routing. Normally, in this technique, big packets introduce extra packet delay at every router stage and also require a significant amount of buffer space to store multiple packets at the same time. This design employs single flit packets which requires minimal buffering space and does not introduce significant packet delay at each router.

3.1.4.2 Data Packetization and Depacketization

Packet preparation is one of the key stages of a network interface architecture since the latency associated with it can significantly impact overall communication latency [27] [24]. In this project, the network back-end modules act as wrapper logic used for packetizing and depacketizing request and response data at the NoC boundaries. At the network fabric sender end, it receives the contents from the DI module, prepares the packets, and dispatches them onto the NoC. At the network fabric receiver end, it receives the packets from the networking logic and presents the content to the DI module.

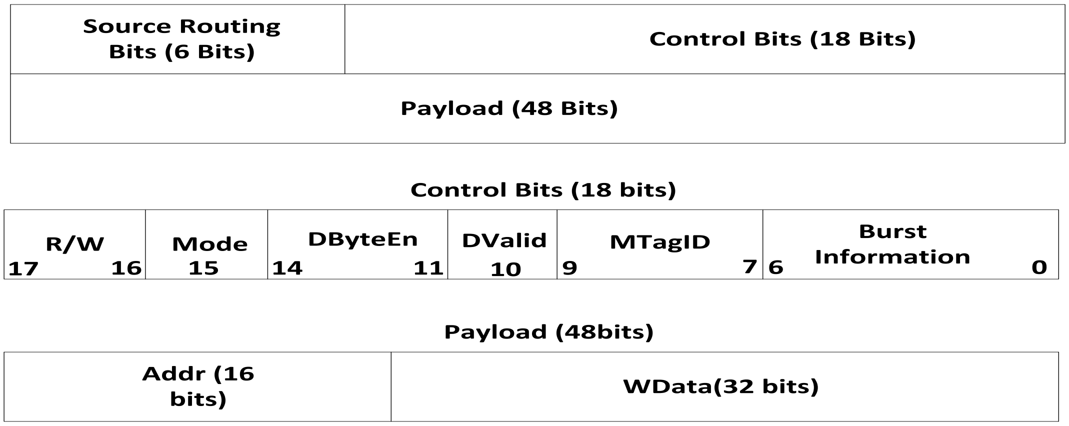
Typically, packets transported on NoCs consist of a header, control information, payload, and tail [24]. The header contains the source and destination address. Control information contains transaction type, tagging, and burst data. The tail contains error checking and correction code. In this design, packet format is customized to contain a header, control information, and payload. The header consists of source routing bits which will be used in traversing the request packets from source to destination address and also will be used for back traversing the response packets from destination to source address. At the NoC back-ends, data packetization involves constructing the request/response packets containing the source routing bits, control information, and payload from received signals from DI module. Depacketizing the data at network back-ends involves conversion of received request/response packet data into DI compatible signals. In this project design, 72-bit packets are used.

3.1.4.3 Request Packet Format

The request packet is a 72-bit packet comprised of source routing bits, control information, and payload as shown in Figure 3.7. Source routing bits are used to guide the request transaction packets to the correct destination across the on-chip network fabric and later during the response phase, the same source routing information will be used for traversing back to the source IP core. Control bits include information about transaction type, mode of operation, request tag, burst data, and write data byte enables. The payload consists of a 16-bit address and 32-bits of write transfer data.

3.1.4.4 Response Packet Format

In modularizing the design, the response packet is constructed similar to a request packet at the cost of not utilizing all the available bits. Figure 3.8 shows the response packet format. Response packets also contain source routing bits, control bits, and payload, but the difference is in the control bits and usage of the payload bits. Only 5 of the 18 control bits are used for the response type (2-bits) and response tag information (3-bits). The payload is only partially used, sending 32-bits of read data. The remainder of the bits are zero filled.



Source Routing Bits (6 Bits)	Control Bits (18 Bits)
Payload (48 Bits)	

Control Bits (18 bits)

R/W 17 16	Mode 15	DByteEn 14 11	DValid 10	MTagID 9 7	Burst Information 6 0

Payload (48bits)

Addr (16 bits)	WData(32 bits)

Figure 3.7. Request Packet Format

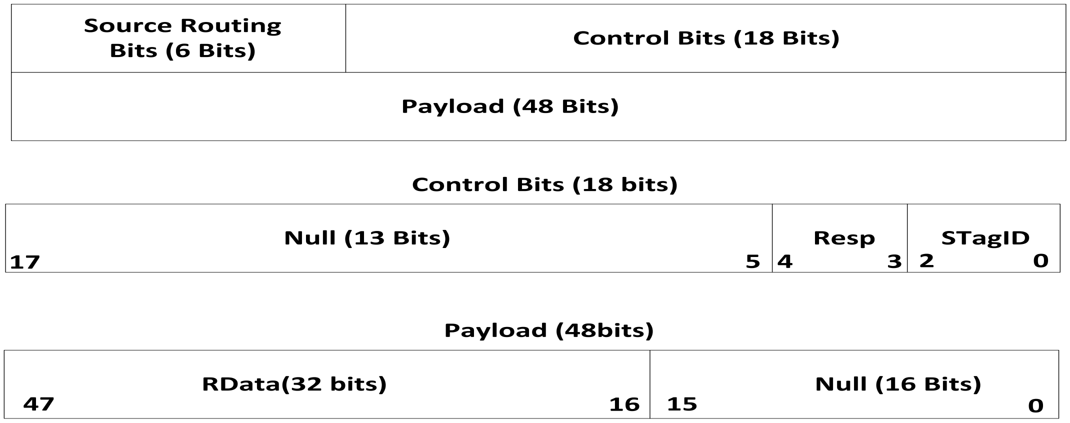


Figure 3.8. Response Packet Format

3.1.4.5 Data Flit Implementation

A typical protocol layer architecture consists of the physical layer, link layer, routing layer, transport layer, and protocol layer [28][27]. The physical layer level consists of the wiring and the transmitter and receiver hardware. The transmission of data is in the form of phits (physical data units). Each link layer is responsible to send and receive data from the physical layer in the form of flits (flow control units). The routing layer is responsible for generating the header message containing source and destination addresses. The transport layer is optional and is not used in point-to-point connections. The protocol layer is responsible for sending and receiving packets on behalf of the device. In this project design, 72-bits of single data flits are send and received across NoC. Figure 3.9 shows the single flit data format used in this project implementation.

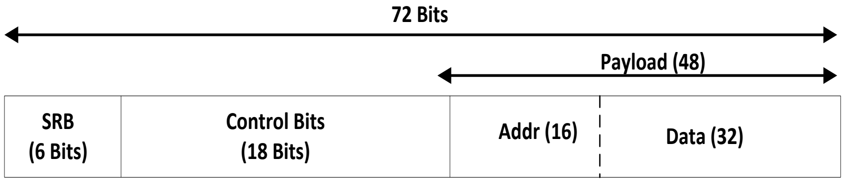


Figure 3.9. Single Flit Data Format

3.2 Design Specifications and Supporting Features

3.2.1 Proposed OCP Subset

In this project, an OCP subset is defined based on the OCP 2.2 specification to support different IP cores' communication requirements. Most of the OCP signals are configurable and can be extended depending on their requirements. Depending on the defined OCP subset, the back-ends and DI modules are customized to handle the transactions data. The OCP subset comprises basic dataflow signals, simple extension signals, burst extension, and tag extension signals. The reset signal is also supported in this implementation. The following subset of OCP is included in this implementation:

a. **Basic Signals**: only MCmd and Clk signals are required; the rest are all optional

- Clk - Clock signal
- MCmd - Transfer command (IDLE, READ and WRITE)
- SCmdAccept - Slave accepts transfer
- MAddr - Transfer address
- MData - Write data
- SDataAccept - Slave accepts write data
- MDataValid - Write data valid
- MRespAccept - Master accepts response
- SResp - Transfer response
- SData - Read data

b. **Simple Extensions**:

- MByteEn - Byte Enables
- MDataByteEn - Write Data Byte Enables

c. **Burst Extensions**:

- MBurstLength - Burst length of the current transaction
- MBurstPrecise - Whether current burst is precise or not
- MBurstSeq - Address sequence of the current burst (User Defined, INCR, WRAP)

d. **Tag Extensions**:

- MTagID - Request tag ID from Master IP.
- MTagInOrder - Current request cannot be reordered on high assertion of this signal.
- STagID - Response tag ID from Slave IP.
- STagInOrder - Current response cannot be reordered on high assertion of this signal.

e. **Sideband Signals**:

- Reset - Asynchronous reset signal.

The above defined OCP subset is implemented to support memory read, write, and idle operations, including burst transactions with out-of-order responses for single- and multifrequency clocked domains in both nonsplit and split modes. OCP supports configurable widths for address, data, bursting, and tagging bits. Table 3.1 lists the OCP subset signals and their chosen widths depending on the IP core's communication requirements. This project implements a subset of the OCP protocol; the complete OCP protocol signals are listed in the appendix.

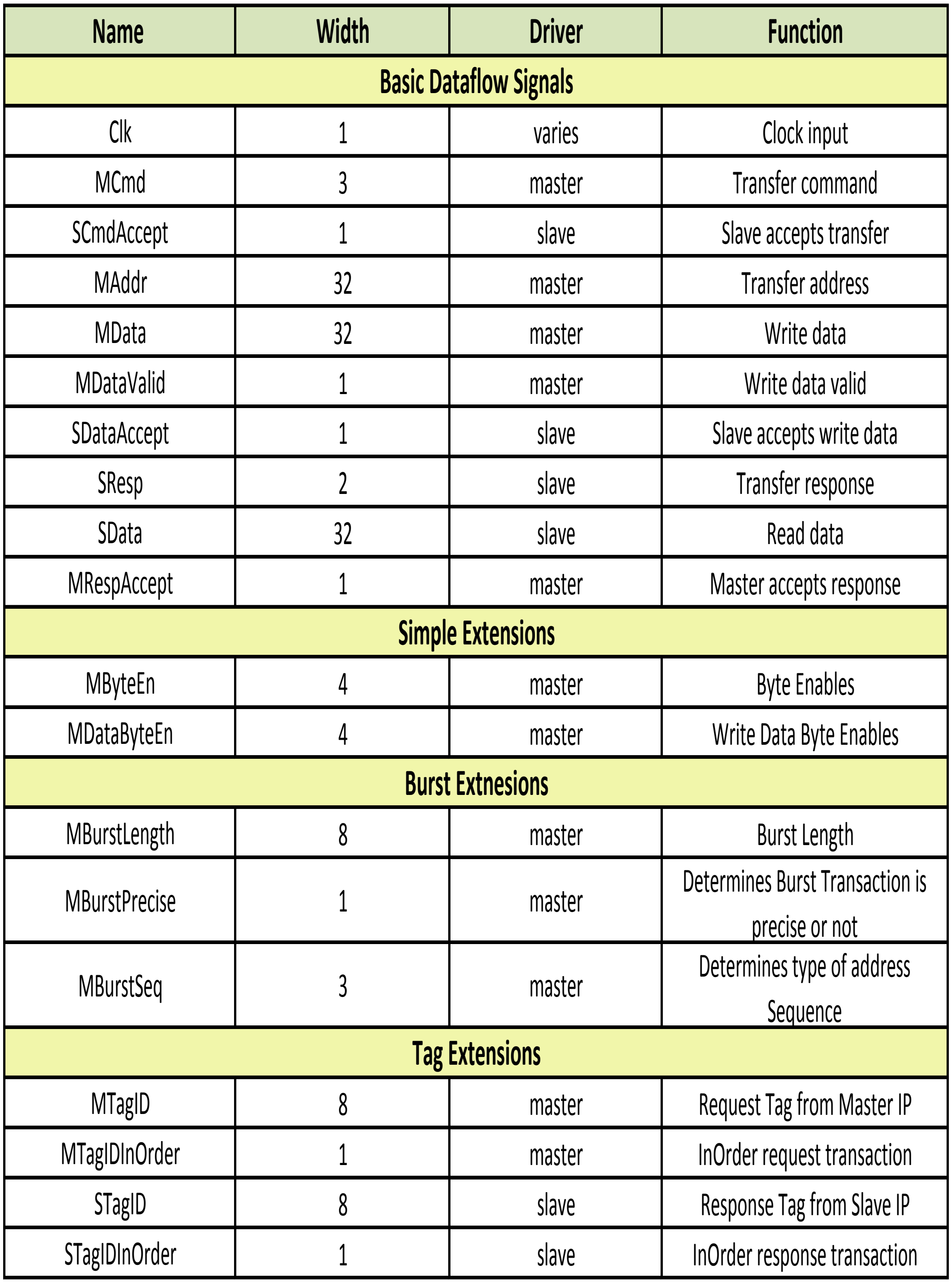
3.2.2 Supporting Features

This project implementation supports the following features.

3.2.2.1 Modes of Operation

In the *nonsplit mode* (NSP) of operation, a request is initiated and waits for a response back from the Slave IP before sending out another request. Pipelining of request transactions from the same IP core or simultaneous requests from different IPs are not possible in this mode of operation. In *split mode* (SP), pipelining of request transactions from the same IP and simultaneous instantiation of requests is possible. Due to the pipelining of requests in split mode, performance and throughput are increased and the maximum NoC bandwidth can be utilized. Split mode reduces overall latency, increases throughput, and lowers energy per transaction compared to nonsplit mode.

Table 3.1. OCP Proposed Subset



Name	Width	Driver	Function
Basic Dataflow Signals			
Clk	1	varies	Clock input
MCmd	3	master	Transfer command
SCmdAccept	1	slave	Slave accepts transfer
MAddr	32	master	Transfer address
MData	32	master	Write data
MDataValid	1	master	Write data valid
SDataAccept	1	slave	Slave accepts write data
SResp	2	slave	Transfer response
SData	32	slave	Read data
MRespAccept	1	master	Master accepts response
Simple Extensions			
MByteEn	4	master	Byte Enables
MDataByteEn	4	master	Write Data Byte Enables
Burst Extnesions			
MBurstLength	8	master	Burst Length
MBurstPrecise	1	master	Determines Burst Transaction is precise or not
MBurstSeq	3	master	Determines type of address Sequence
Tag Extensions			
MTagID	8	master	Request Tag from Master IP
MTagIDInOrder	1	master	InOrder request transaction
STagID	8	slave	Response Tag from Slave IP
STagIDInOrder	1	slave	InOrder response transaction

3.2.2.2 Burst Transactions

A burst transaction is a set of transfers that are linked together, which have a predefined address relation and number of transfers. Compared to single data phase transactions, burst transactions improve data throughput since the address is transferred only during initial bus grant followed by chunks of data. Burst transactions are supported not only to increase the throughput but also to reduce latency and data activity factor across the network router links.

Different burst implementations are possible depending on the burst length (size of the burst) and burst sequence (relation between the addresses). This project design supports a maximum burst size of 8 (8* word size) i.e., 256-bits of data. This project implementation supports three types of address sequences (user defined, incremental, and wrapped types), and data can be transferred during each clock cycle.

3.2.2.3 Tagging or Out-of-Order Response

Basically, tags are used to support out-of-order response by directly linking the Slave IP core response to the original request which triggered it. By supporting out-of-order responses, in most cases, the use of tags can improve overall system performance since responses are not halted due to dependencies on previous transactions. In this project implementation, a tag size of 8 is employed.

CHAPTER 4

DESIGN IMPLEMENTATION

4.1 Clocked OCP Design Implementation

The project design implementation includes building of standard OCP interfaces to IP cores and NoC in single- and multifrequency clocked domains. Figure 4.1 shows a clocked OCP design implementation with a signal level description from Master to Slave IP cores. Both designs include customized back-end interfaces to IP cores and NoC, front-ends (OCP entities), and DI module. A standard ASIC design flow is used in developing the designs from RTL code to physical placement and routing. The developed modules are functionally tested at each phase of the implementation. An emulated RISC pipelined CPU and synchronous memory (RAM/ROM) are employed as Master IP (system initiator) and Slave IP (system target), respectively, and customized synchronous routers are used as NoC for testing the functionality. Data transfer transactions include simple memory read, write, and idle operations (nonsplit transactions) through pipelining requests (split transactions) with read /write bursts to complex out-of-order operations (tagging).

The following steps are executed in building OCP interfaces between IP cores and the NoC in both single- and multifrequency clocked domains.

a. Single-Frequency Clocked Domain Implementation (SFCD):
   i. Customized back-end interfaces to IP-cores and NoC
   ii. A generic DI module
   iii. OCP Master and Slave entities (Front-ends)
   iv. Integrating built modules with on-chip network fabric

b. Multifrequency Clocked Domain (MFCD) Implementation: This implementation utilizes the developed back-ends and front-ends modules from single-frequency

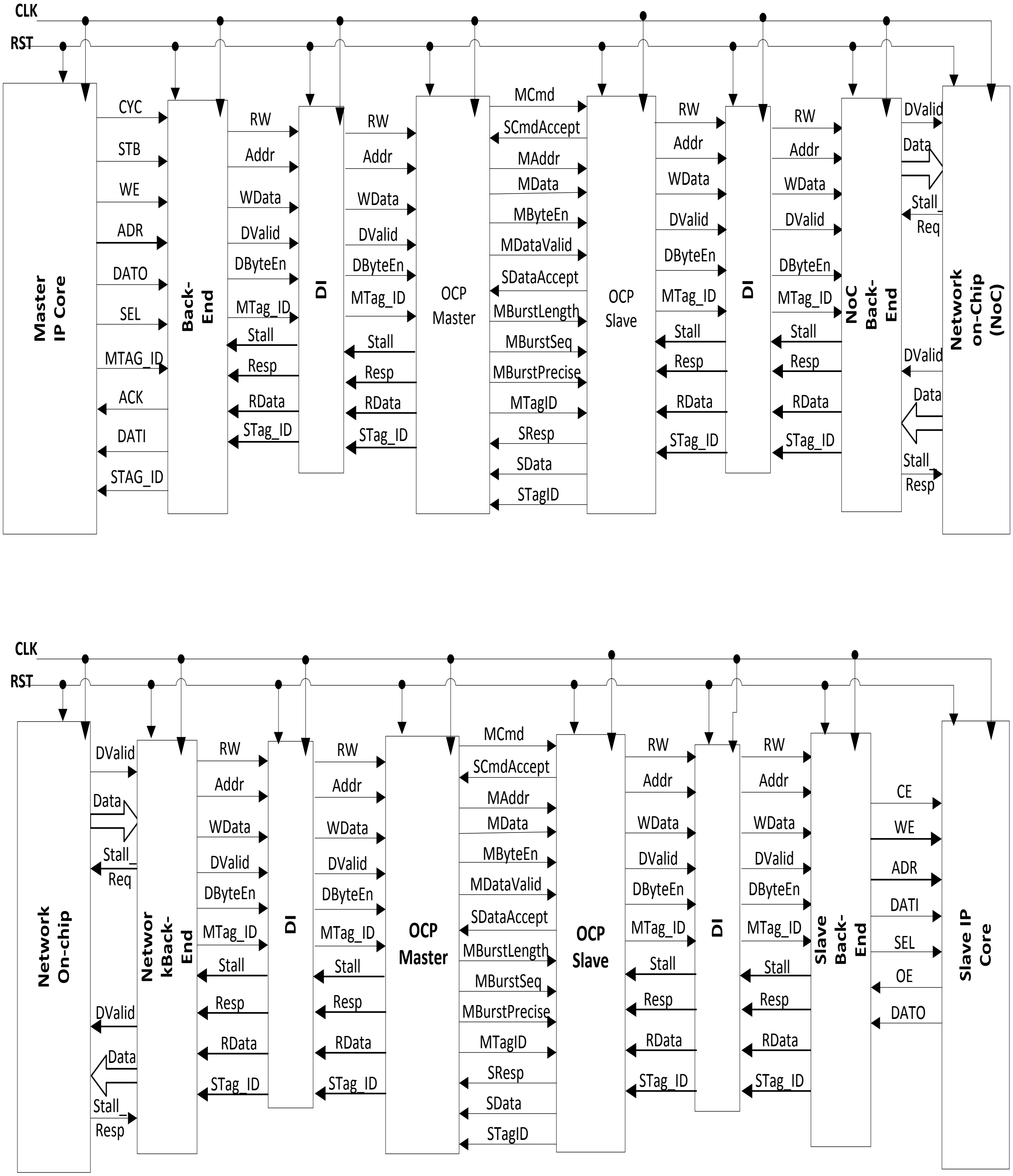


Figure 4.1. Clocked OCP Design Implementation with IP Cores and NoC

clocked domain. The DI module design and implementation with FIFOs is pivotal in this implementation.

i. Customize and build asynchronous dual clocked pointer FIFOs

ii. A generic DI module with asynchronous FIFOs

iii. Determining proper placement of buffering and synchronizing schemes at
   - IP core and network fabric boundaries
   - Interface between back-end modules and DI module
   - Interface between front-end modules (OCP entities) and DI module

iv. Integrating built modules with on-chip network fabric

4.1.1 Single-Frequency Clocked Domain Implementation

Figure 4.2 illustrates a single-frequency clocked domain (SFCD) design with NoC interfaces over a single IP Core-NoC-IP Core path. The following steps list the data flow transaction steps across the design in a single-frequency clocked domain.

a. A global clock and asynchronous reset is employed to the IP cores, On-Chip network fabric and NoC interfaces.

b. Communication between all modules is triggered on the positive edge of the clock cycle.

c. On assertion of the asynchronous reset signal high, all the current transaction data are cleared.

d. During request and response phases, transaction data are traversed and guided from one module to the next module in a pipelined stage fashion as described below.
   - Master IP core required communication signals are mapped to the customized master back-end signals.
   - Master back-end transfers the customized signals onto DI stage and pipelined at OCP master entity.
   - OCP master entity maps the customized signals into OCP protocol compatible signals and transfers them onto the OCP slave entity.
   - The OCP slave entity remaps the OCP protocol signals into DI module signals which are pipelined at the on-chip network fabric back-end.

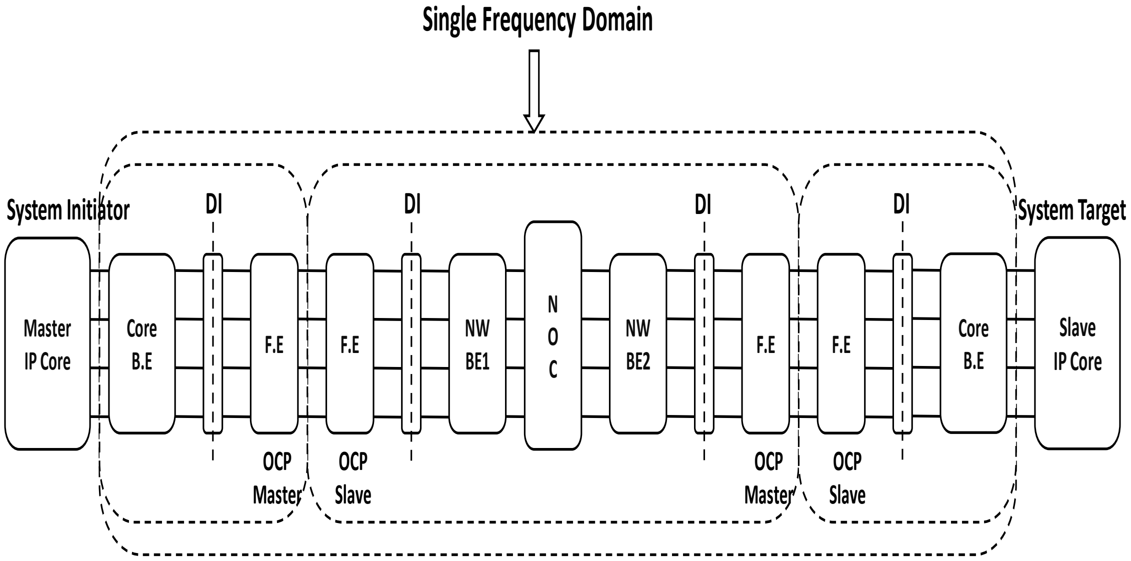


Figure 4.2. Single-Frequency Clocked Domain (SFCD) Implementation with Modular Components Across Design

- The network back-end interface packetizes the data in the required packet structure format and presents it to the on-chip network fabric.
- Data flits are traversed across the routers using source routing information and presented at the other network fabric back-end.
- The other network fabric back-end interface depacketizes the data and the above process is followed until the data are mapped to required Slave IP cores existing signals.

e. Stall signal generated from any successive module is traversed back to source IP generating requests.

f. In nonsplit operation mode, the IP core master back-end module blocks the channel until the reception of a response for the nonsplit transaction.

g. In split mode, simultaneous or pipelining of requests are possible and inclusion of tagging supports out-of-order responses from Slave IP.

h. Burst read/write transactions are supported with passage of burst information (burst length and address sequence) from Master IP to Slave IP.

4.1.1.1 IP Cores Back-end Modules

Figure 4.2 shows the customized back-end interfaces to IP cores (Master Back-end and Slave Back-end). IP core back-ends are customized to interface standard point-to-point protocols like OCP. The IP core back-end modules are primarily responsible for mapping legacy IP core signals to a defined set of DI module signals and vice-versa. Mapping of signals between back-ends and DI module include request, response, tagging information, and stall signals.

The master back-end provides the functionality to determine data validity of each initiated request by validating request transaction control signals from IP cores and stall signals from the DI module. The master back-end is also responsible for generating source routing bits from the received logical address using customized memory mapped registers and inserting them as part of the address which traverses to the NoC back-end. In this implementation, the master back-end is distinguished from the slave back-end in the handling of operation modes (nonsplit and split) depending on the received commands from a Master IP. In nonsplit mode, it does not allow the IP core to initiate another transaction or pipeline any more requests before the reception of response to previous nonsplit transaction.

A slave back-end handles the functionality to enable the control signals (Chip Enable, Read/Write Enable) of a Slave IP core (Synchronous memory) for the current transaction depending on the valid requests received from DI module. It is also responsible for communicating burst, tagging information, and for synchronizing the response data from Slave IP to DI module.

4.1.1.2 Domain Interface (DI) Module

The domain interface module serves as a bridge between IP cores/NoC back-ends and front-ends (OCP entities) following standard network interface structure. The DI module handles the request and response data transfer flow between back-ends and front-ends. In this implementation, the need for asynchronous FIFOs to provide synchronization and buffering is not required with a global clock employed to all modules. The synthesized and physically routed DI module introduces combinational logic delay on the output ports.

4.1.1.3 On-Chip Network Fabric Back-end Modules

Each network fabric back-end module is responsible for data packetization and depacketization functionalities with a single flit implementation methodology onto the NoC using a store-and-forward packet switching technique. Figure 4.3 shows on-chip network back-ends interfacing the NoC. Packetizing of the data includes assembling source routing bits, control information, and payload into a single required transaction format packet (flit). Depacketizing the data involves retrieving the required control and data signals of the DI module. Each data flit containing 72-bits of information is transmitted and received to/from the NoC with a DataValid (DValid) signal asserted high during the transaction indicating valid data on the channel. For transactions where a stall signal is asserted high or DataValid signal asserted low during dataflits transmission/reception, the current dataflits are ignored as invalid data at the NoC and network back-ends.

The Slave IP end of the network fabric back-end is customized to store the source routing information in memory arrays for sending the response packets back to the correct destination address. Initially, during the request transactions, the network back-end stores the required routing, tag bits, transaction type, and burst information bits in 2-dimensional register arrays. On reception of a response from a Slave IP, the tag bits are compared with the stored information and the correct destination address routing bits are retrieved. Once the destination routing bits are retrieved for the current response transaction, the allocated memory is cleared for storing next request

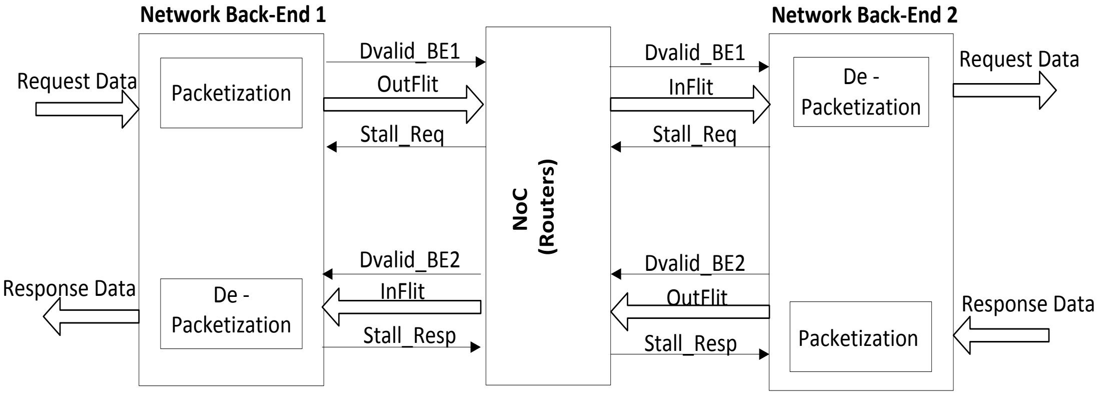


Figure 4.3. On-Chip Network Back-ends Interfacing NoC

transaction routing bits. In case of burst request transactions, the allocated memory is cleared only after reception of the required number of burst responses from the Slave IP. In this design, 2-dimensional memory arrays each with 8 words of capacity are used to maintain the traffic.

4.1.1.4 OCP Master and Slave Entity Modules

OCP is a point-to-point synchronous interface with complimentary master and slave entities communicating in a peer-to-peer fashion [11]. Each OCP entity can only communicate with its complementary side of OCP. The OCP master and slave entities are modularized for reusability across the design. Depending on the requirements of a system, an IP module can have only OCP Master or OCP Slave or both for communicating with the external world. Figure 4.4 illustrates the transaction phases between OCP entities.

During the request phase, each transfer across the complementary entities starts with a request initiated from the OCP master presenting a transfer request command (MCmd), address, tag bits (MTagID), and other control information to the OCP slave. Depending on the Slave IP or bus availability to accept the request, the OCP slave entity acknowledges to the OCP master with a slave accept transfer signal (SCmdAccept). If the slave is ready to accept the request, it asserts the SCmdAccept signal high and the OCP master sends out the current transaction data in the Datahandshake phase on the next cycle. The OCP slave, after receiving the current transaction data, acknowledges the OCP master by asserting SDataAccept

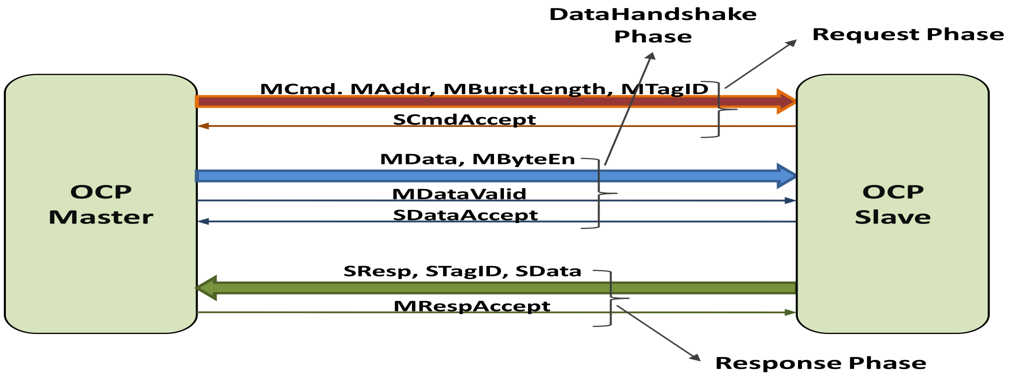


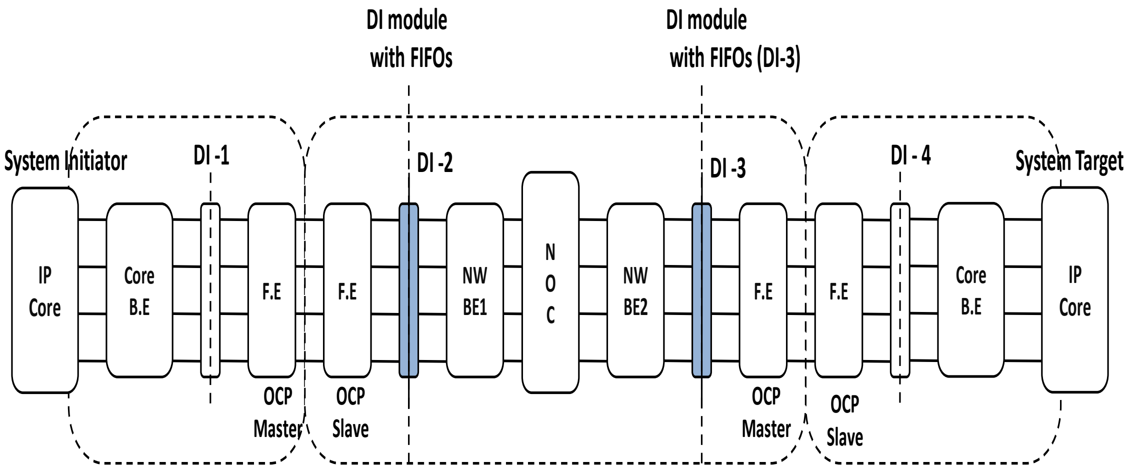
Figure 4.4. Transaction Phases between OCP Master and OCP Slave Entities

signal high and maps the request data into the DI module compatible signals. If the slave is busy, then the OCP slave de-asserts the SCmdAccept signal low which makes the OCP master assert and maintain the stall signal high back to DI module high until the OCP slave is ready to accept the next request. During the response phase, the OCP slave presents the response data from the Slave IP/NoC for the requested transaction with a response signal (SResp), response tag bits (STagID), and read data (SData) signal. On reception of a response, the OCP master acknowledges the OCP slave by asserting MRespAccept signal high. This process continues. The datahandshake and response phases are optional between OCP entities.

4.1.2 Multifrequency Clocked Domain Implementation

Figure 4.5 illustrates the design of a multifrequency clocked domain (MFCD). The back-ends and front-ends from the single-frequency clocked domain are reused in a multifrequency clocked domain. In a multifrequency clocked domain, the IP cores and NoC can be operated at different clock frequencies, resulting in mutually asynchronous clock domains. In order to avoid metastability and uncertainty in timing when interfacing two mutually asynchronous clock domains, synchronization and buffering is implemented in the DI module. The DI module employs asynchronous dual clocked pointer FIFOs in the request and response paths to provide synchronization, buffering, and dataflow control. In this domain, the same dataflow transaction steps as the single-frequency clocked domain are implemented, with the exception of added synchronization and buffering in the DI module.

FIFO placement across the design is one of the critical steps in this project implementation since performance can be degraded or improved depending on the placement across the design path (IP core-NoC-IP core). Synchronization can be employed in any of the DI modules while crossing from one clock domain to another clock domain. Note that it is preferable to have the same clock domain at the IP core back-ends and front-ends (OCP entities), since this reduces the overhead of synchronizers and buffering. In Figure 4.5, the Master IP, Slave IP, and NoC are in mutually asynchronous clock domains. In this scenario ,it is always advisable to employ synchronization and buffering in the DI modules closer to on-chip network



Front-ends (OCP entities) operating on IP Cores clock frequency
NoC and network B.E are operating on NoC clock
D.I module with FIFOs has dual clocks (for writing into and reading out transactions simultaneously in the request and response path).

Figure 4.5. Multifrequency Clocked Domain (MFCD) Implementation Using Asynchronous Clocked FIFOs

fabric back-ends. Employing asynchronous FIFOs only in the DI modules closer to network fabric back-ends not only segregates the NoC from IP cores but also provides an efficient way to reduce the overhead of synchronization and buffering on IP core back-ends and front-ends.

4.1.2.1 DI Module with FIFOs

Figure 4.6 shows a DI module with integrated asynchronous FIFOs. In Figure 4.5, DI modules DI-2 and DI-3 have integrated FIFOs. Module DI-2 has two mutually asynchronous clock signals as inputs, the write clock (WCLK) and read clock (RCLK). In the request path, the request transaction data from the OCP slave entity are written into DI module (FIFO memory) using WCLK and the network fabric back-end reads out the data from DI module (FIFO memory) using the read clock (RCLK). Similarly, in the design, module DI-3 uses the write clock to write request transaction data from the network fabric back-end and OCP master reads out request data from DI module (FIFO memory) using read clock (RCLK). This process continues. In the response path, FIFOs are employed using the same mechanism as the request path.

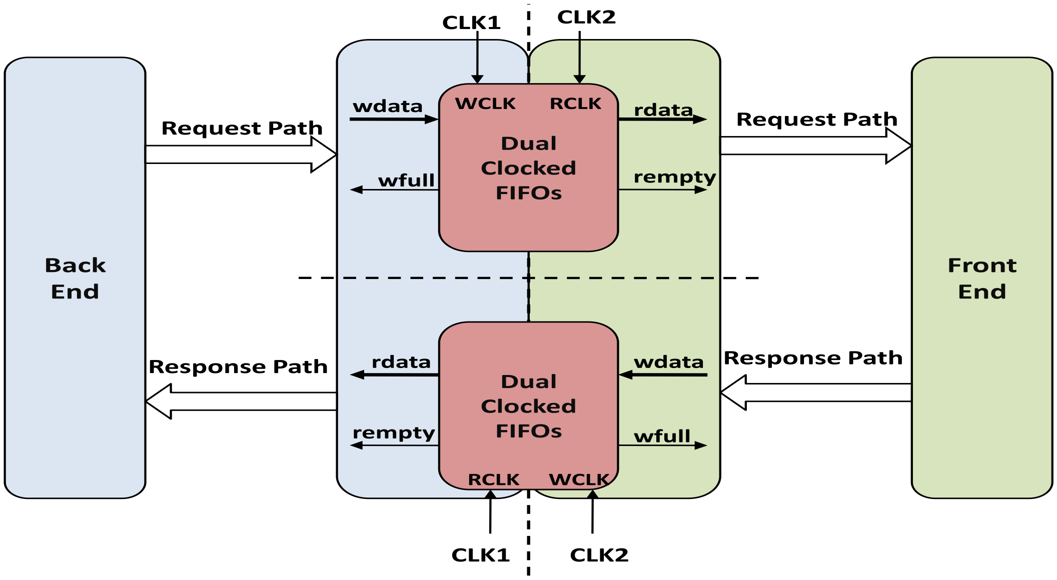


Figure 4.6. DI Module with Asynchronous Clocked FIFOs

Whenever the two pointers (write and read) catch up with each other, or on reset, an additional two clock cycles are required to synchronize signals between the clock domains. In the case where the write pointer catches up with the read pointer, an extra two write clock cycles are required to synchronize the read pointer value into write clock domain to guarantee a vacant slot for writing into the FIFO memory. Similarly, when the read pointer catches up with the write pointer, an extra two read cycles are required to synchronize the write pointer into read clock domain to guarantee valid data are read out from the FIFO memory. A synchronized write pointer is used to generate rptr_empty (read empty) signal and a synchronized read pointer is used to generate wptr_full (write full) signal.

Writing request transaction data into a FIFO is stalled or halted on the assertion of the wptr_full signal high (occurs when write pointer catches up with read pointer). When the DI module stall signal goes high on assertion of wptr_full signal high, the OCP slave is not allowed to accept any more request transactions from its OCP Master entity. A rptr_empty signal is asserted high (occurs when read pointer catches up with write pointer), when there are no more data to be read out. This design implementation uses 8-words of memory for data storage in the request and response path of DI module which are employing asynchronous FIFOs.

4.2 Design Implementation Flow

The standard ASIC design flow is used to build clocked interfaces in single- and multifrequency clocked domains. Figure 4.7 illustrates the development flow and required implementation files. In the implementation flow, each design phase is functionally verified with integrated IP cores and NoC. During logic synthesis and automatic place & route phases, the synthesized designs are tested with sdf back annotation for meeting setup and hold time requirements. In this project implementation flow, the Modelsim simulator is used for testing the design functionality.

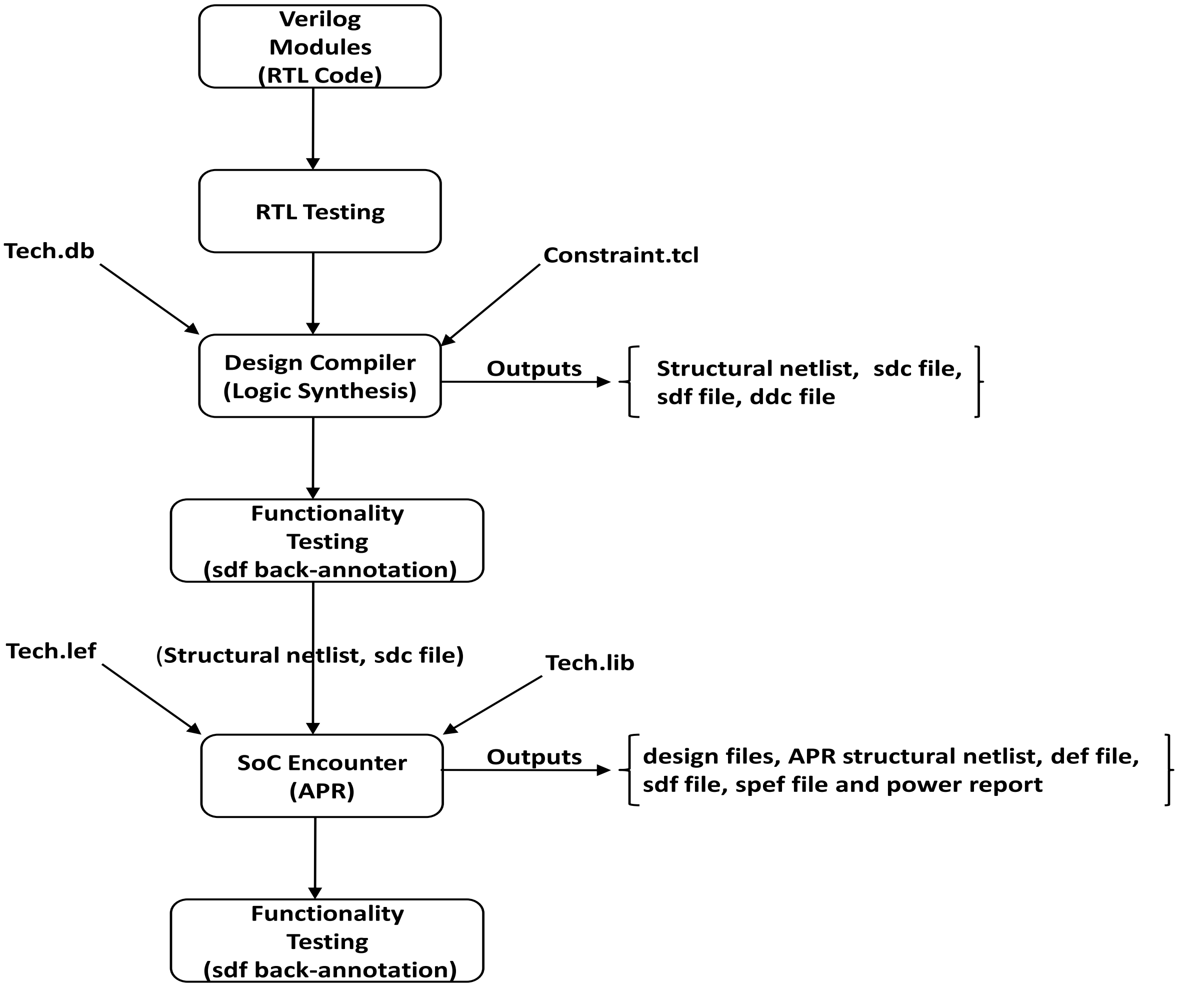

Verilog Modules (RTL Code)
RTL Testing
Tech.db
Constraint.tcl
Design Compiler (Logic Synthesis)
Outputs
Structural netlist, sdc file, sdf file, ddc file
Functionality Testing (sdf back-annotation)
Tech.lef
(Structural netlist, sdc file)
Tech.lib
SoC Encounter (APR)
Outputs
design files, APR structural netlist, def file, sdf file, spef file and power report
Functionality Testing (sdf back-annotation)

Figure 4.7. Design Implementation Flow with Technology Library Files

4.2.1 RTL Source Codes Using Verilog HDL

In this project implementation, RTL codes are developed using Verilog HDL. The project design is customized to construct modular components across the design. The project implementation includes macros to define the widths of address, data, and other control signals. The widths are configurable by modifying the defined macros widths and corresponding changes are required in back-ends. Figure 4.8 illustrates the RTL structure of this project design. Following are the Verilog modules coded in this project implementation

a. **Test Bench**:

```
timescale.v  : Timescale definition. Included in all files
defines.v    : Macro module defines configurable signal widths.
               Included in all files.
test.v       : Test Bench
top.v        : Design Top Layer
```

b. **Single-Frequency Clocked Domain**:

```
master_be.v  : Customized back-end to Master IP core.
di.v         : Domain Interface module (only with combinational logic).
di_ocpm.v    : OCP Master entity interfacing with DI module.
ocps_di.v    : OCP Slave entity interfacing with DI module.
nw_be1.v     : Customized network back-end (from Master IP end) to NoC.
nw_be2.v     : Customized network back-end (from Slave IP end) to NoC.
slave_be.v   : Customized back-end to Slave IP core.
```

c. **Multifrequency Clocked Domain**:

The developed modules in SFCD are used in MFCD. In this implementation, the DI module is constructed with asynchronous FIFOs to handle synchronization and buffering. The FIFOs are developed by customizing the dual clocked pointer FIFOs proposed in *Simulation and Synthesis Techniques for Asynchronous FIFO Design* papers [23] [25]. The developed FIFOs are used in both the request and response paths. Following are the Verilog modules.

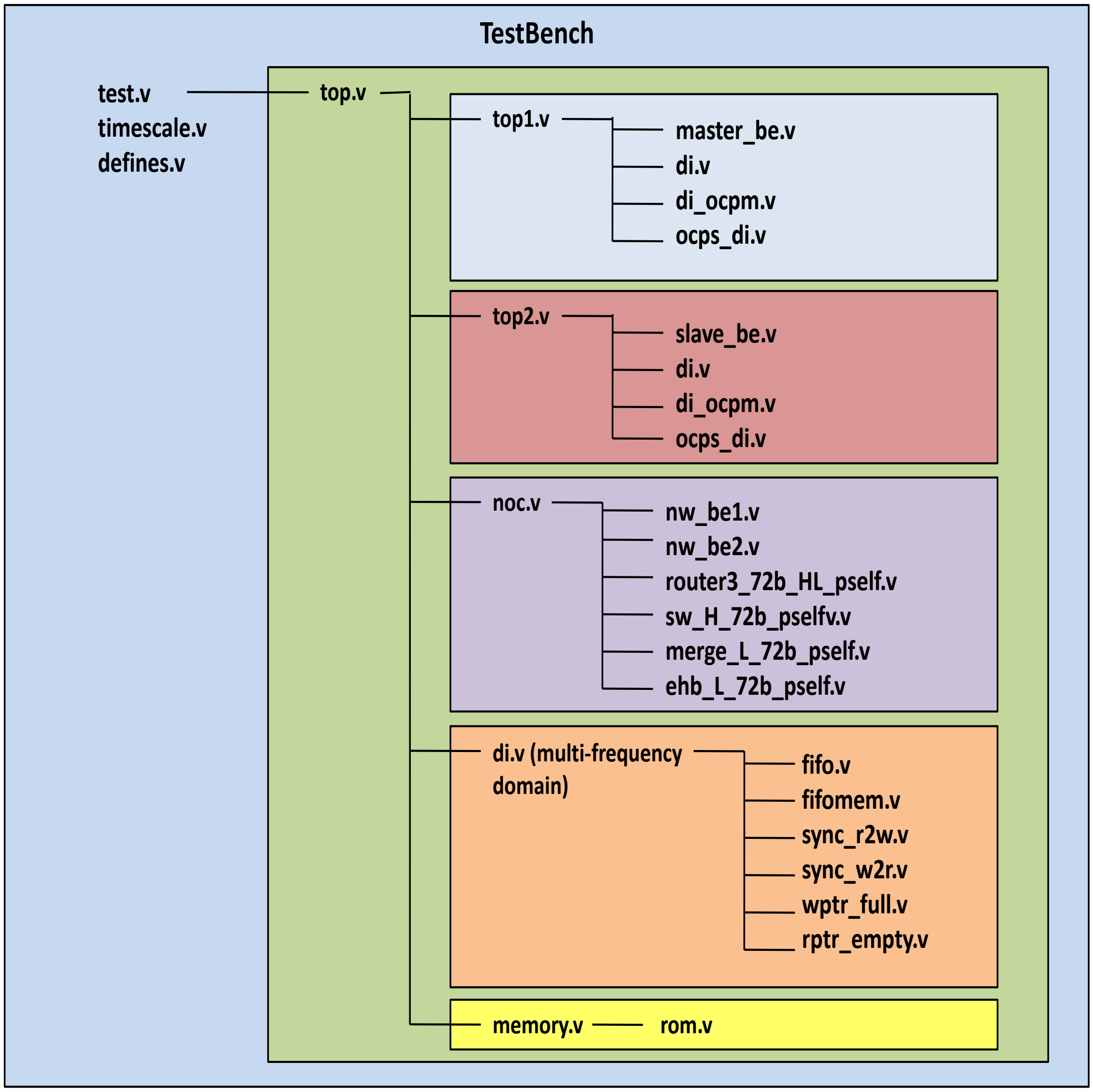


Figure 4.8. Design RTL Structure

master_be.v : Customized Back-end to Master IP core.

di.v : Domain Interface module (with asynchronous clocked FIFOs).

di_ocpm.v : OCP Master entity interfacing with DI module.

ocps_di.v : OCP Slave entity interfacing with DI module.

nw_be1.v : Customized Network Back-end (from Master IP end) to NoC.

nw_be2.v : Customized Network Back-end (from Slave IP end) to NoC.

slave_be.v : Customized back-end to Slave IP core.

fifo.v : Top level FIFOs module.

```
fifomem.v     : FIFO memory module (instantiated with 8 words of data)
sync_r2w.v    : Module to synchronize read pointer into write clock domain.
sync_w2r.v    : Module to synchronize write pointer into read clock domain.
wptr_full.v   : Module to generate write pointer full signal.
rptr_empty.v  : Module to generate read pointer empty signal.
```

d. **Slave IP**:

```
memory.v      : ROM (8KB) and RAM (8KB) for testing verilog codes.
rom.v         : Description created by a converter from S-format.
```

e. **On-Chip Network Fabric or NoC**:

```
router3_72b_HL_pself.v : 3-Port Router top-level module.
sw_H_72b_pselfv.v      : Router Switch module.
merge_L_72b_pself.v    : Router Merge module.
ehb_L_72b_pself.v      : Elastic Half Buffer module.
```

4.2.2 Design Synthesis Using Synopsys Design Compiler

The design is segregated into three blocks corresponding to the IP cores and NoC to explore their performance characteristics individually, as shown in Figure 4.9. The Top1 and Top2 modules include customized back-ends to IP cores, OCP entities, and DI module. The NoC module includes the routers and on-chip network fabric back-ends. The project design is synthesized using Synopsys Design Compiler with Arm's Artisan static cell library using IBM's 65nm process technology. The University of Utah tcl scripts for logic synthesis and auto place & route are modified to accomplish design synthesis. The generated structural Verilog netlists are integrated into one unit by manually wiring and functionally tested with sdf back-annotation.

4.2.3 Design APR Using Cadence SoC Encounter

Cadence SoC encounter is used for automatic place and route of this project design. The individual top level synthesized structural Verilog netlists (Top1, Top2, and NoC) and with other corresponding required files, as shown in Figure 4.7, are

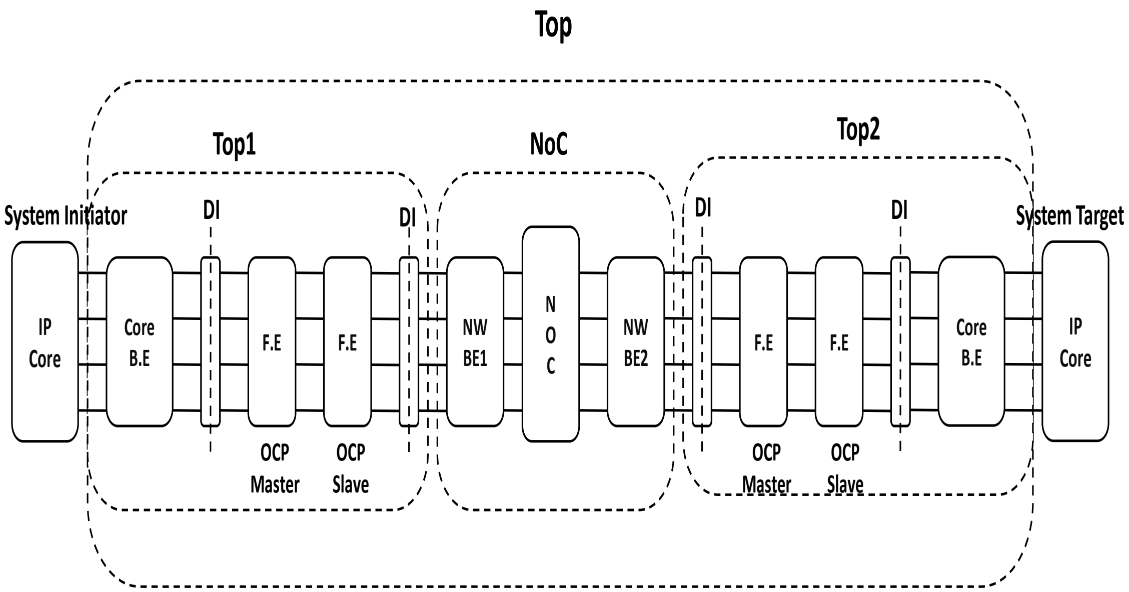


Figure 4.9. Design Synthesis Structure

imported into SoC encounter for auto place and route. The physically routed modules are integrated and functionally tested with sdf back-annotation. After achieving the expected functionality on the final routed design with setup and hold time requirements, the performance metrics are captured.

CHAPTER 5

VALIDATION OF CLOCKED OCP COMPLIANT INTERFACES

Increasing design complexity, cost, and probability of errors on an SoC has raised the importance of verification to reduce design time and risk, ensuring rapid time-to-market. Ideally, in a standard ASIC design flow, a certain degree of verification is required at every stage to improve product quality. In industry, verification is more often performed only at critical design implementation stages (RTL, logic synthesis, and physical implementation).

In this research, the Sonics OCP Library for Verification (SOLV) package is used to validate the clocked interface designs for OCP compliance [29]. The SOLV package supports compliance for all the released OCP-IP *Open Core Protocol* specifications. The SOLV package comprises three components: an OCP checker, a dissembler, and a performance analyzer. Figure 5.1 shows the SOLV components and tool flow. Basically, the SOLV component provides a system Verilog assertion (SVA) based checker which can be integrated into a Verilog testbench to validate the protocol compliance. The checker captures OCP interface signals during simulation on each OCP clock cycle and compares them to the OCP protocol requirements. An assertion-based property verification mechanism is employed to check signals and report protocol violations at the same clock cycle at which the assertion is failed.

The Sonics OCP checker dynamically validates OCP interfaces during simulation, and generates OCP trace files for use by the postprocessing tools ocpdis (OCP disassembler) and ocpperf (OCP performance analyzer). During simulation, the OCP connection activity is logged into an OCP trace files, consisting of hexadecimal valued tables. An OCP dissembler uses the OCP trace files to display OCP connection

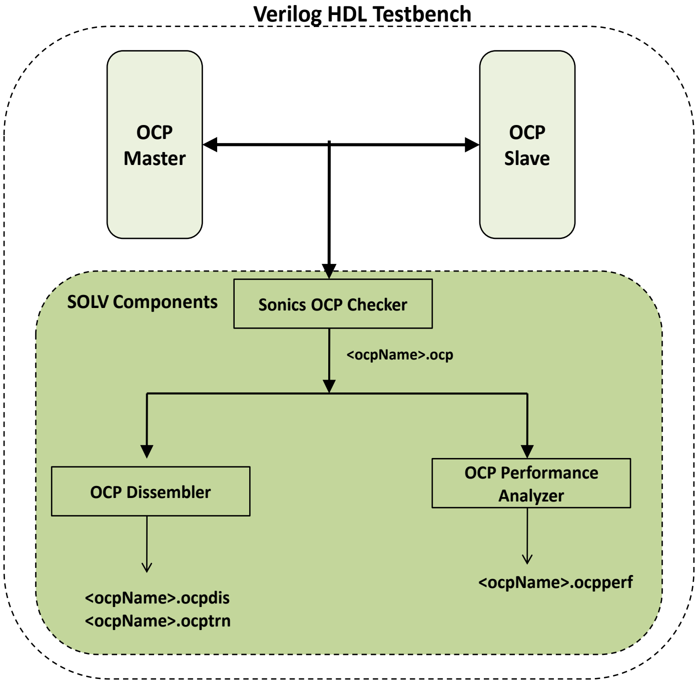


Figure 5.1. SOLV Components and Tool Flow

activity in a convenient report format. An OCP performance analyzer uses the OCP trace files to measure the performance of OCP basic transfers and burst transactions.

5.1 OCP Checker Setup and Instantiation

Sonics provides system Verilog files for a limited set of simulators for the OCP checker. Software is also provided for command line tools: the OCP dissembler and performance analyzer. SOLV supports NC Verilog, VCS, and MTI simulators.

The Sonics OCP Checker can be instantiated using a Verilog module containing two maps: one for instance and protocol parameters and another for ports. Each OCP connection has a unique set of instance and protocol configuration parameters which are enabled depending on the supporting OCP subset. Each connection also has a unique set of wires that connects to the SVA checker, and depending on the signals connected, the checker instance port map is defined. When an OCP signal is not specified, the checker reserves a one-bit wide signal for it and uses the default values. Figure 5.2 shows a code snippet for an OCP checker instance and protocol parameter map, and a port map used in this design validation.

```
ocp2_sva_checker #(
//Instance Parameters
.version ("ocp2.2-1.9"),
.checkername ("coretb.ocp"),
.name ("checker_port"),
.trace_name ("master_ocp.ocp"),
.ocpcheck_enable (1),
.trace_enable (1),
.max_idle_enable (1),

//Protocol Parameters.
.cmd(1),
.addr (1),
.addr_wdth (32),
.read_enable (1),
.write_enable (1),
.datahandshake (1),
.burstlength (1),
.burstlength_wdth (3),
.burstprecise (1),
.burstseq (1),
.broadcast_enable (0),
.burstseq_dflt1_enable (1),
.burstseq_incr_enable (1),
.burstseq_wrap_enable (1),
.byteen (1),
.cmdaccept (1),
.data_wdth (32),
.mdata (1),
.resp (1),
.respaccept (1),
.sdata (1),
.dataaccept (1),
.tags (3),
.taginorder (1),
.mreset (1),
.sreset (1),
.threads (0),
..
..)
```

```
\\ Port Map
checker_port (
.Clk_i (Clk),
.MCmd_i (MCmd),
.MAddr_i (MAddr),
.MData_i (MData),
.MDataValid_i (MDataValid),
.MRespAccept_i (MRespAccept),
.SCmdAccept_i (SCmdAccept),
.SData_i (SData),
.SDataAccept_i (SDataAccept),
.SResp_i (SResp),
.MByteEn_i (MByteEn),
.MBurstLength_i (MBurstLength),
.MBurstPrecise_i (MBurstPrecise),
.MBurstSeq_i (1'b0),
.MTagID_i (MTagID),
.MTagInOrder_i (MTagInOrder),
.STagID_i (STagID),
.STagInOrder_i (STagInOrder),
.MReset_ni (RST),
.SReset_ni (RST),
..
..);
```

Figure 5.2. OCP Checker Code Snippet

CHAPTER 6

DESIGN TESTING AND RESULTS

6.1 Testing Environment

Figure 6.1 shows the test setup used for this design. The design uses an emulated RISC pipelined CPU as Master IP core, a synchronous memory (RAM and ROM) as Slave IP core, and customized 3-port synchronous routers as NoC [18] [19]. A Verilog testbench generates the CPU emulation traffic to communicate with the Slave IP core through the network fabric. The network fabric contains synchronous routers connected back-to-back. Data packets are traversed across the routers and transferred to the destination address by using the source routing information provided by the processor emulator. The Memory module (Slave IP) contains 8KB each of RAM and ROM customized to interact with the Master IP using OCP.

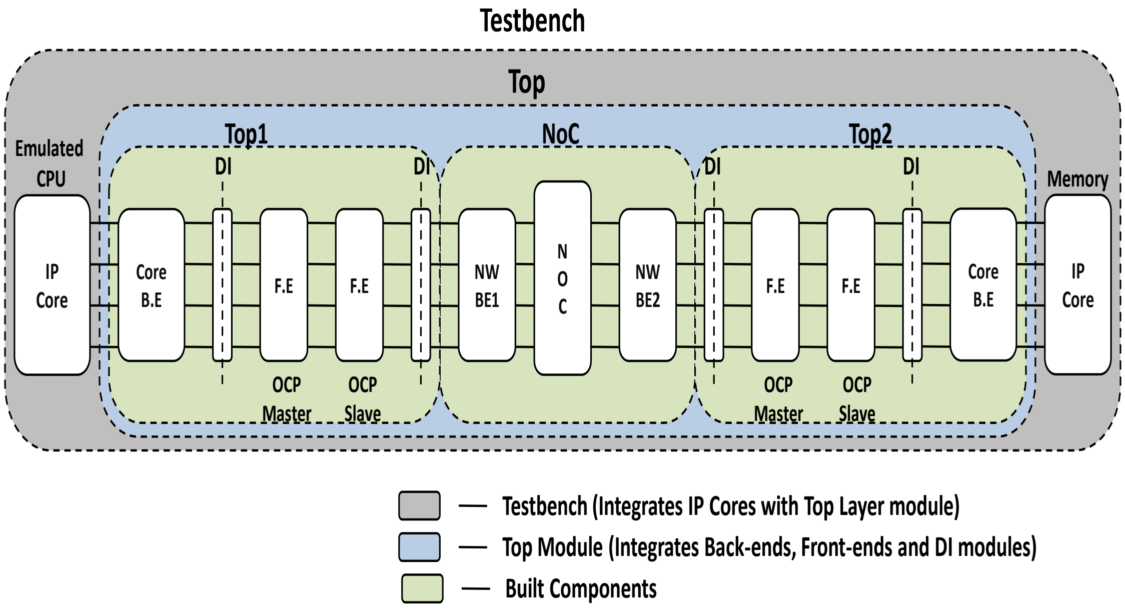


Figure 6.1. Design Test Setup

A Verilog HDL test bench (test.v) is coded to manually integrate the synthesized submodules (Top1, Top2, and NoC) containing back-ends, front ends, and DI modules with the IP cores. The test bench creates the clock and generates other required stimulus to the top layer module (top.v). Different testbenches are coded to test single- and multifrequency clocked designs.

Nonsplit and split operating modes are tested with the same stimulus data, but applied at different run times. In both modes, the same number of data tokens are applied which includes simple memory transactions through burst transactions. In nonsplit mode, as pipelining of requests is not possible, the traffic corresponds to simple memory transactions. Split mode supports pipelining of requests, including burst transactions. During nonsplit transactions, the Master IP reserves a request channel, and cannot initiate on any more request transactions until it receives the response for the previous initiated transaction. The nonsplit mode test bench uses nonburst transactions and the split mode test bench uses burst transactions with a burst size of four. This produces the worst and best case traffic scenarios. Both test benches have an average 15% data activity factor per transaction for the 36 data tokens. Tables 6.1 and 6.2 shows nonsplit mode and split mode test benches used in this design, respectively.

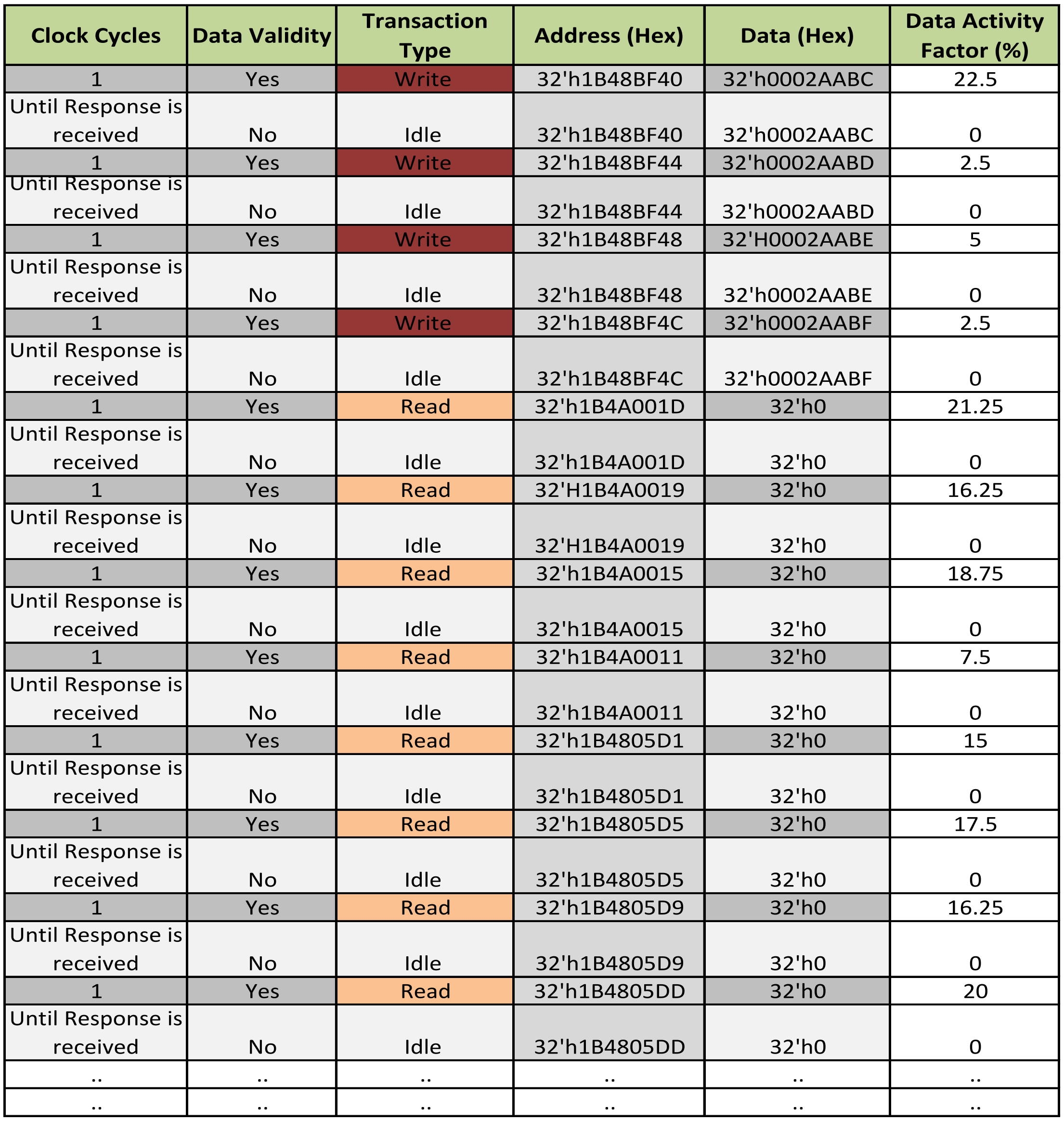
The design uses a 16-bit address, 32-bit data paths, and other control informations such as bursting, byte enables, and tagging. Each request transaction validity is determined using cycle (CYC), strobe (STB), write enable (WE), address, and data signals. Simultaneous responses are determined using acknowledge or response (ACK) signal and tagging information (STagID). In this design implementation, incremental (INCR), wrap, and user-defined address sequences are supported.

6.2 Performance Models and Results

6.2.1 Performance Metrics

Performance metrics are reported for single- and multifrequency clocked designs. Metrics include target frequency of operation, latency, average power, energy per transaction, area, and maximum bandwidth across the network. Simulations are performed on the postlayout designs with delays extracted from SoC encounter based

Table 6.1. Nonsplit Mode Test Bench



Clock Cycles	Data Validity	Transaction Type	Address (Hex)	Data (Hex)	Data Activity Factor (%)
1	Yes	Write	32'h1B48BF40	32'h0002AABC	22.5
Until Response is received	No	Idle	32'h1B48BF40	32'h0002AABC	0
1	Yes	Write	32'h1B48BF44	32'h0002AABD	2.5
Until Response is received	No	Idle	32'h1B48BF44	32'h0002AABD	0
1	Yes	Write	32'h1B48BF48	32'H0002AABE	5
Until Response is received	No	Idle	32'h1B48BF48	32'h0002AABE	0
1	Yes	Write	32'h1B48BF4C	32'h0002AABF	2.5
Until Response is received	No	Idle	32'h1B48BF4C	32'h0002AABF	0
1	Yes	Read	32'h1B4A001D	32'h0	21.25
Until Response is received	No	Idle	32'h1B4A001D	32'h0	0
1	Yes	Read	32'H1B4A0019	32'h0	16.25
Until Response is received	No	Idle	32'H1B4A0019	32'h0	0
1	Yes	Read	32'h1B4A0015	32'h0	18.75
Until Response is received	No	Idle	32'h1B4A0015	32'h0	0
1	Yes	Read	32'h1B4A0011	32'h0	7.5
Until Response is received	No	Idle	32'h1B4A0011	32'h0	0
1	Yes	Read	32'h1B4805D1	32'h0	15
Until Response is received	No	Idle	32'h1B4805D1	32'h0	0
1	Yes	Read	32'h1B4805D5	32'h0	17.5
Until Response is received	No	Idle	32'h1B4805D5	32'h0	0
1	Yes	Read	32'h1B4805D9	32'h0	16.25
Until Response is received	No	Idle	32'h1B4805D9	32'h0	0
1	Yes	Read	32'h1B4805DD	32'h0	20
Until Response is received	No	Idle	32'h1B4805DD	32'h0	0
..	..	..	..	..	..
..	..	..	..	..	..

on the layout and parasitics of the design. Performance metrics are determined using a Verilog simulation in the Modelsim simulator. The difference in performance between nonsplit and split modes are calculated using different Verilog test benches.

Target frequency of operation is the maximum frequency at which the physically routed design works with setup and hold time requirements. For both clocked designs, the maximum operating frequencies corresponding to the IP cores and NoC are reported. Latency is calculated as the amount of time required for each request transaction to complete in nonsplit and split operating modes. It is computed from

Table 6.2. Split Mode Test Bench

Clock Cycles	Data Validity	Transaction Type	Address (Hex)	Data (Hex)	Burst Type	Data Activity Factor (%)
1	Yes	Write	32'h1B48BF40	32'h0002AABC	INCR	22.5
1	Yes	Write	32'h1B48BF44	32'h0002AABD		2.5
1	Yes	Write	32'h1B48BF48	32'H0002AABE		5
1	Yes	Write	32'h1B48BF4C	32'h0002AABF		2.5
1	No	Idle	32'h1A00BF4C	32'h0002AABF	NA	0
1	Yes	Read	32'h1B4A001D	32'h0	WRAP	21.25
1	Yes	Read	32'H1b4A0019	32'h0		16.25
1	Yes	Read	32'h1B4A0015	32'h0		18.75
1	Yes	Read	32'h1B4A0011	32'h0		7.5
1	No	Idle	32'h1B4A0011	32'h0	NA	0
1	Yes	Read	32'h1B4805D1	32'h0	INCR	15
1	Yes	Read	32'h1B4805D5	32'h0		17.5
1	Yes	Read	32'h1B4805D9	32'h0		16.25
1	Yes	Read	32'h1B4805DD	32'h0		20
1	No	Idle	32'h1B4805DD	32'h0	NA	0
1	Yes	Write	32'h1B48BF60	32'h0002AABC	INCR	26.25
1	Yes	Write	32'h1B48BF64	32'h0002AABD		2.5
1	Yes	Write	32'h1B48BF68	32'H0002AABE		5
1	Yes	Write	32'h1B48BF6C	32'h0002AABF		2.5
..	..	..	..	..	..	..
..	..	..	..	..		

the number of clock cycles required per transaction. Area is reported from postlayout design, by summing up the areas of submodules (Top1, Top2, and NoC). The maximum bandwidth across the network is calculated depending on the latency associated in the operating modes and frequency of operation.

Power is measured using simulation switching activity of the physical design including extracted parasitics. Power measured for the designs includes only the NoC interface designs (it does not include IP cores power). The Modelsim simulator generates a VCD file containing activity factors for all nodes and registers of the design, and SoC Encounter calculates the extracted layout parasitics. The Synopsys Primetime tool is used in this research to determine accurate power numbers with the generated VCD file. Primetime reports total average power which includes the net switching power, cells internal power, and cells leakage power. The average power reported from SoC Encounter is also presented as part of the power metric calculation.

Energy is reported in addition to the average power since it is the best metric to distinguish the efficiency in nonsplit and split operation modes. Energy per transaction is computed from the reported total average power and number of transactions in the simulation test time. The average power required to perform a nonsplit transaction is less but the energy is much greater, because of the large differences in run-times.

6.2.2 Performance Models

Parameterized computational models are developed to calculate performance for single- and multifrequency clocked design configurations. The models can be used to determine the performance metrics of latency, energy per transaction, and maximum bandwidth for any given NoC configuration.

6.2.2.1 Model for Latency Cycles

Models are developed with reference to Figure 4.5. Latency for each transaction in the design is computed depending on the required number of clock cycles in the request and response path. For easier computation, the design is segregated into three components: the Master IP interface block (MIB), network interface block (NIB), and Slave IP interface block (SIB). The Master IP interface block comprises a master back-end (B.E), domain interface modules (DI1 & DI2), and front-end OCP entities (F.E). The network block comprise of network back-ends (1 & 2) and network routers. The Slave IP block comprise slave back-end, domain interface modules (DI3 & DI4), and OCP entities. The latency model for the design is computed as follows:

$$TotalLatency \Leftarrow Latency\,(Request\ Path\ +\ Response\ Path +\ IP\ Cycles\,) \tag{6.1}$$

```
where:
Latency in Request Path: MIB + NIB + SIB
Latency in Response Path: SIB + NIB + MIB
IP Cycles : Sum of IPs request and response cycles
```

```
Request Path Latency:
MIB : Master B.E + DI1 + OCP Master + OCP Slave + DI2    (Clk Domain 1)
NIB : Sync + Network B.E1 + NoC + Network B.E2 + DI3     (Clk Domain 2)
SIB : Sync + OCP Master + OCP Slave + DI4 + Slave B.E    (Clk Domain 3)

Response Path Latency:
SIB : Slave B.E + DI4 + OCP Slave  + OCP Master + DI3    (Clk Domain 3)
NIB : Sync + Network B.E2 + NoC + Network B.E1 + DI2     (Clk Domain 2)
MIB : Sync + OCP Slave + OCP Master + DI1 + Master B.E   (Clk Domain 1)
```

Latency Cycles Across the Design Path:

The Master IP (CPU emulated test bench) generates traffic and latches data into the customized back-end interface (1-clock cycle). The IP cores and NoC back-ends introduce a 1-clock cycle delay in the request and response paths for latching the signals and mapping them onto the DI module signals. In the single-frequency clocked domain, the DI modules do not introduce any clock cycle delays because of the simple combinational logic implementation used to map its signals onto OCP compatible signals (0-clock cycle). In case of multifrequency clocked domain, the DI modules 2 and 3 are designed using asynchronous dual clocked FIFOs to synchronize asynchronous clock domains, as shown in Figure 4.5. Here the request and response path DI2 and DI3 modules introduce synchronization overhead. Synchronization requires 4-clock cycles of latency which includes 1-clock cycle in the source clock domain for latching data into DI module, 2-cycles in the destination clock domain for synchronization, and 1-clock cycle to send out the data.

In the request path, OCP entities introduce 3-clock cycles of latency which includes 1-clock cycle for the OCP Master to latch data from the DI module and 1-clock cycle to initiate request command and 1-clock cycle to send request transaction data onto the OCP Slave entity. In the response path, the OCP entities introduce only 2-clock cycles of latency, including 1-clock cycle for the OCP slave entity to latch response data from the DI module, and 1-clock cycle to send out response data to the OCP master entity. The NoC introduces variable clock cycle delays depending on the number of routers. Each router introduces 1-clock cycle of delay in request and response paths. The IP cycles includes a 1-clock cycle latency for the Master IP (test bench) to accept the response data from its back-end. The customized memory module is employed as the Slave IP in this design, which takes 2-clock cycles of latency to accepting the request data from the back-end module and to send out response data acknowledge on successful completion. The following latency equations are used to derive latency cycles in nonsplit and split modes:

$$NSP\,mode\,cycles \Leftarrow (Latency\,from\,eqn.\,6.1) \times No.\,of\,Transactions \quad (6.2)$$

$$SP\,mode\,cycles \Leftarrow (Latency\,from\,eqn.\,6.1) + No.\,of\,Transactions \quad (6.3)$$

6.2.2.2 Model for Energy per Transaction

The energy per transaction in nonsplit and split modes can be determined from total average power, test time, and number of transactions. The total energy can be computed using the equation below:

$$Energy/Transaction \Leftarrow (Ave.\,Power \times Test\,time)\,/No.ofTransactions (6.4)$$

Test time for NSP mode and SP mode can be derived as follows:
Test time (NSP mode) $\leq$ (Latency cycles from eqn 6.2 + Idle cycles) × Time period.
Test time (SP mode) $\leq$ (Latency cycles from eqn 6.3 + Idle cycles) × Time period.

6.2.2.3 Maximum Bandwidth

The maximum bandwidth (BW) is theoretically derived based on the throughput of the requests. In split mode, a request transaction can be initiated on every clock cycle. Thus, the maximum bandwidth across the network fabric is the maximum operating clock frequency of that design. However, in nonsplit mode, the maximum bandwidth is limited by the transaction response time. Therefore, the maximum bandwidth for nonsplit mode is determined by the associated latency cycles. The following equations define theoretical maximum bandwidths for nonsplit and split operating modes.

For NSP Mode:

$$MaximumBW \Leftarrow (Max.Operatingfrequency)/(LatencycyclesfromEqn6.2) \quad (6.5)$$

For SP Mode:

$$Maximum\,BW \quad \Leftarrow \quad Max.Operating\,frequency \qquad (6.6)$$

6.2.3 Simulation Results

6.2.3.1 Single-Frequency Clocked Domain

In this domain, the Verilog testbench generates a global clock for the IP cores, NoC, and IP cores. This design employs 1-router NoC with the same test stimulus applied in NSP and SP modes. Performance metrics are reported using the simulation results and performance models as follows:

i. **Design Maximum Frequency of Operation**: 1.35 GHz

ii. **Latency**:

Time period : 0.74ns

Number of transactions in NSP mode : 36

Number of transactions in SP mode : 9 (Burst size = 4)

Equation 6.1 is used to calculate the number of cycles required for each transaction for this design configuration. Equations 6.2 and 6.3 are used to determine the latency cycles for NSP and SP modes, respectively.

Latency for each transaction (or) Number of cycles for each transaction :

Latency cycles in request path : MIB (4) + NIB (3) + SIB (4) = 11 cycles

Latency cycles in response path : SIB (3) + NIB (3) + MIB (3) = 9 cycles

Total latency cycles : 11 + 9 + 4 (Master IP (2) + Slave IP (2)) = 24 cycles

Latency cycles in NSP mode : $(24 \times 36) \rightarrow 864$ cycles

Latency cycles in SP mode : $(24 + 36) \rightarrow 60$ cycles

Latency cycles/transaction in NSP mode : $864/36 \rightarrow 24$ cycles

Latency cycles/transaction in SP mode : $60/9 \rightarrow 6.67$ cycles

iii. **Maximum Bandwidth**: Equations 6.5 and 6.6 are used to calculate theoretical maximum bandwidths for NSP and SP modes, respectively.

Maximum bandwidth (NSP mode): 1.35(GHz)/24 → 56.25 MFlits/sec

Maximum bandwidth (SP mode): 1.35 GFlits/sec

iv. **Area**: Table 6.3 presents the individual and total area of the automatic place and routed design blocks.

v. **Average Power and Energy per Transaction**:

The total average power and energy/transaction are computed from individual synthesized components (Top1, Top2, and NoC). Table 6.4 presents average power reported from SoC Encounter. Tables 6.5, 6.6, and 6.7 presents average power for Top1, Top2, and NoC blocks from Synopsis Primetime, respectively. Total energy for NSP and SP modes are calculated using equation 6.4.

Test time (NSP/SP mode) ← (Latency cycles in NSP/SP mode + Idle cycles) × Time period

Idle cycles in NSP mode → 28 cycles

Idle cycles in SP mode → 9 cycles

Test time in NSP mode: (864 + 28) x 0.74(ns) → 667.5 ns

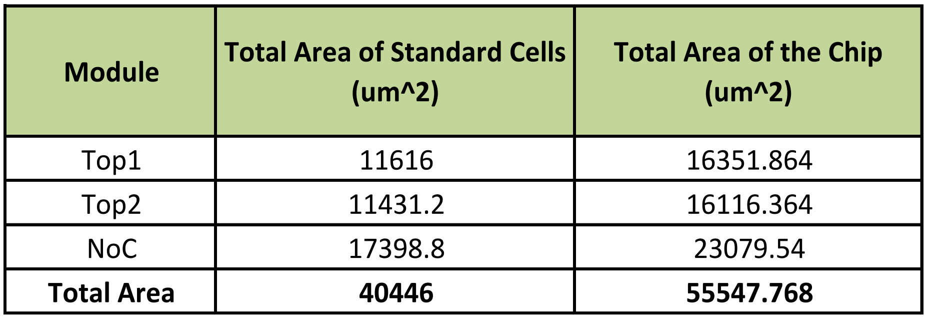
Test time in SP mode: (60 + 9) x 0.74 (ns) → 51.1 ns

Modelsim Simulation Reported Test Times:

In NSP mode : 666.990 ns

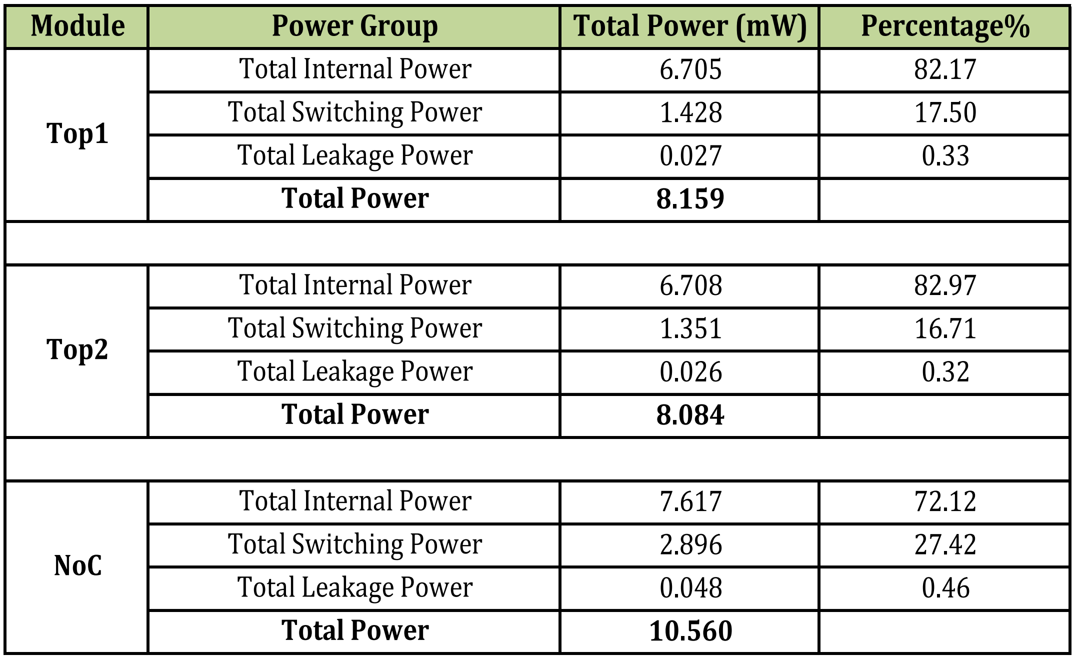
In SP mode : 51.31 ns

Table 6.3. SFCD: Individual and Total Area Reported from SoC Encounter in Case of 1-router NoC



Module	Total Area of Standard Cells (um^2)	Total Area of the Chip (um^2)
Top1	11616	16351.864
Top2	11431.2	16116.364
NoC	17398.8	23079.54
Total Area	**40446**	**55547.768**

Table 6.4. SFCD: Reported Total Average Power from SoC Encounter in Case of 1-router NoC



Module	Power Group	Total Power (mW)	Percentage%
Top1	Total Internal Power	6.705	82.17
	Total Switching Power	1.428	17.50
	Total Leakage Power	0.027	0.33
	Total Power	**8.159**	
Top2	Total Internal Power	6.708	82.97
	Total Switching Power	1.351	16.71
	Total Leakage Power	0.026	0.32
	Total Power	**8.084**	
NoC	Total Internal Power	7.617	72.12
	Total Switching Power	2.896	27.42
	Total Leakage Power	0.048	0.46
	Total Power	**10.560**	

Table 6.8 presents the computation involved in calculating total average power and energy per transaction in nonsplit and split modes.

Total average power in NSP mode : 22.402 mW

Total average power in SP mode : 28.336 mW

Total energy in NSP mode : 14.942 nJ

Total energy in SP mode : 1.454 nJ

Energy per transaction in NSP mode: 0.415 nJ

Energy per transaction in SP mode: 0.162 nJ

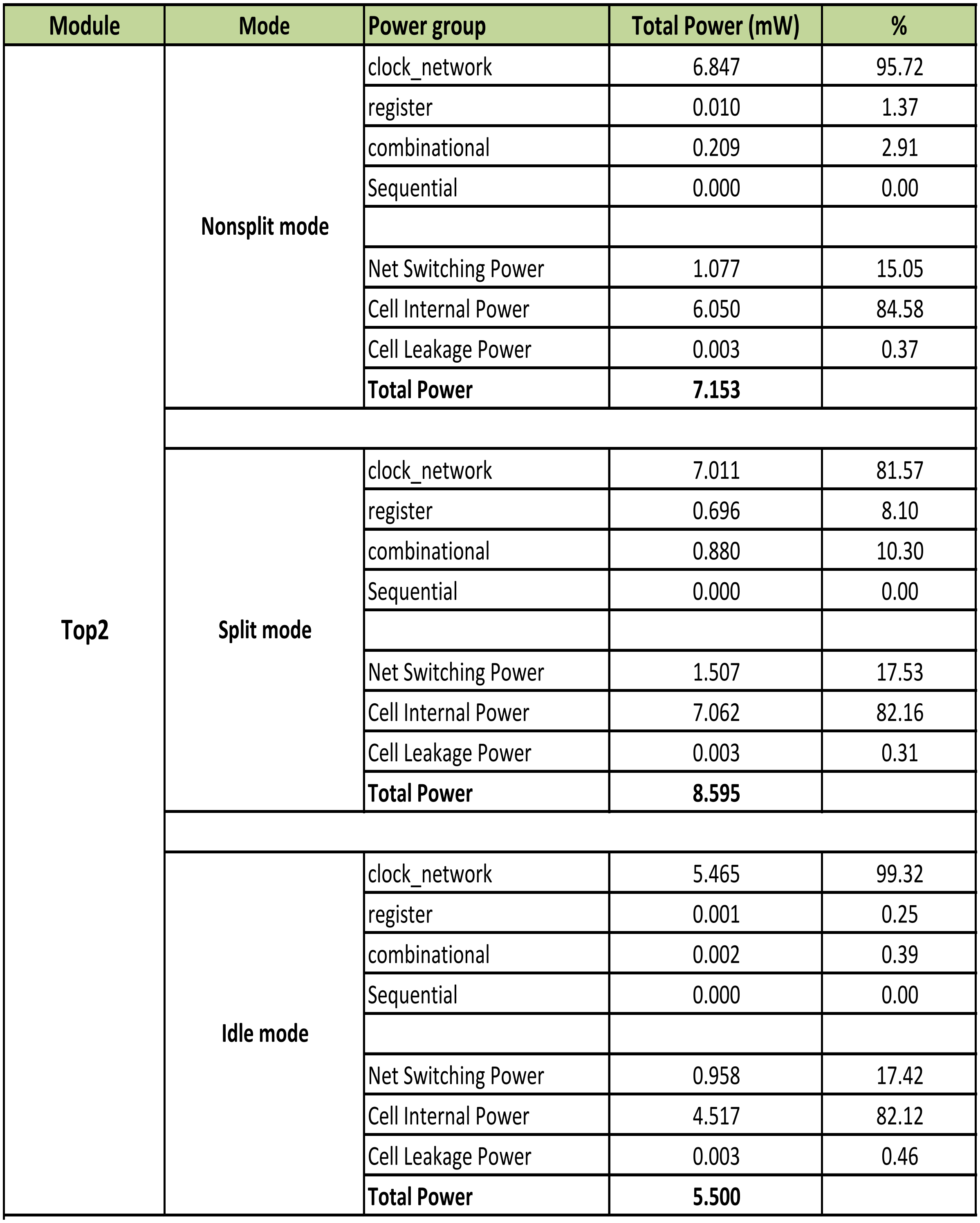
6.2.3.2 Multifrequency Clocked Domain

In this domain, the Verilog testbench generates three asynchronous clocks for the Master IP, Slave IP, and NoC, respectively. This design employs 2 NoC routers with the same test stimulus applied in NSP and SP modes. Performance is reported using the simulation results and performance models as follows:

Table 6.5. SFCD: Reported Top1 Module Average Power from Primetime in Case of 1-router NoC

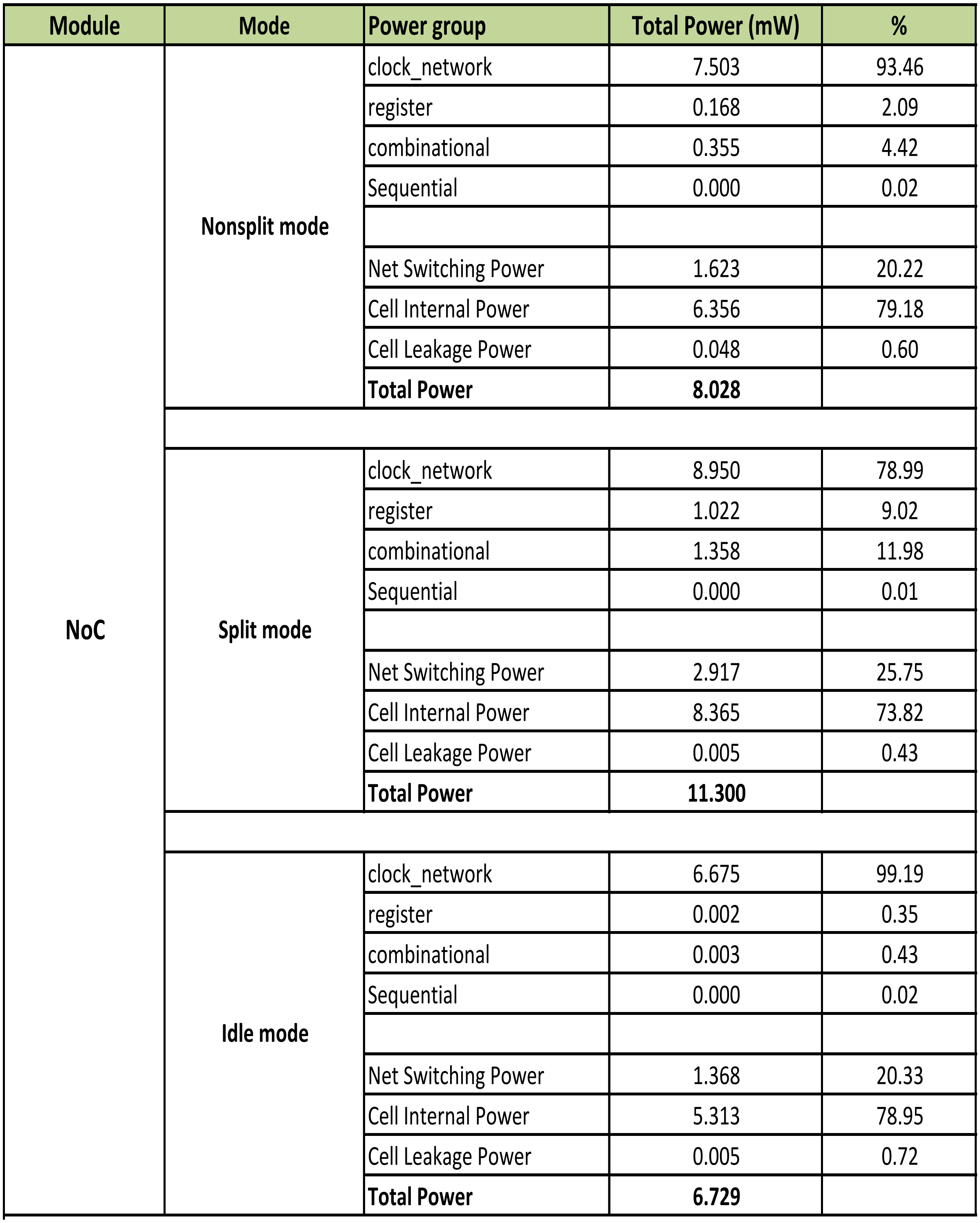
Module	Mode	Power group	Total Power (mW)	%
Top1	Nonsplit mode	clock_network	6.900	95.56
		register	0.154	1.54
		combinational	0.210	2.91
		Sequential	0.000	0.00
		Net Switching Power	1.104	15.28
		Cell Internal Power	6.090	84.34
		Cell Leakage Power	0.003	0.37
		Total Power	**7.221**	
	Split mode	clock_network	6.840	81.03
		register	0.732	8.67
		combinational	0.870	10.30
		Sequential	0.000	0.00
		Net Switching Power	1.155	18.40
		Cell Internal Power	6.861	81.28
		Cell Leakage Power	0.003	0.32
		Total Power	**8.441**	
	Idle mode	clock_network	5.518	99.32
		register	0.001	0.25
		combinational	0.002	0.42
		Sequential	0.000	0.00
		Net Switching Power	0.982	17.68
		Cell Internal Power	4.547	81.84
		Cell Leakage Power	0.003	0.47
		Total Power	**5.556**	

Table 6.6. SFCD: Reported Top2 Module Average Power from Primetime in Case of 1-router NoC



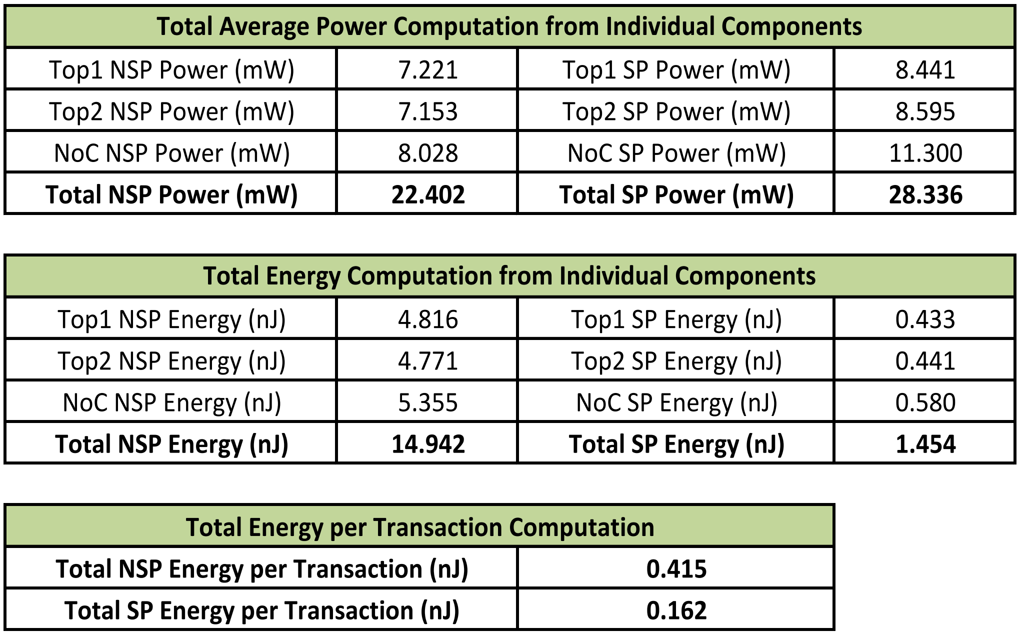
Module	Mode	Power group	Total Power (mW)	%
Top2	Nonsplit mode	clock_network	6.847	95.72
		register	0.010	1.37
		combinational	0.209	2.91
		Sequential	0.000	0.00
		Net Switching Power	1.077	15.05
		Cell Internal Power	6.050	84.58
		Cell Leakage Power	0.003	0.37
		Total Power	**7.153**	
	Split mode	clock_network	7.011	81.57
		register	0.696	8.10
		combinational	0.880	10.30
		Sequential	0.000	0.00
		Net Switching Power	1.507	17.53
		Cell Internal Power	7.062	82.16
		Cell Leakage Power	0.003	0.31
		Total Power	**8.595**	
	Idle mode	clock_network	5.465	99.32
		register	0.001	0.25
		combinational	0.002	0.39
		Sequential	0.000	0.00
		Net Switching Power	0.958	17.42
		Cell Internal Power	4.517	82.12
		Cell Leakage Power	0.003	0.46
		Total Power	**5.500**	

Table 6.7. SFCD: Reported NoC Module Average Power from Primetime in Case of 1-router NoC



Module	Mode	Power group	Total Power (mW)	%
NoC	Nonsplit mode	clock_network	7.503	93.46
		register	0.168	2.09
		combinational	0.355	4.42
		Sequential	0.000	0.02
		Net Switching Power	1.623	20.22
		Cell Internal Power	6.356	79.18
		Cell Leakage Power	0.048	0.60
		Total Power	**8.028**	
	Split mode	clock_network	8.950	78.99
		register	1.022	9.02
		combinational	1.358	11.98
		Sequential	0.000	0.01
		Net Switching Power	2.917	25.75
		Cell Internal Power	8.365	73.82
		Cell Leakage Power	0.005	0.43
		Total Power	**11.300**	
	Idle mode	clock_network	6.675	99.19
		register	0.002	0.35
		combinational	0.003	0.43
		Sequential	0.000	0.02
		Net Switching Power	1.368	20.33
		Cell Internal Power	5.313	78.95
		Cell Leakage Power	0.005	0.72
		Total Power	**6.729**	

Table 6.8. SFCD: Total Average Power and Energy per Transaction Computation in Case of 1-router as NoC



Total Average Power Computation from Individual Components			
Top1 NSP Power (mW)	7.221	Top1 SP Power (mW)	8.441
Top2 NSP Power (mW)	7.153	Top2 SP Power (mW)	8.595
NoC NSP Power (mW)	8.028	NoC SP Power (mW)	11.300
Total NSP Power (mW)	**22.402**	**Total SP Power (mW)**	**28.336**

Total Energy Computation from Individual Components			
Top1 NSP Energy (nJ)	4.816	Top1 SP Energy (nJ)	0.433
Top2 NSP Energy (nJ)	4.771	Top2 SP Energy (nJ)	0.441
NoC NSP Energy (nJ)	5.355	NoC SP Energy (nJ)	0.580
Total NSP Energy (nJ)	**14.942**	**Total SP Energy (nJ)**	**1.454**

Total Energy per Transaction Computation	
Total NSP Energy per Transaction (nJ)	**0.415**
Total SP Energy per Transaction (nJ)	**0.162**

i. **Design Maximum Frequency of Operation**:

Master IP : 1 GHz

Slave IP : 0.925 GHz

NoC : 1.11 GHz

ii. **Latency**:

Time period : 1.08 ns

Number of transactions in NSP mode : 36

Number of transactions in SP mode : 9 (burst size = 4) Total number of latency cycles: 42

Equation 6.1 is used to calculate the number of cycles required for each transaction for this design configuration. Equations 6.2 and 6.3 are used to determine the latency for NSP and SP modes, respectively.

Latency cycles in Req path : MIB (5) + NIB (8) + SIB (7) = 20 cycles

Latency cycles in Resp path : SIB (4) + NIB (8) + MIB (6) = 18 cycles

Total latency cycles : 20 + 18 + 4 = 42 cycles

Latency cycles/transaction in NSP mode : 1512/36 → 42 cycles

Latency cycles/transaction in SP mode : 78/9 → 8.67 cycles

iii. **Maximum Bandwidth**: Equations 6.5 and 6.6 are used to calculate theoretical maximum bandwidths for NSP and SP modes, respectively.

Maximum bandwidth (NSP mode): 0.925(GHz)/42 → 22.02 MFlits/sec

Maximum bandwidth (SP mode): 0.925 GFlits/sec

iv. **Area**: Table 6.9 presents the individual and total area of the automatic place and routed blocks.

v. **Average Power and Energy per Transaction**:

The total average power and energy/transaction are computed from individual synthesized components (Top1, Top2, and NoC). The tabulated results correspond to IPs operating at 0.925 GHz and the NoC at 1.11 GHz. Table 6.10 presents the average power reported for the design from SoC Encounter. Tables 6.11, 6.12, and 6.13 present the average power for Top1, Top2, and NoC modules, respectively, using Primetime. Total energy for NSP and SP modes are calculated using eqn 6.4.

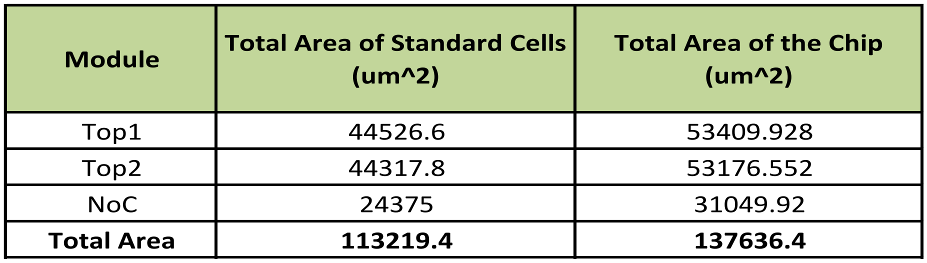
Idle cycles in NSP mode → 2 cycles

Idle cycles in SP mode → 9 cycles

Test time in NSP mode: (1512 + 2) x 1.08(ns) → 1634.8 ns

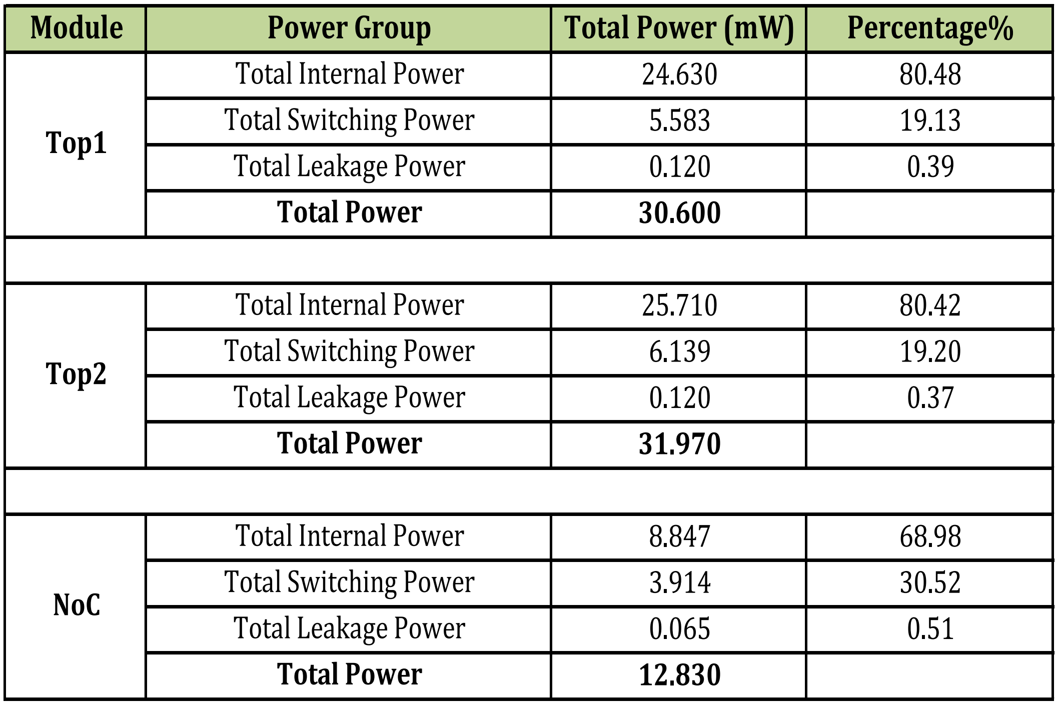
Test time in SP mode: (78 + 9) x 1.08(ns) → 93.96 ns

Table 6.9. MFCD: Individual and Total Area Reported from SoC Encounter in Case of 2-routers NoC



Module	Total Area of Standard Cells (um^2)	Total Area of the Chip (um^2)
Top1	44526.6	53409.928
Top2	44317.8	53176.552
NoC	24375	31049.92
Total Area	**113219.4**	**137636.4**

Table 6.10. MFCD: Reported Total Average Power from SoC Encounter in Case of 2-routers NoC



Module	Power Group	Total Power (mW)	Percentage%
Top1	Total Internal Power	24.630	80.48
	Total Switching Power	5.583	19.13
	Total Leakage Power	0.120	0.39
	Total Power	**30.600**	
Top2	Total Internal Power	25.710	80.42
	Total Switching Power	6.139	19.20
	Total Leakage Power	0.120	0.37
	Total Power	**31.970**	
NoC	Total Internal Power	8.847	68.98
	Total Switching Power	3.914	30.52
	Total Leakage Power	0.065	0.51
	Total Power	**12.830**	

Modelsim Simulation Reported Test Times:

In NSP mode : 1634.350 ns

In SP mode : 94.25 ns

Table 6.14 presents the computation involved in calculating total average power and energy per transaction in nonsplit and split modes.

Total average power in NSP mode : 46.579 mW

Total average power in SP mode : 56.800 mW

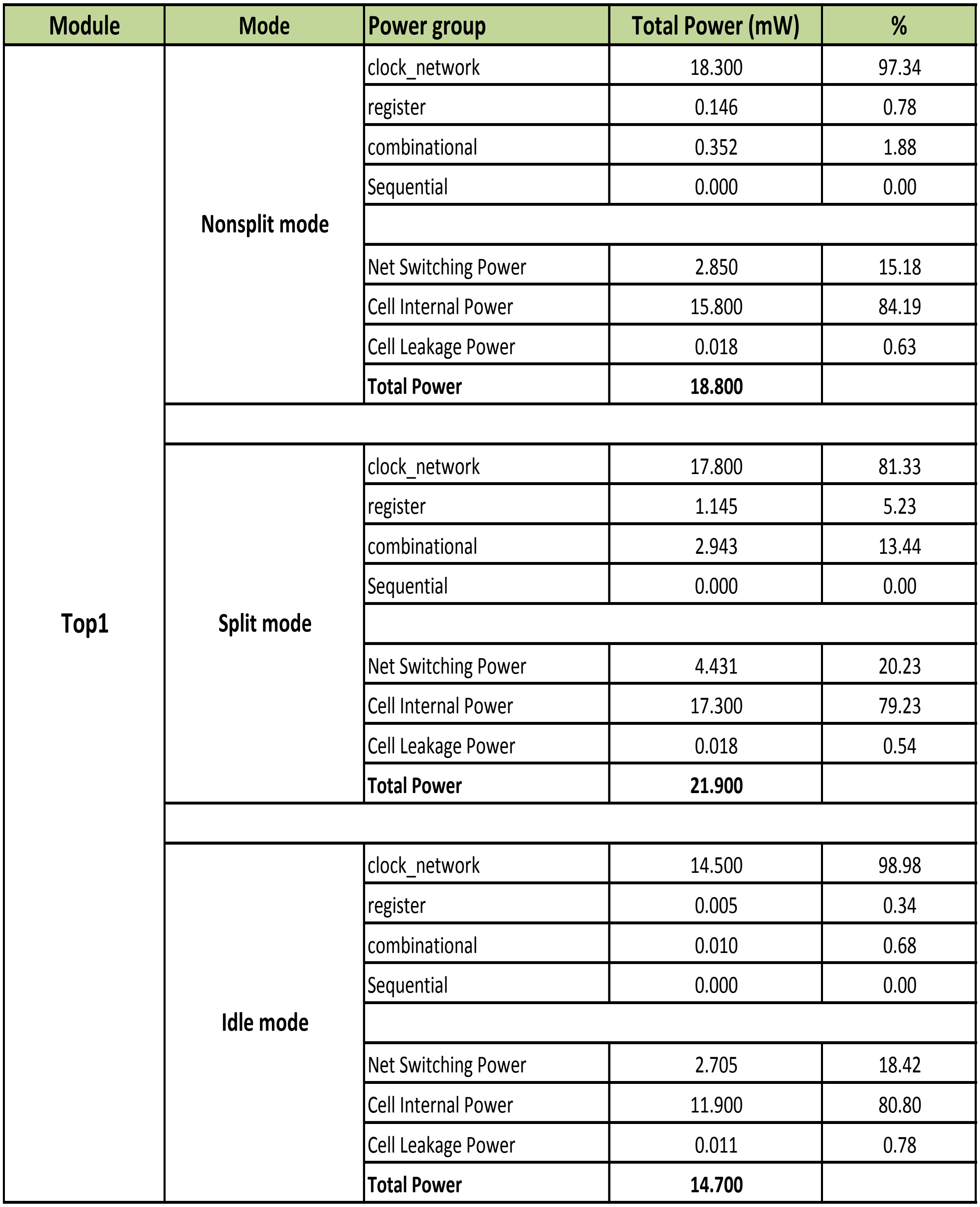
Total energy in NSP mode : 76.126 nJ

Total energy in SP mode : 5.353 nJ

Energy per transaction in NSP mode: 2.114 nJ

Energy per transaction in SP mode: 0.595 nJ

Table 6.11. MFCD: Reported Top1 Module Average Power from Primetime in Case of 2-routers NoC



Module	Mode	Power group	Total Power (mW)	%
Top1	Nonsplit mode	clock_network	18.300	97.34
		register	0.146	0.78
		combinational	0.352	1.88
		Sequential	0.000	0.00
		Net Switching Power	2.850	15.18
		Cell Internal Power	15.800	84.19
		Cell Leakage Power	0.018	0.63
		Total Power	**18.800**	
	Split mode	clock_network	17.800	81.33
		register	1.145	5.23
		combinational	2.943	13.44
		Sequential	0.000	0.00
		Net Switching Power	4.431	20.23
		Cell Internal Power	17.300	79.23
		Cell Leakage Power	0.018	0.54
		Total Power	**21.900**	
	Idle mode	clock_network	14.500	98.98
		register	0.005	0.34
		combinational	0.010	0.68
		Sequential	0.000	0.00
		Net Switching Power	2.705	18.42
		Cell Internal Power	11.900	80.80
		Cell Leakage Power	0.011	0.78
		Total Power	**14.700**	

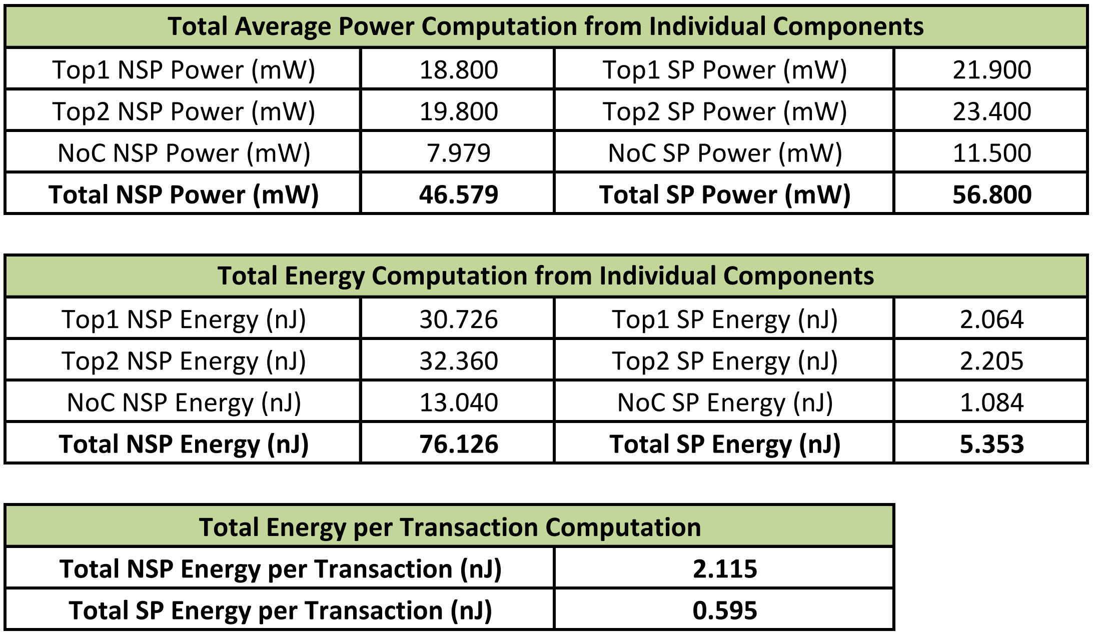
Table 6.12. MFCD: Reported Top2 Module Average Power from Primetime in Case of 2-routers NoC

Module	Mode	Power group	Total Power (mW)	%
Top2	Nonsplit mode	clock_network	19.300	97.50
		register	0.014	0.71
		combinational	0.355	1.80
		Sequential	0.000	0.00
		Net Switching Power	2.898	14.67
		Cell Internal Power	16.700	84.73
		Cell Leakage Power	0.012	0.60
		Total Power	**19.800**	
	Split mode	clock_network	19.300	82.53
		register	1.082	4.62
		combinational	3.007	12.85
		Sequential	0.000	0.00
		Net Switching Power	4.445	18.99
		Cell Internal Power	18.800	80.50
		Cell Leakage Power	0.017	0.50
		Total Power	**23.400**	
	Idle mode	clock_network	15.400	99.06
		register	0.005	0.32
		combinational	0.010	0.62
		Sequential	0.000	0.00
		Net Switching Power	2.759	17.70
		Cell Internal Power	12.700	81.57
		Cell Leakage Power	0.011	0.73
		Total Power	**15.600**	

Table 6.13. MFCD: Reported NoC Module average Power from Primetime in Case of 2-routers NoC

Module	Mode	Power group	Total Power (mW)	%
NoC	Nonsplit mode	clock_network	7.695	96.44
		register	0.010	1.29
		combinational	0.018	2.23
		Sequential	0.000	0.04
		Net Switching Power	1.943	24.35
		Cell Internal Power	5.972	74.84
		Cell Leakage Power	0.006	0.81
		Total Power	**7.979**	
	Split mode	clock_network	9.258	80.68
		register	0.083	7.23
		combinational	1.384	12.06
		Sequential	0.000	0.03
		Net Switching Power	3.372	29.39
		Cell Internal Power	8.037	70.04
		Cell Leakage Power	0.006	0.57
		Total Power	**11.500**	
	Idle mode	clock_network	6.973	98.99
		register	0.003	0.47
		combinational	0.003	0.49
		Sequential	0.000	0.04
		Net Switching Power	1.803	25.60
		Cell Internal Power	5.175	73.48
		Cell Leakage Power	0.006	0.92
		Total Power	**7.043**	

Table 6.14. MFCD: Total Average Power and Energy per Transaction Computation in Case of 2-routers as NoC



Total Average Power Computation from Individual Components			
Top1 NSP Power (mW)	18.800	Top1 SP Power (mW)	21.900
Top2 NSP Power (mW)	19.800	Top2 SP Power (mW)	23.400
NoC NSP Power (mW)	7.979	NoC SP Power (mW)	11.500
Total NSP Power (mW)	**46.579**	**Total SP Power (mW)**	**56.800**

Total Energy Computation from Individual Components			
Top1 NSP Energy (nJ)	30.726	Top1 SP Energy (nJ)	2.064
Top2 NSP Energy (nJ)	32.360	Top2 SP Energy (nJ)	2.205
NoC NSP Energy (nJ)	13.040	NoC SP Energy (nJ)	1.084
Total NSP Energy (nJ)	**76.126**	**Total SP Energy (nJ)**	**5.353**

Total Energy per Transaction Computation	
Total NSP Energy per Transaction (nJ)	**2.115**
Total SP Energy per Transaction (nJ)	**0.595**

6.2.4 Design Evaluation

The clocked designs are tested for performance differences between nonsplit and split operating modes over a range of frequencies. At different operating frequencies, the design performance metrics of total average power, energy per transaction, latency, and maximum bandwidth are determined. Tables 6.15 and 6.16 show the summarized performance over a range of frequencies for single- and multifrequency domains. These designs employ a 2-router NoC and use the stimulus shown in Tables 6.1 and 6.2 for NSP and SP modes, respectively.

6.2.4.1 Clocked Designs (SFCD and MFCD)

In SFCD, a global clock is generated to feed the IP cores and NoC. Its frequency is varied from 100 MHz to 1.33 GHz. In MFCD, three asynchronous clocks are generated to feed the Master IP, Slave IP, and NoC. Since multiple cases are possible to evaluate design performance, the following test scenario is considered. The Slave IP and NoC are operated at a fixed target frequency while the Master IP is varied from 100 MHz to 1 GHz and the Slave IP and NoC are operated at their maximum frequencies of

Table 6.15. Performance Metrics Over a Range of Frequencies for Single-Frequency Clocked Domain and Multifrequency Clocked Domain

Single-Frequency Clocked Domain				
Frequency (MHZ)	**Average Power in NSP Mode (mW)**	**Average Power in SP Mode (mW)**	**Energy/Txn in NSP Mode (nJ)**	**Energy/Txn in SP Mode (nJ)**
100	1.9103	2.4469	0.516	0.196
200	3.652	4.384	0.494	0.175
300	5.3937	6.3211	0.486	0.180
400	7.232	9.167	0.489	0.183
500	9.0703	12.0129	0.487	0.184
600	10.858	14.199	0.488	0.189
700	12.6457	16.3851	0.488	0.185
800	14.278	17.365	0.483	0.174
900	15.9103	18.3449	0.485	0.170
1000	17.923	21.511	0.485	0.164
1100	19.9357	24.6771	0.486	0.175
1200	21.457	27.878	0.487	0.188
1300	22.9783	31.0789	0.487	0.187
1333	24.029	31.392	0.487	0.187

Multifrequency Clocked Domain				
Frequency (MHZ)	**Average Power in NSP Mode (mW)**	**Average Power in SP Mode (mW)**	**Energy/Txn in NSP Mode (nJ)**	**Energy/Txn in SP Mode (nJ)**
100	34.222	37.074	14.478	2.533
200	35.767	40.665	7.586	1.446
300	37.312	44.256	6.330	1.250
400	38.67	47	4.117	0.914
500	40.209	49.6	3.431	0.805
600	41.748	52.2	2.910	0.730
700	43.287	54.2	2.710	0.680
825	45.136	55.1	2.328	0.617
1000	47.599	57.5	2.051	0.569

0.925 GHz and 1.11 GHz, respectively. The performance difference between NSP and SP modes for SFCD is graphically presented in Figures 6.2, 6.3, 6.4, and 6.5 as follows:

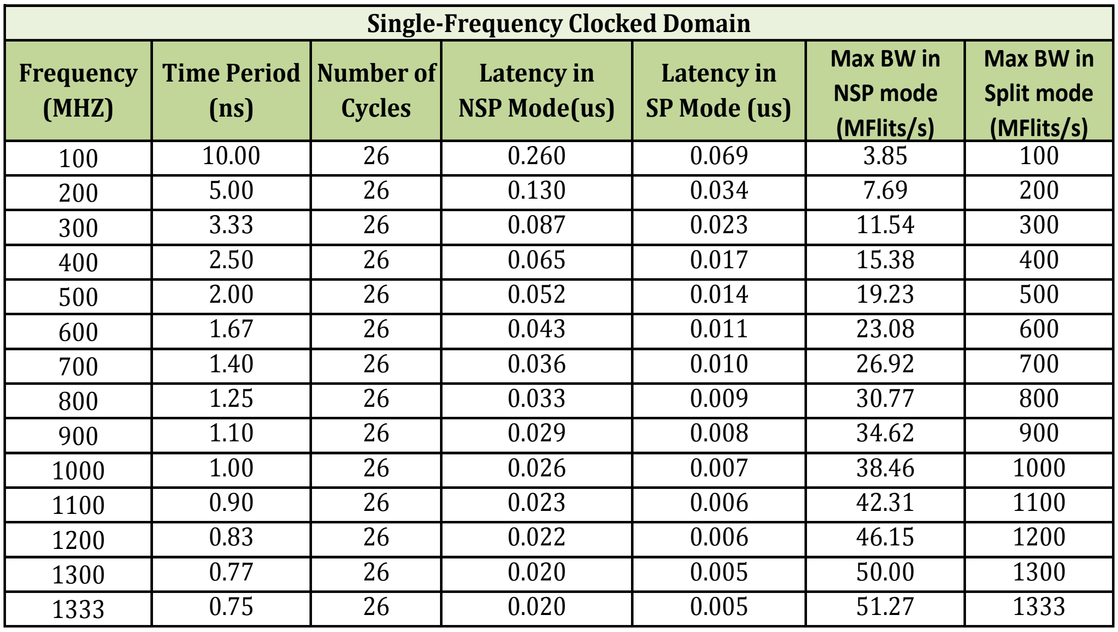
i. **Frequency v/s Average Power**

ii. **Frequency v/s Energy per Transaction**

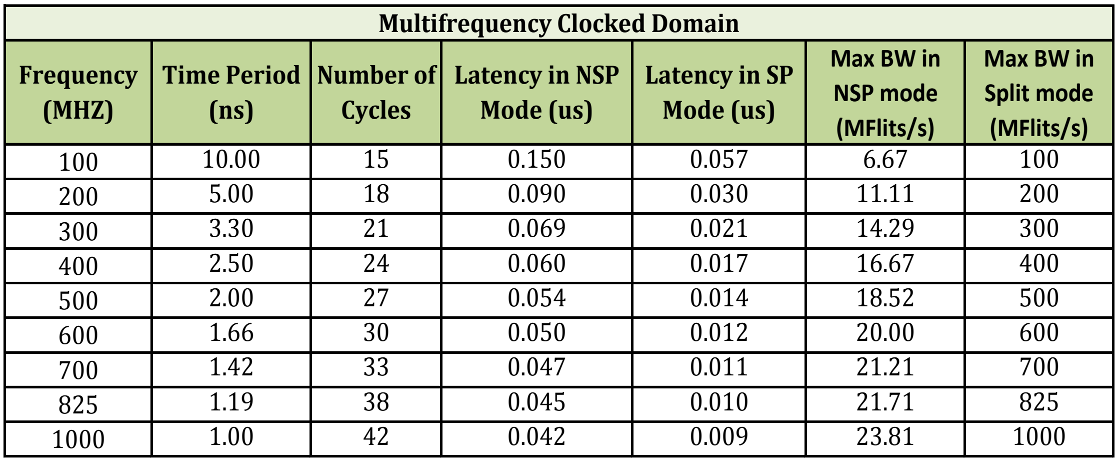
iii. **Frequency v/s Latency**

iv. **Frequency v/s Maximum Bandwidth**

Table 6.16. Performance Metrics Over a Range of Frequencies for Single-Frequency Clocked Domain and Multifrequency Clocked Domain



Single-Frequency Clocked Domain						
Frequency (MHZ)	Time Period (ns)	Number of Cycles	Latency in NSP Mode(us)	Latency in SP Mode (us)	Max BW in NSP mode (MFlits/s)	Max BW in Split mode (MFlits/s)
100	10.00	26	0.260	0.069	3.85	100
200	5.00	26	0.130	0.034	7.69	200
300	3.33	26	0.087	0.023	11.54	300
400	2.50	26	0.065	0.017	15.38	400
500	2.00	26	0.052	0.014	19.23	500
600	1.67	26	0.043	0.011	23.08	600
700	1.40	26	0.036	0.010	26.92	700
800	1.25	26	0.033	0.009	30.77	800
900	1.10	26	0.029	0.008	34.62	900
1000	1.00	26	0.026	0.007	38.46	1000
1100	0.90	26	0.023	0.006	42.31	1100
1200	0.83	26	0.022	0.006	46.15	1200
1300	0.77	26	0.020	0.005	50.00	1300
1333	0.75	26	0.020	0.005	51.27	1333



Multifrequency Clocked Domain						
Frequency (MHZ)	Time Period (ns)	Number of Cycles	Latency in NSP Mode (us)	Latency in SP Mode (us)	Max BW in NSP mode (MFlits/s)	Max BW in Split mode (MFlits/s)
100	10.00	15	0.150	0.057	6.67	100
200	5.00	18	0.090	0.030	11.11	200
300	3.30	21	0.069	0.021	14.29	300
400	2.50	24	0.060	0.017	16.67	400
500	2.00	27	0.054	0.014	18.52	500
600	1.66	30	0.050	0.012	20.00	600
700	1.42	33	0.047	0.011	21.21	700
825	1.19	38	0.045	0.010	21.71	825
1000	1.00	42	0.042	0.009	23.81	1000

The performance difference between NSP and SP modes for MFCD is graphically presented in Figures 6.6, 6.7, 6.8, and 6.9 as follows:

i. **Frequency v/s Average Power**

ii. **Frequency v/s Energy per Transaction**

iii. **Frequency v/s Latency**

iv. **Frequency v/s Maximum Bandwidth**

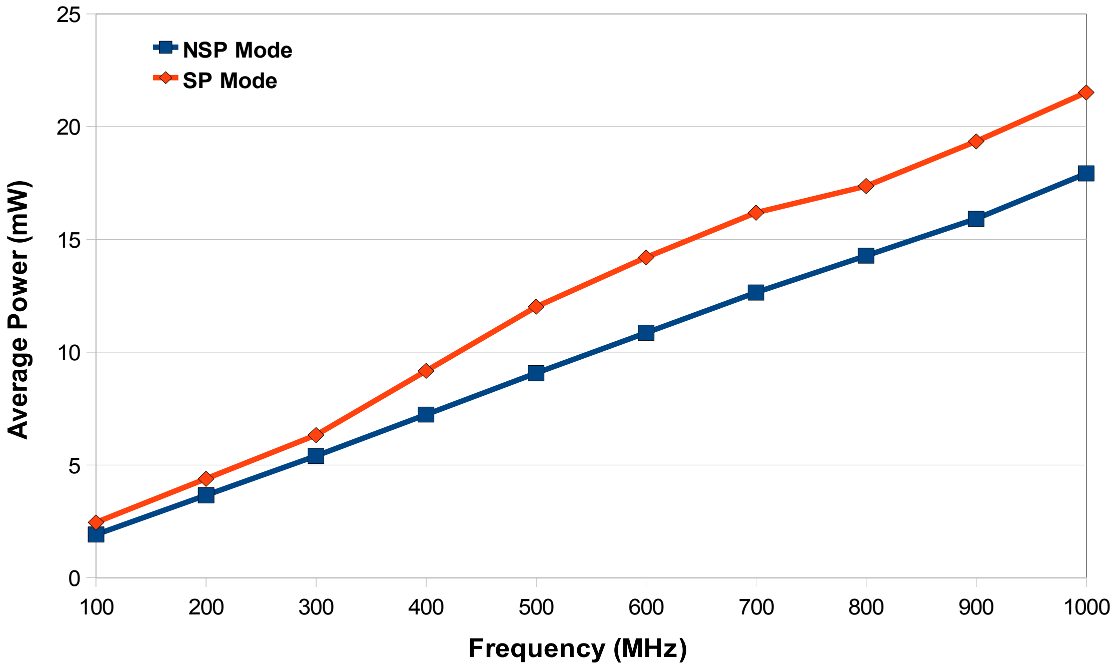


Figure 6.2. Performance Comparison Between NSP and SP Operating Modes in SFCD: Frequency v/s Average Power

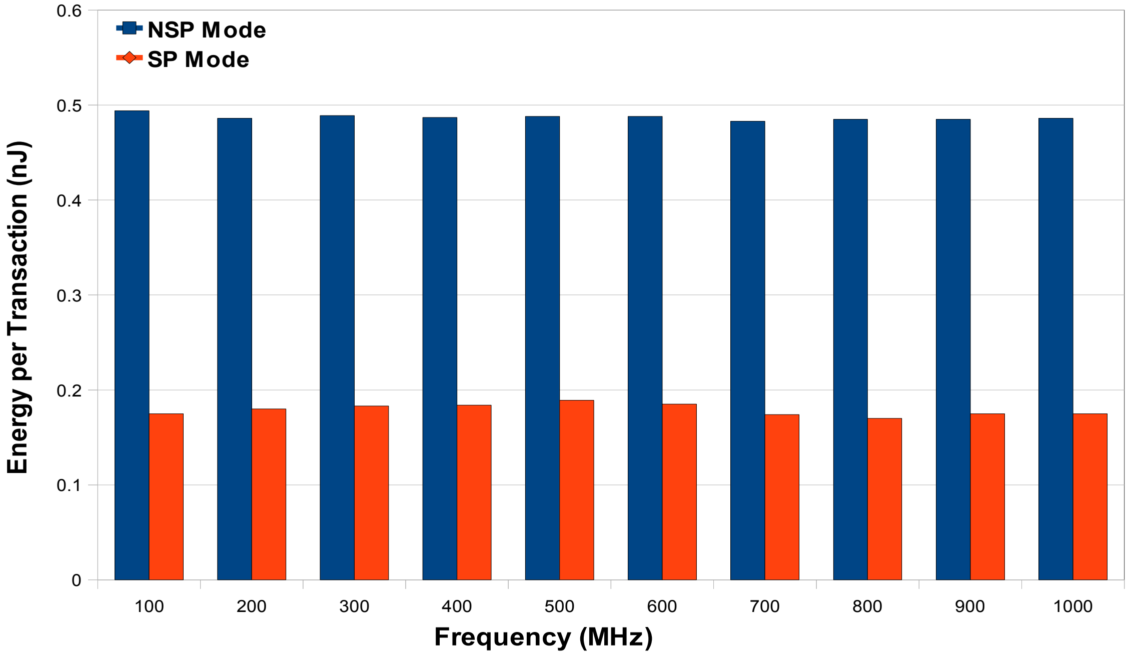


Figure 6.3. Performance Comparison Between NSP and SP Operating Modes in SFCD: Frequency v/s Energy per Transaction

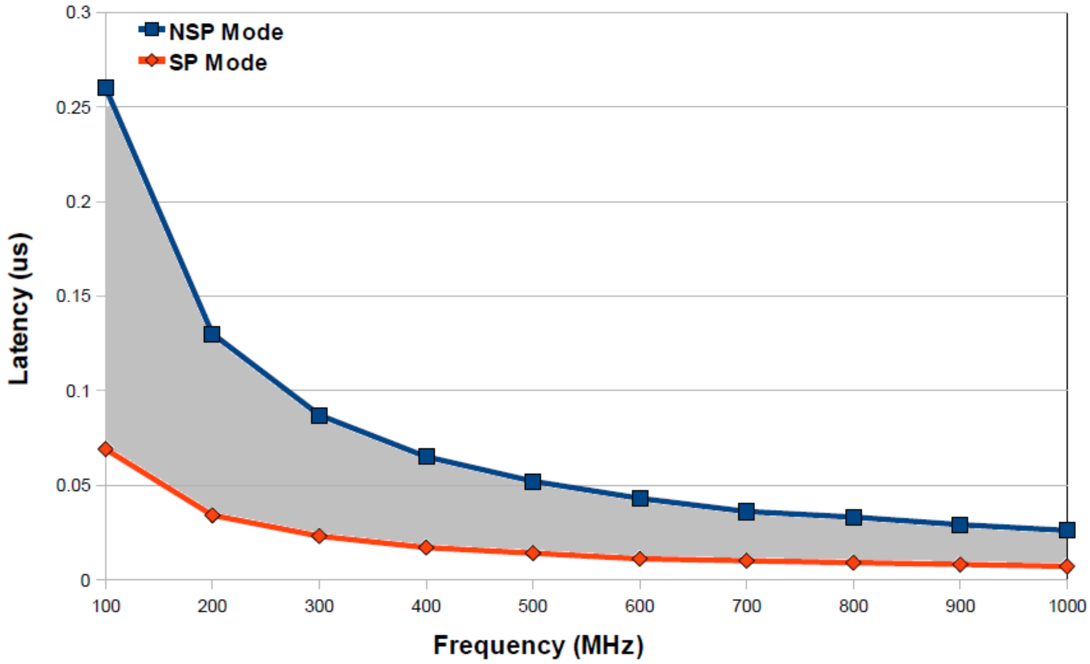


Figure 6.4. Performance Comparison Between NSP and SP Operating Modes in SFCD: Frequency v/s Latency

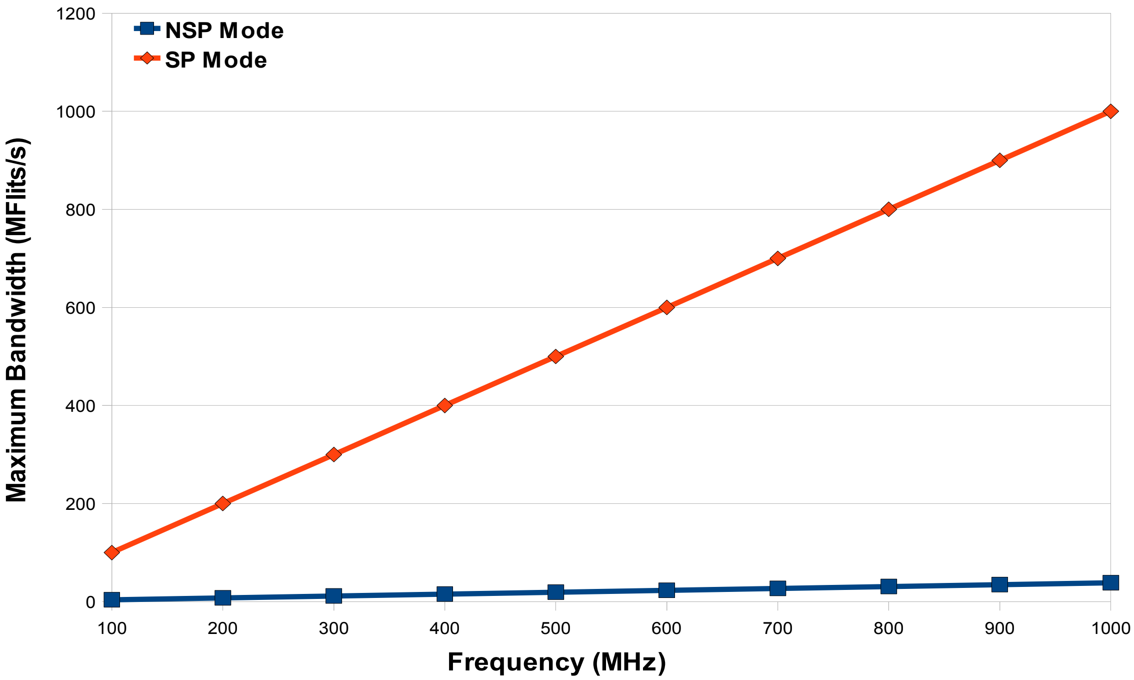


Figure 6.5. Performance Comparison Between NSP and SP Operating Modes in SFCD: Frequency v/s Maximum Bandwidth

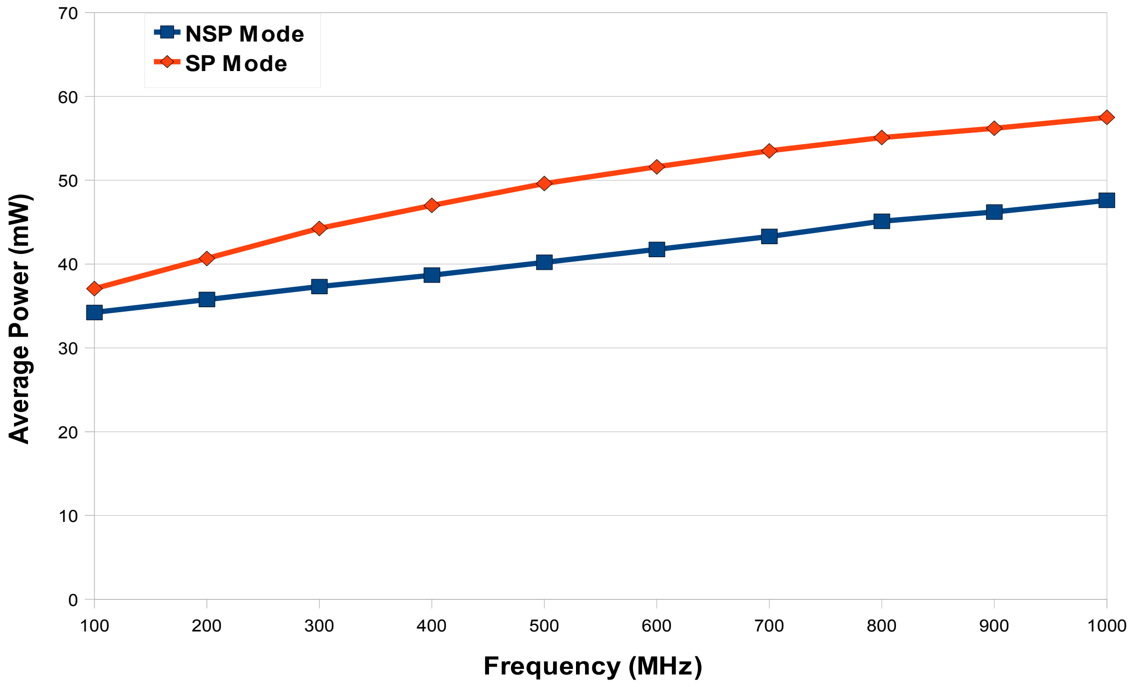


Figure 6.6. Performance Comparison Between NSP and SP Operating Modes in MFCD: Frequency v/s Average Power

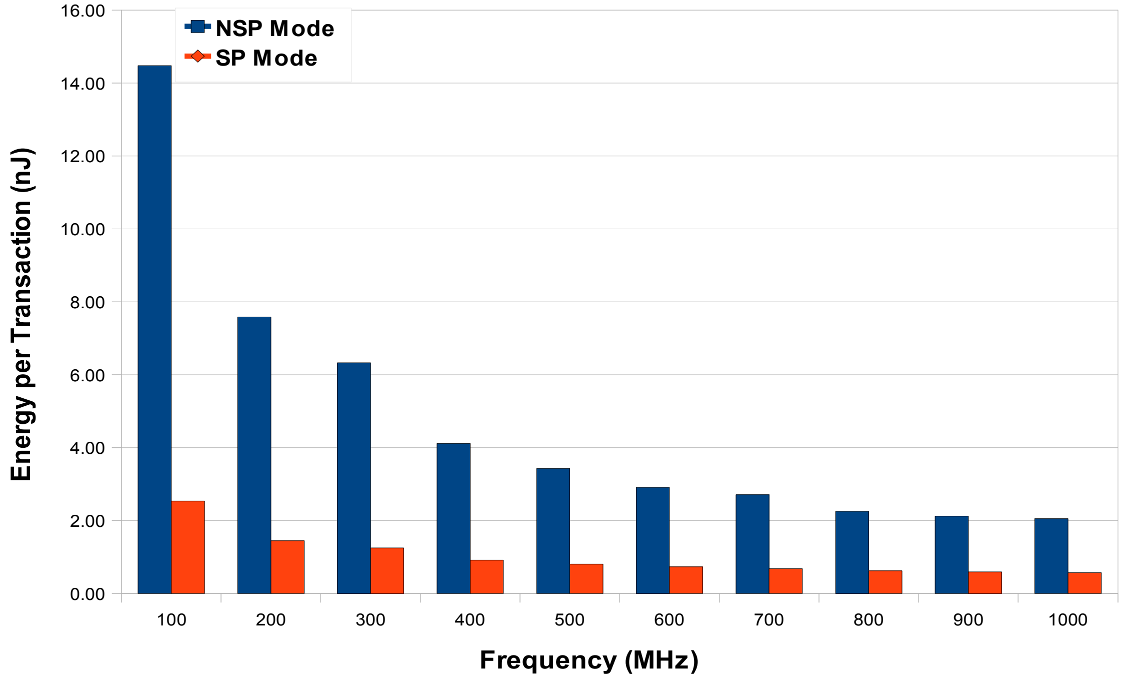


Figure 6.7. Performance Comparison Between NSP and SP Operating Modes in MFCD: Frequency v/s Energy per Transaction

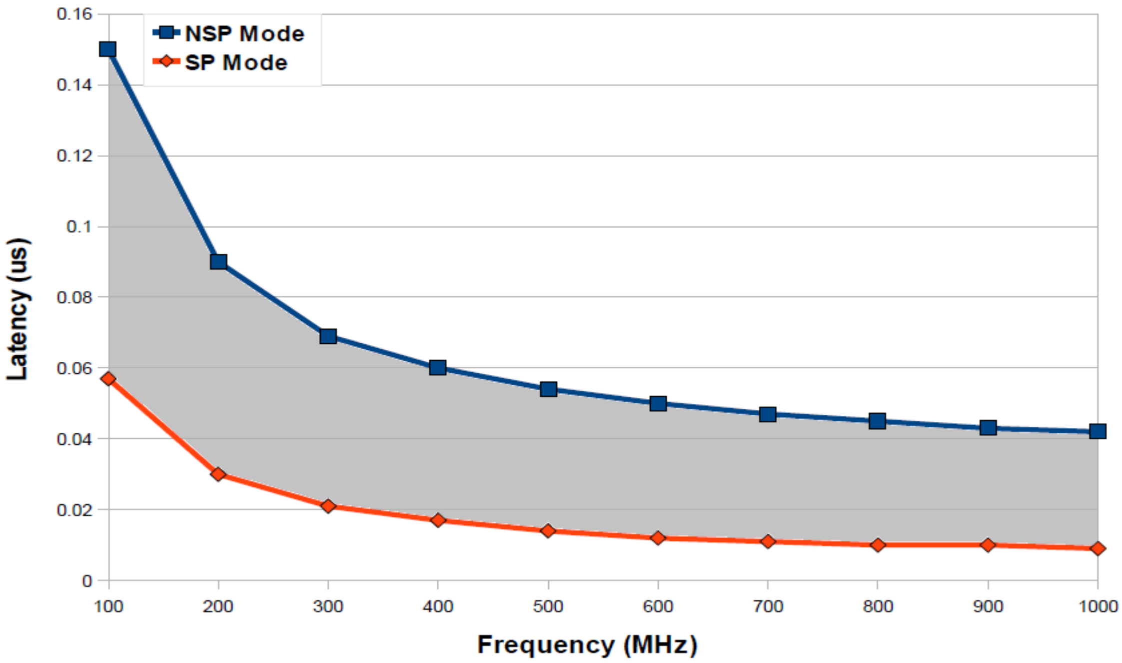


Figure 6.8. Performance Comparison Between NSP and SP Operating Modes in MFCD: Frequency v/s Latency

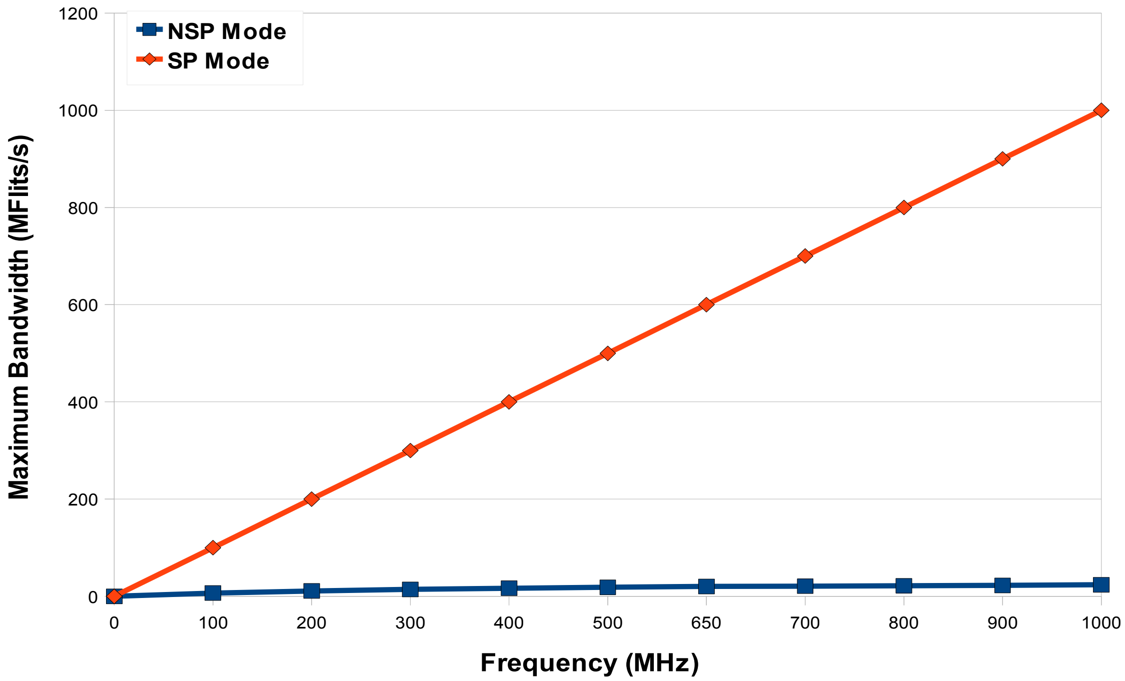
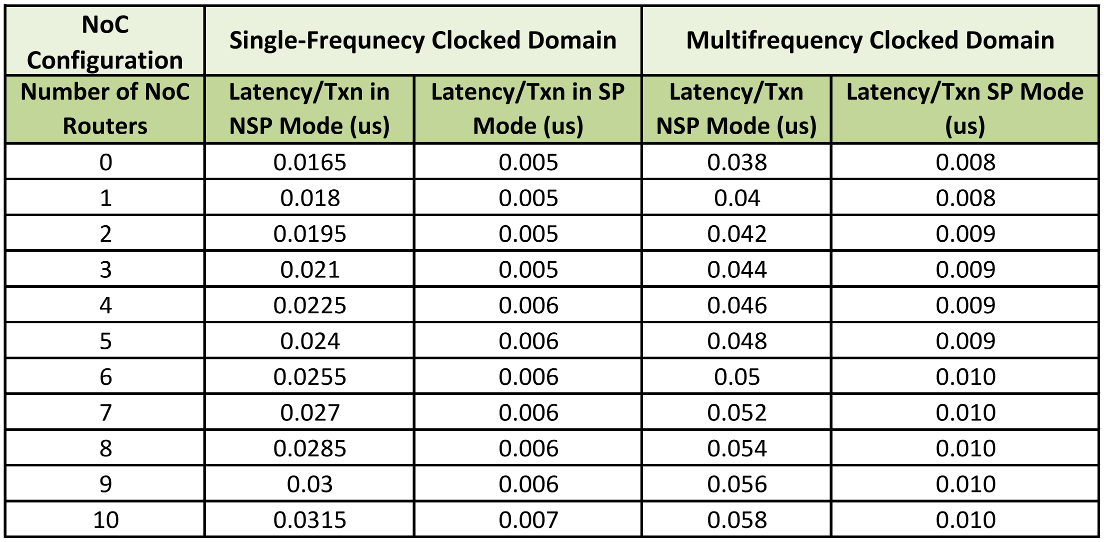


Figure 6.9. Performance Comparison Between NSP and SP Operating Modes in MFCD: Frequency v/s Maximum Bandwidth

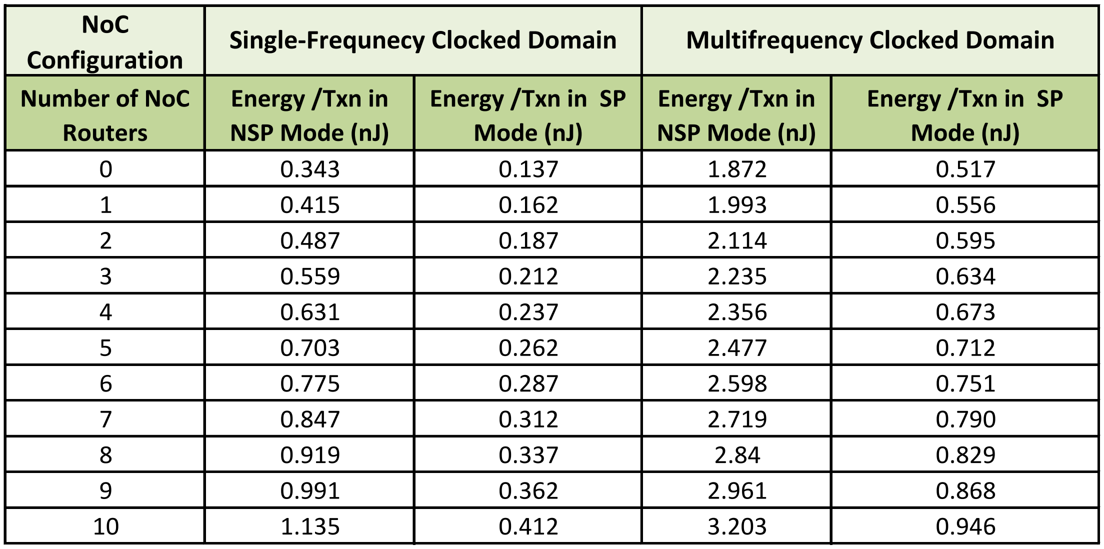
6.2.4.2 Network-On-Chip

Network-on-chip performance is evaluated by determining the NoC latency, average power, and energy per transaction under a varying number of routers. The results are obtained from clocked designs interfacing with 1-router and 2-router NoC cases, then projected to n router cases. The projected data are summarized in Tables 6.17, 6.18, 6.19, and 6.20. The summarized results for n-router cases for SFCD and MFCD are graphically represented in Figures 6.10, 6.11, 6.10, 6.12, 6.13, 6.14, 6.15, 6.16, and 6.17 .

Table 6.17. NoC Evaluation: Latency, Energy per Transaction for Varying Number of Routers



NoC Configuration	Single-Frequncy Clocked Domain		Multifrequency Clocked Domain	
Number of NoC Routers	Latency/Txn in NSP Mode (us)	Latency/Txn in SP Mode (us)	Latency/Txn NSP Mode (us)	Latency/Txn SP Mode (us)
0	0.0165	0.005	0.038	0.008
1	0.018	0.005	0.04	0.008
2	0.0195	0.005	0.042	0.009
3	0.021	0.005	0.044	0.009
4	0.0225	0.006	0.046	0.009
5	0.024	0.006	0.048	0.009
6	0.0255	0.006	0.05	0.010
7	0.027	0.006	0.052	0.010
8	0.0285	0.006	0.054	0.010
9	0.03	0.006	0.056	0.010
10	0.0315	0.007	0.058	0.010



NoC Configuration	Single-Frequncy Clocked Domain		Multifrequency Clocked Domain	
Number of NoC Routers	Energy /Txn in NSP Mode (nJ)	Energy /Txn in SP Mode (nJ)	Energy /Txn in NSP Mode (nJ)	Energy /Txn in SP Mode (nJ)
0	0.343	0.137	1.872	0.517
1	0.415	0.162	1.993	0.556
2	0.487	0.187	2.114	0.595
3	0.559	0.212	2.235	0.634
4	0.631	0.237	2.356	0.673
5	0.703	0.262	2.477	0.712
6	0.775	0.287	2.598	0.751
7	0.847	0.312	2.719	0.790
8	0.919	0.337	2.84	0.829
9	0.991	0.362	2.961	0.868
10	1.135	0.412	3.203	0.946

Table 6.18. NoC Evaluation: NSP Mode Average Power for Varying Number of Routers

Single-Frequency Clocked Domain: NSP Mode					
NoC Routers	**Net Switching Power (mW)**	**Cell Internal Power (mW)**	**Cell Leakage Power (mW)**	**Total Average Power (mW)**	**Clock Network Power (mW)**
0	3.00	17.69	0.04	20.73	19.69
1	3.80	18.50	0.05	22.35	21.25
2	4.61	19.30	0.07	23.98	22.89
3	5.41	20.11	0.09	25.61	24.33
4	6.22	20.91	0.10	27.23	25.87
5	7.02	21.72	0.12	28.86	27.42
6	7.83	22.52	0.14	30.49	28.96
7	8.63	23.33	0.15	32.11	30.51
8	9.44	24.13	0.17	33.74	32.05
9	10.24	24.94	0.18	35.36	33.60
10	11.05	25.74	0.20	36.99	35.14

Multifrequency Clocked Domain: NSP Mode					
NoC Routers	**Net Switching Power (mW)**	**Cell Internal Power (mW)**	**Cell Leakage Power (mW)**	**Total Average Power (mW)**	**Clock Network Power (mW)**
0	6.27	36.72	0.03	43.02	42.16
1	6.98	37.59	0.03	44.61	43.73
2	7.69	38.47	0.04	46.20	45.30
3	8.40	39.35	0.04	47.79	46.83
4	9.11	40.23	0.04	49.38	48.39
5	9.82	41.11	0.04	50.97	49.95
6	10.53	41.98	0.04	52.56	51.50
7	11.24	42.86	0.05	54.14	53.06
8	11.95	43.74	0.05	55.73	54.62
9	12.65	44.62	0.05	57.32	56.18
10	13.36	45.50	0.05	58.91	57.73

6.2.5 Results Summary

The performance comparison between nonsplit and split modes are graphically represented over a frequency range in both single- and multifrequency clocked domains. With increase in frequency, the average power in nonsplit and split modes linearly increases, since frequency is directly proportional to power dissipation. In both clocked domains, the average power in split mode is higher compared to nonsplit mode due to more switching activity over a smaller time period. In this design, about 75-95% of the total power is dissipated in the clock network components, due

Table 6.19. NoC Evaluation: SP Mode Average Power for Varying Number of Routers

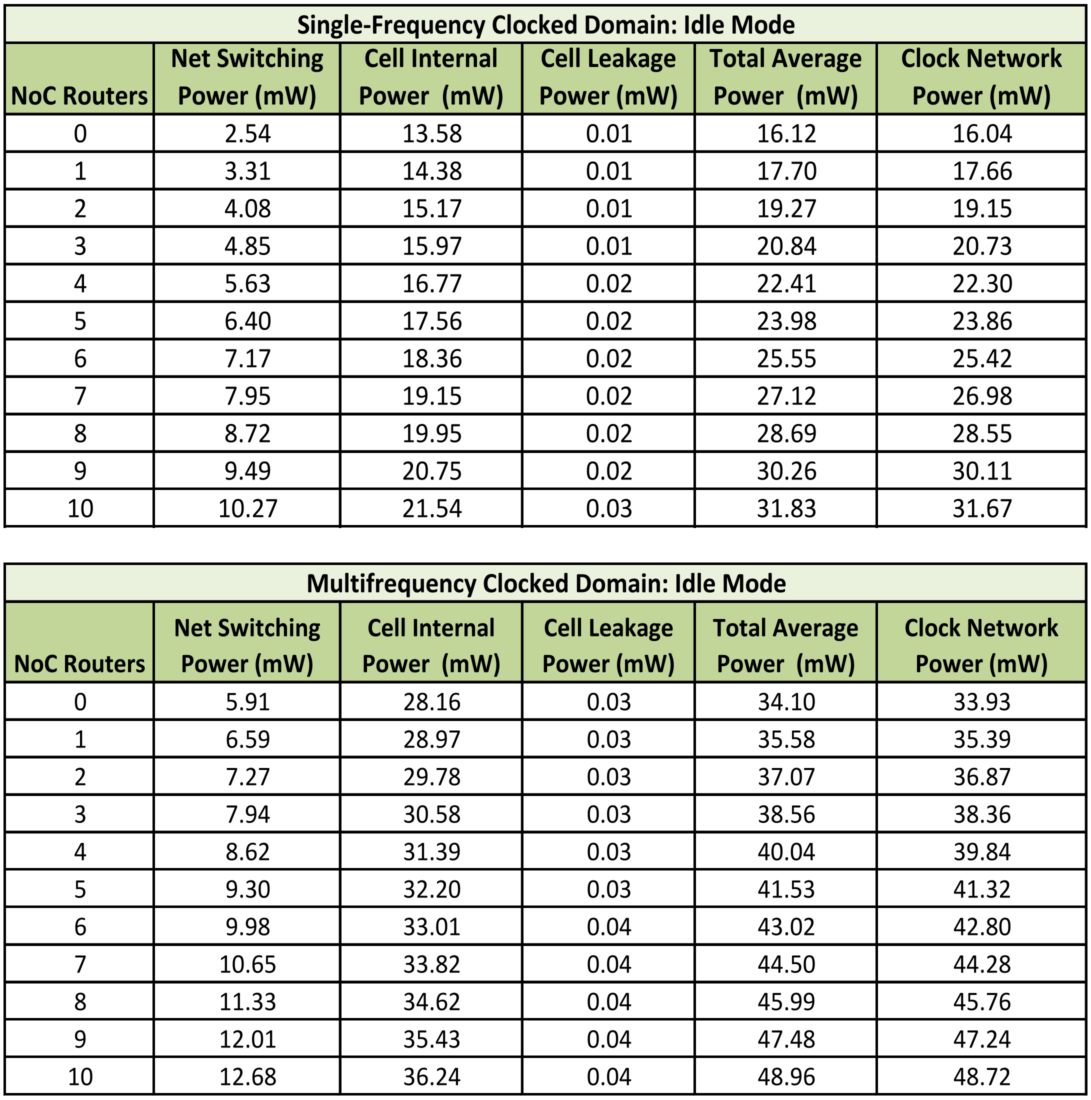
Single-Frequency Clocked Domain: SP Mode					
NoC Routers	**Net Switching Power (mW)**	**Cell Internal Power (mW)**	**Cell Leakage Power (mW)**	**Total Average Power (mW)**	**Clock Network Power (mW)**
0	3.85	20.63	0.01	24.49	20.02
1	5.58	22.29	0.01	27.88	22.80
2	7.31	23.95	0.01	31.27	25.52
3	9.04	25.61	0.01	34.66	28.34
4	10.77	27.27	0.01	38.05	31.12
5	12.51	28.92	0.01	41.44	33.89
6	14.24	30.58	0.02	44.84	36.66
7	15.97	32.24	0.02	48.23	39.44
8	17.70	33.90	0.02	51.62	42.21
9	19.43	35.56	0.02	55.01	44.98
10	21.16	37.22	0.02	58.40	47.76

Multifrequency Clocked Domain: SP Mode					
NoC Routers	**Net Switching Power (mW)**	**Cell Internal Power (mW)**	**Cell Leakage Power (mW)**	**Total Average Power (mW)**	**Clock Network Power (mW)**
0	10.13	41.77	0.03	51.93	42.58
1	11.19	42.95	0.03	54.18	44.11
2	12.25	44.14	0.04	56.43	46.36
3	13.31	45.32	0.05	58.68	48.11
4	14.37	46.50	0.05	60.93	49.96
5	15.43	47.69	0.06	63.18	51.81
6	16.49	48.87	0.07	65.43	53.65
7	17.55	50.05	0.07	67.68	55.50
8	18.61	51.24	0.08	69.93	57.34
9	19.68	52.42	0.09	72.18	59.19
10	20.74	53.60	0.09	74.43	61.03

to continuous driving of flops and registers even during idle clock cycles. In the case of multifrequency clocked domain, the total average power in both nonsplit and split operating modes is significantly higher compared to single-frequency clocked due to the addition of asynchronous clocked FIFOs in the DI modules. Also, in the multifrequency domain the, average power magnitude difference between both modes is significant compared to single-frequency clocked domain.

In the SFCD, the energy per transaction in split mode is less than nonsplit mode due to the longer run times. Also, in nonsplit mode, 90% of the total power is

Table 6.20. NoC Evaluation: Idle Mode Average Power for Varying Number of Routers



Single-Frequency Clocked Domain: Idle Mode					
NoC Routers	Net Switching Power (mW)	Cell Internal Power (mW)	Cell Leakage Power (mW)	Total Average Power (mW)	Clock Network Power (mW)
0	2.54	13.58	0.01	16.12	16.04
1	3.31	14.38	0.01	17.70	17.66
2	4.08	15.17	0.01	19.27	19.15
3	4.85	15.97	0.01	20.84	20.73
4	5.63	16.77	0.02	22.41	22.30
5	6.40	17.56	0.02	23.98	23.86
6	7.17	18.36	0.02	25.55	25.42
7	7.95	19.15	0.02	27.12	26.98
8	8.72	19.95	0.02	28.69	28.55
9	9.49	20.75	0.02	30.26	30.11
10	10.27	21.54	0.03	31.83	31.67

Multifrequency Clocked Domain: Idle Mode					
NoC Routers	Net Switching Power (mW)	Cell Internal Power (mW)	Cell Leakage Power (mW)	Total Average Power (mW)	Clock Network Power (mW)
0	5.91	28.16	0.03	34.10	33.93
1	6.59	28.97	0.03	35.58	35.39
2	7.27	29.78	0.03	37.07	36.87
3	7.94	30.58	0.03	38.56	38.36
4	8.62	31.39	0.03	40.04	39.84
5	9.30	32.20	0.03	41.53	41.32
6	9.98	33.01	0.04	43.02	42.80
7	10.65	33.82	0.04	44.50	44.28
8	11.33	34.62	0.04	45.99	45.76
9	12.01	35.43	0.04	47.48	47.24
10	12.68	36.24	0.04	48.96	48.72

dissipated driving flops during nonswitching activity (idle cycles). In the SFCD, the energy required per transaction in nonsplit and split modes does not change significantly over a frequency range. In the case of the MFCD tests, the Slave IP and NoC are operated at fixed target frequencies and the Master IP is varied over a frequency range. A decrease in energy required per flit is observed in nonsplit and split modes due to significant decrease run times and average power.

In both clocked domains, as expected, the overall latency (delay) decreases with an increase in frequency for nonsplit and split operating modes. Latency in the nonsplit

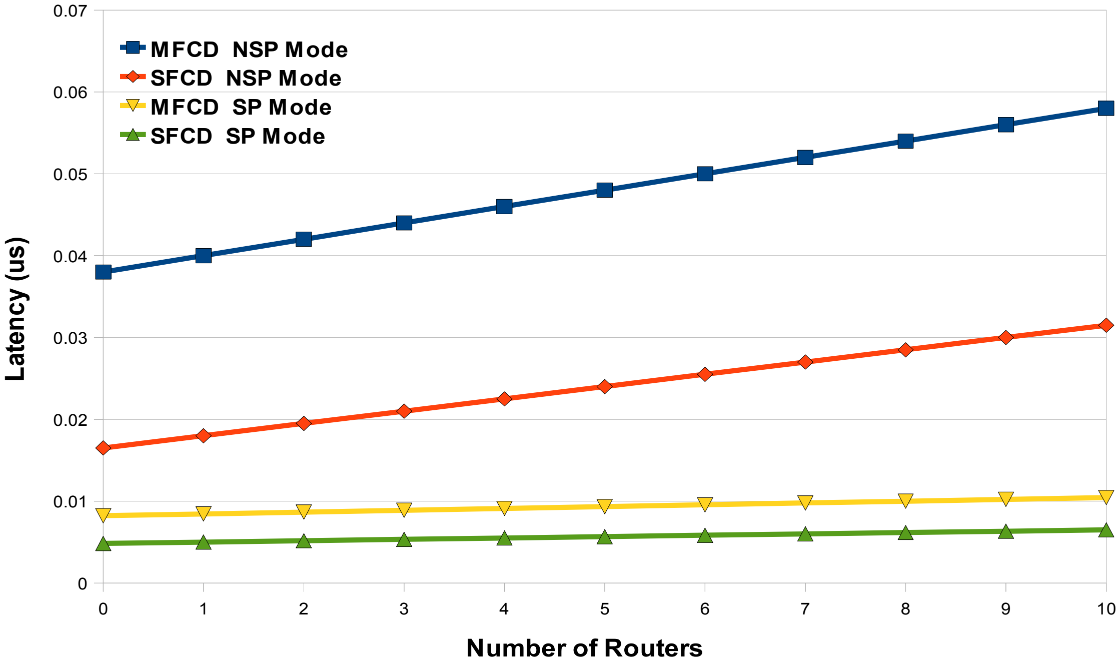


Figure 6.10. NoC Performance: Number of Routers v/s Latency

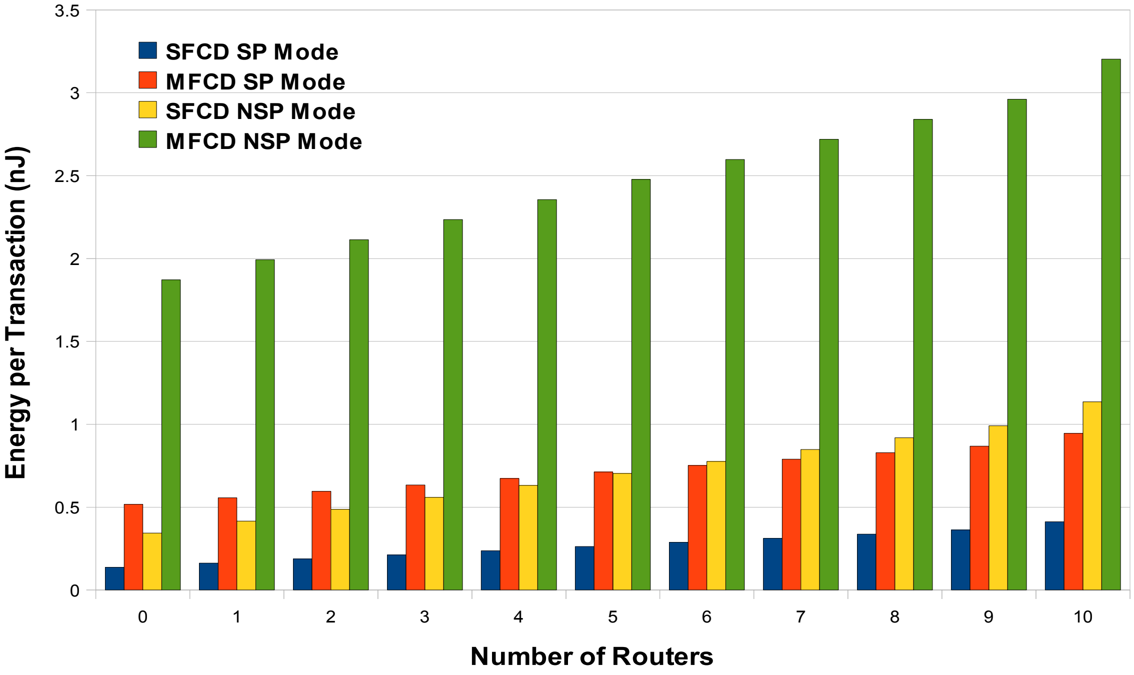


Figure 6.11. NoC Performance: Number of Routers v/s Energy per Transaction

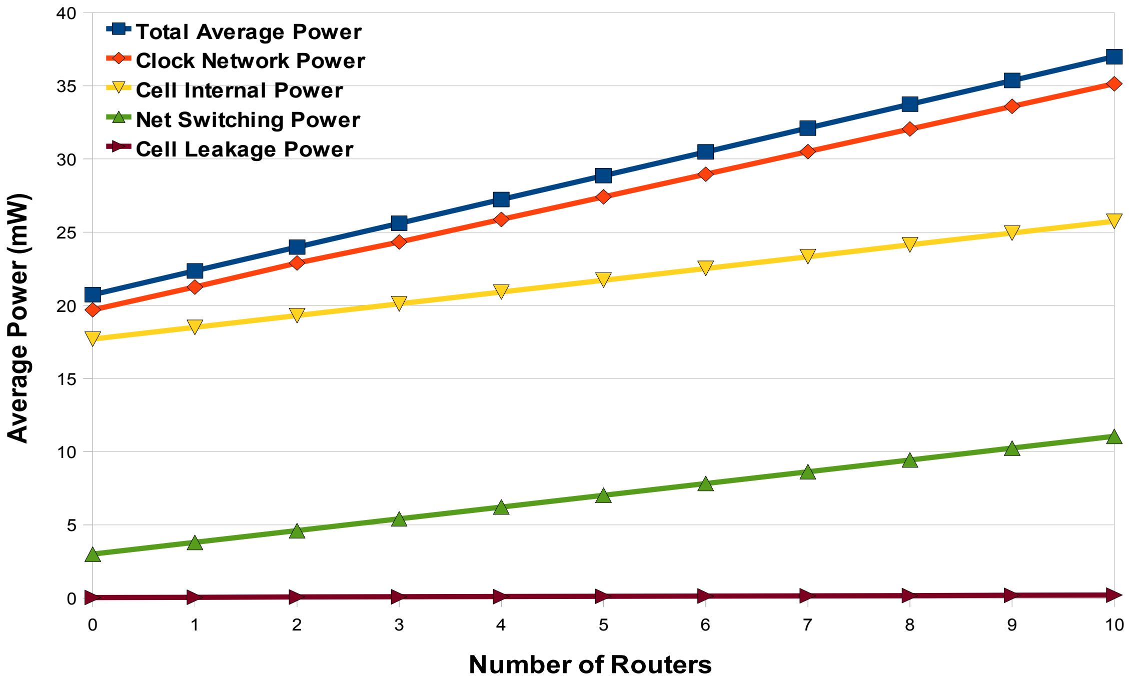


Figure 6.12. NoC Performance in SFCD: Frequency v/s NSP Mode Average Power

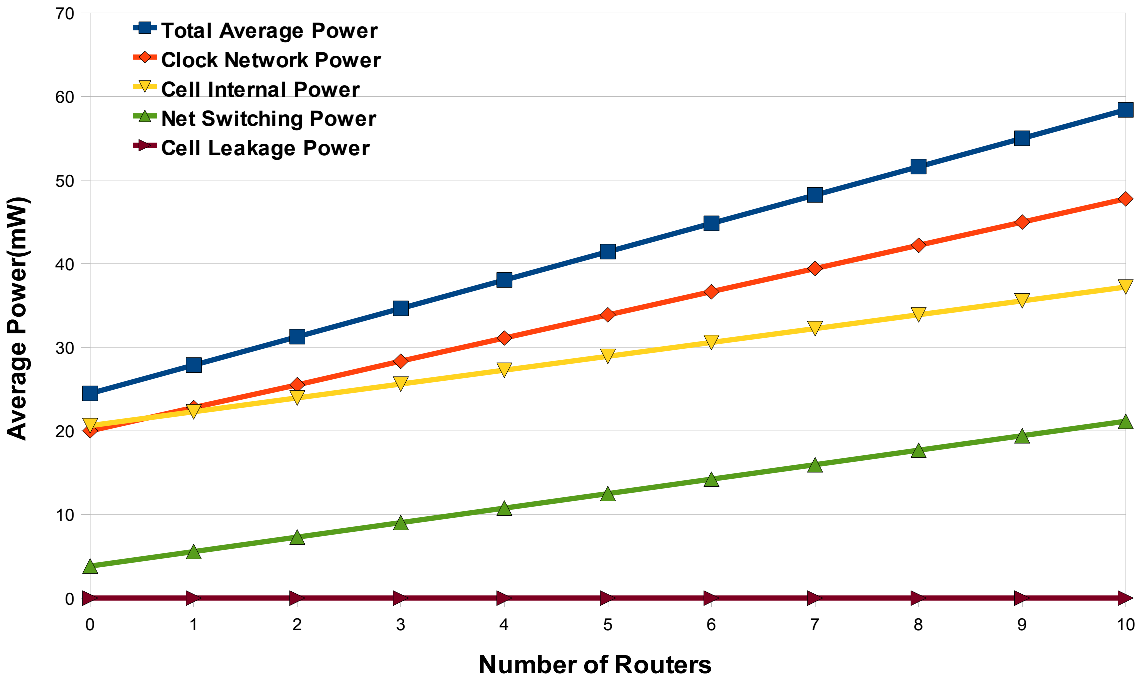


Figure 6.13. NoC Performance in SFCD: Frequency v/s SP Mode Average Power

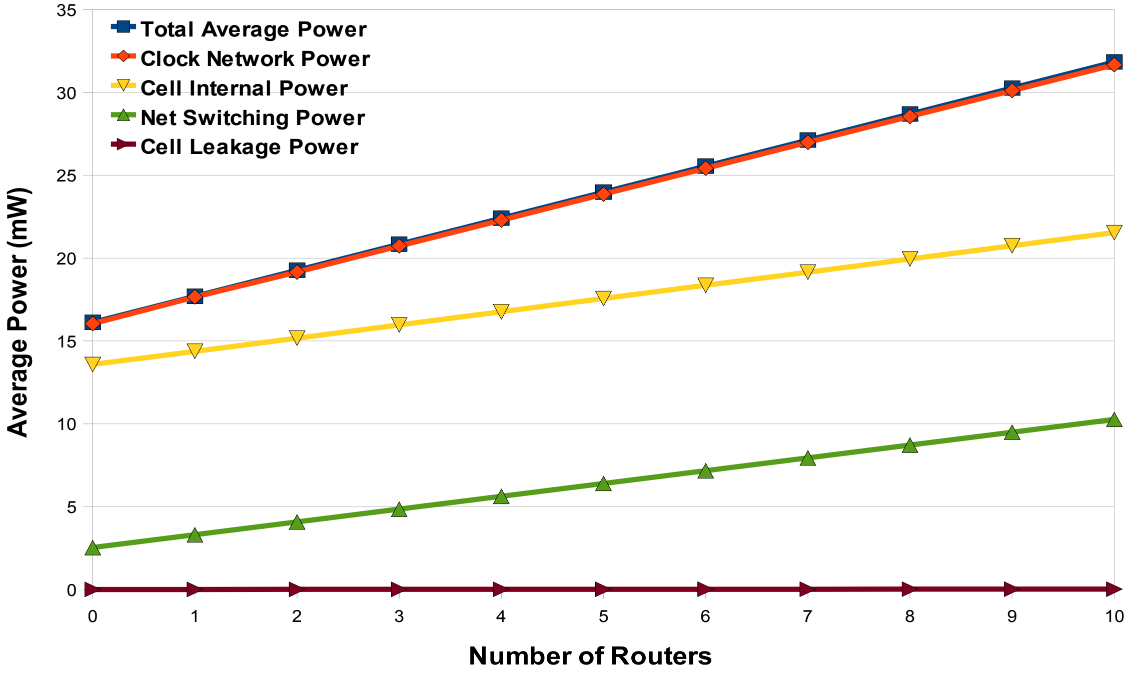


Figure 6.14. NoC Performance in SFCD: Frequency v/s Idle Mode Average Power

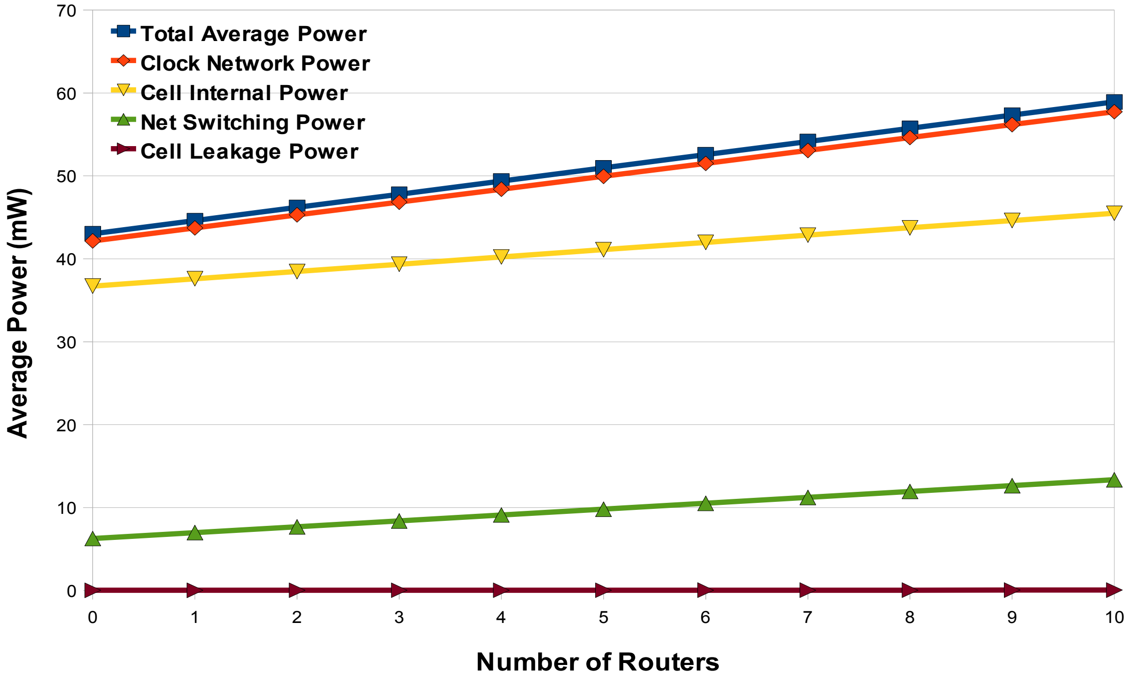


Figure 6.15. NoC Performance in MFCD: Frequency v/s NSP Mode Average Power

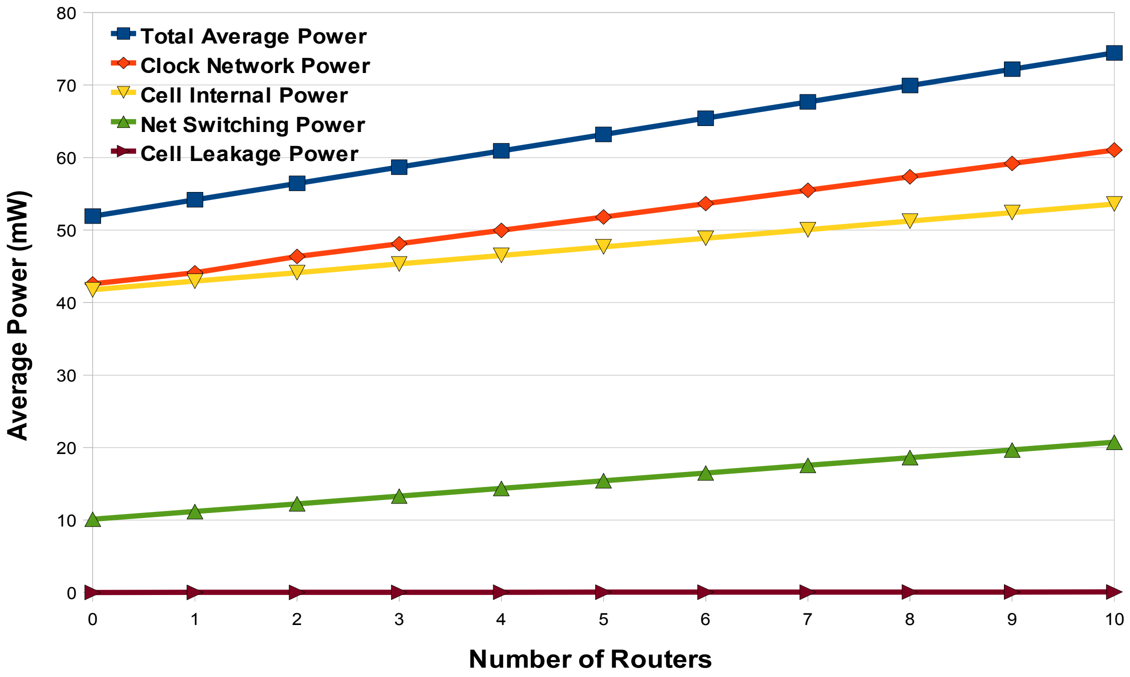


Figure 6.16. NoC Performance in MFCD: Frequency v/s SP Mode Average Power

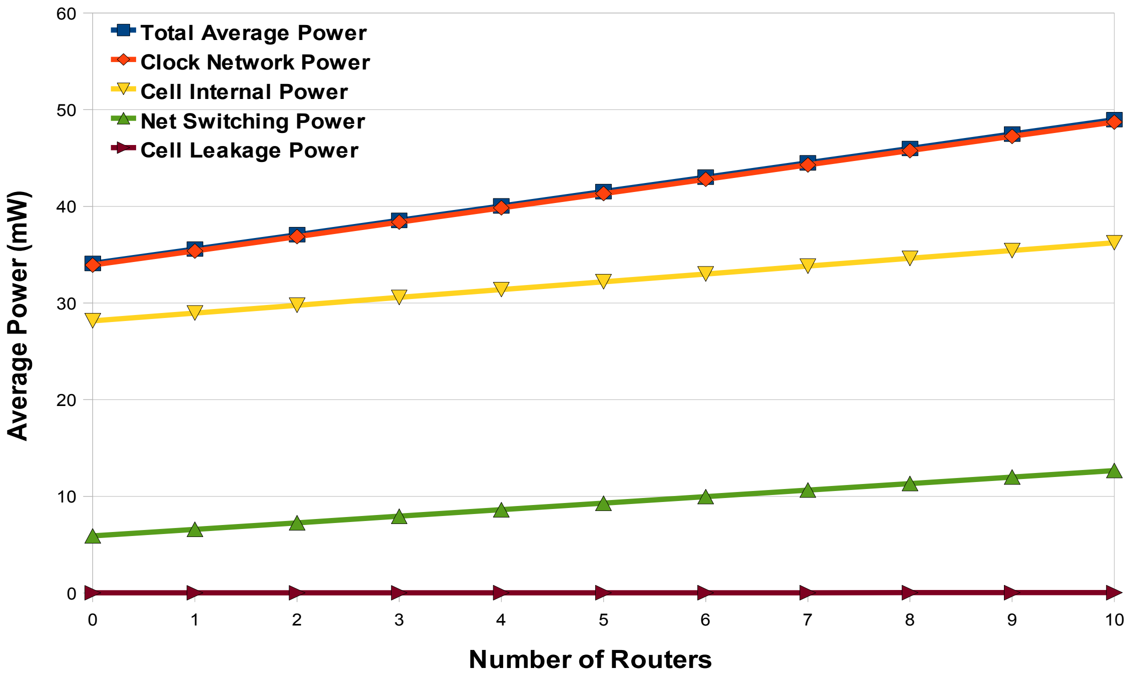


Figure 6.17. NoC Performance in MFCD: Frequency v/s Idle Mode Average Power

mode has a huge overhead since it requires more clock cycles for each request/response transaction completion compared to split mode. In the SFCD, a minimum number of clock cycles are required for each transaction over a frequency range. In the case of MFCD, with increase in Master IP clock frequency, the number of Master IP clock cycles cycles required for the same number of data flits increases (time period decreases) due to the interfacing of asynchronous clock domains.

The maximum bandwidth across the network increases with an increase in frequency for nonsplit and split operating modes. Network bandwidth is defined as the number of packets delivered per second. The maximum bandwidth in case of nonsplit mode is dependent on the number of cycles required for each transaction (target frequency/number of cycles) whereas in case of split mode, the network can maximally deliver packets at the design target clock frequency. Therefore, the split mode can always support higher bandwidths across the design compared to nonsplit mode.

Finally, from the performance evaluation results, it can be concluded that split operating mode with request/response pipelining support on each clock cycle is more efficient compared to nonsplit mode over a frequency range. Also, the split operating mode improves the overall system performance with the continuous data bursting feature.

6.2.5.1 NoC Performance

From the results, it can be concluded that the latency and energy per transaction when projected to n routers in case of split operating mode increases very marginally, but in case of nonsplit operating mode, both metrics are increased significantly. The total average power dissipation in both modes linearly increases with the number of routers, and at the same time, the clock network power is increased significantly (dominant component of the total power dissipated). Due to the employment of FIFOs in multifrequency clocked domain, the energy and power are higher when compared to single-frequency clocked domain.

CHAPTER 7

CONCLUSION AND FUTURE RESEARCH

7.1 Summary

This research work presents an efficient solution to address SoC contemporary design issues and shorter time-to-market requirements using an industry standard socket. OCP, an industry standard interface protocol, is employed in this research to build clocked NoC interfaces with a new design style. The built NoC interfaces enables the integration of different clocked IP cores and also simplifies design validation and the characterization of NoC power and performance. This research implementation includes building clocked interfaces in single- and multifrequency domains supporting different modes of operation, data bursting, and out of order response. The following items provides a brief research summary:

i. **Building Standard NoC Interfaces for IP Cores and Clocked On-chip Network Fabric Using OCP**
   - Defining OCP subset and customized interface signals depending on an IP core's/NoC communication requirements.
   - Design and implementation of customized back-end interfaces to an IP core/NoC and front-end interfaces (OCP Master and Slave entities).
   - Building a generic Domain Interface (DI) protocol and module for single- and multifrequency clocked domains.
   - Building asynchronous dual clocked FIFOs for the DI module to enable synchronization and data flow control.
   - Modularizing the interface components to improve reliability and reduce design time to interface different IP cores to the OCP socket.
   - Proper and efficient placement of buffering and synchronization schemes.

ii. **Design Implementation**

- RTL code development for clocked designs using Verilog HDL.
- Logic synthesis with IBM's 65nm process technology using Synopsys DC.
- Automatic Place and Route of clocked designs using Cadence SoC Encounter.
- UofU TCL scripts are modified to implement back-end steps of this design physical implementation.
- Functionality testing is done at each stage of the design flow meeting the proposed specifications. Synthesized and postlayout structural Verilog designs are validated with sdf back-annotation.
- Design simulations in the Modelsim simulator are automated with TCL scripts.

iii. **Design Testing and Performance Evaluation**

- Different Verilog HDL test benches are developed to verify the clocked designs in nonsplit and split operating modes.
- The performance metrics: target frequency of operation, latency, area, maximum bandwidth across a network, average power, and energy per transaction are determined in various design configurations.
- Designs are tested over a range of frequencies, and performance metrics are summarized and graphically presented.
- NoC performance is determined by configuring a different number of synchronous 3-port routers. The results obtained are projected to clocked designs with n routers to determine NoC performance for latency, average power, and energy per transaction.

iv. **Validating OCP Compliant Clocked Architectures**

The built clock designs have successfully met the proposed specifications. In the single-frequency clocked design, the IP cores and NoC (using 2-routers) can operate at a maximum frequency of 1.33 GHz. In the multifrequency clocked domain, the Master IP can be operated at a frequency of 1 GHz, the NoC at 1.11 GHz, and the Slave IP at 0.925 GHz. The multifrequency clocked domain could not achieve its maximum frequency of operation because of the increased latency in the request/response paths and synchronization overhead. However, at the expense of maximum operating frequency limitation, the multifrequency clocked domain enables a real-time system where different clocked IP cores can be integrated.

From the design performance evaluation, it can be concluded that split operating mode is more efficient in latency and energy per transaction compared to nonsplit operating mode at the expense of increased power dissipation. Designs implementing the split operating mode improve overall system performance with increased maximum bandwidths across the network and pipelined data bursting support. In case of nonsplit mode, it introduces huge latency overhead, and low bandwidth resources across network compared to split operating mode.

Both clocked designs compromise on power dissipation due to the continuous clock network power dissipation even during nonswitching activity phase. The clock network power component dominates the total dissipated power compared to combinatorial, sequential, and register logic components. In nonsplit operating mode, about 90-95% of the total dissipated power is contributed from the clock network component, since it drives the flops and registers constantly even during the idle clock cycles. In split operating mode with the increase in switching activity, the combinatorial power component and register logic component increases marginally compared to clock network power (75%), which overall dominates the total dissipated power.

With the above conclusions, it would be a good direction in the future research to develop power efficient industry standard NoC interfaces. The clock designs built as part of this research can be used as the base models for deriving elastic and asynchronous NoC interfaces.

7.2 Extensions and Future Research

The research has developed a base model with a new design approach paving way to explore potential contemporary SoC design research areas. Major extensions and research directions in this research area are as follows:

7.2.1 Extensions and Improvements to OCP Design

i. **Expand OCP Subset and Design Specifications**:

This project has implemented only the basic tagging and bursting data flow features of OCP. The other features of OCP, such as sideband and test signals, can be supported as extensions to this project. The complete list of OCP signals are provided in the Appendix. Tagging and bursting sizes, as well as address and data widths, can also be increased to support higher dataflow requirements.

ii. **Designing Efficient Clocked FIFOs for Synchronization and Buffering Schemes**:

The multifrequency clocked design implementation employed customized asynchronous dual clocked based pointer FIFOs for synchronization and buffering. The FIFOs have served the basic functionality but at the expense of a substantial latency overhead, area, and power dissipation. Improved asynchronous FIFOs may be able to provide more performance and power efficient designs.

iii. **Clock Gating to Reduce Clock Network Power**:

The clocked designs loose 90% of the total power to the clock network component due to the continuous driving of flops and registers during idle clock cycles. Power saving techniques, such as clock gating, can be employed in the synchronous circuits to save power. Clock gating adds extra logic to prune the clock tree activity by disabling portions of the circuitry during idle transactions.

iv. **Dynamic Packetization and Depacketization Techniques**:

A fixed 72-bit packet comprising address, data, and control bits are supported in the request and response path of this project implementation. Due to the fixed packet size, significant power and area are expended, since the 72-bits allocated to the response path are never fully used. Only 11-bits are used in the case of a write transaction, and 33-bits for a read transaction. Designs which can employ dynamic packetization techniques may provide power and area advantages.

v. **Designing Efficient On-chip Network Back-ends**:

The synchronous NoC can operate at a maximum frequency of 2.99 GHz, but the complexity in the NoC back-ends limits that frequency. More efficient or pipelined

NoC back-end interfaces can enable the maximum target network frequency and bandwidth.

vi. **Support Dual Flit Format at the NoC Back-ends**:
This project supports simple single flit packet and flit formats at the NoC back-ends. Double pumped or dual flit formats can be supported at the NoC back-ends to reduce overall latency and improve performance.

vii. **Better Power Modeling of Network to Include Wire Energy**:
In this design implementation, wire energy across the NoC was not considered. Modeling wire delay and energy cost will give a more accurate evaluation of a full design.

7.2.2 Future Research

i. The clocked designs built here serve as the starting point for research on different architectural clocking strategies and SoC architectures. The NoC interfaces that were developed can be used to explore research on different end-to-end NoC architectures. In the future, other architectural designs can provide a platform to determine the best clocking strategy depending on various performance metrics.

ii. **Deriving Elastic and Asynchronous Designs from Clocked Designs**:
Elastic and asynchronous designs can significantly boost power advantages over clocked designs. Designs without clock networks can provide more power efficient components. Desynchronization is a design methodology, which converts a synchronous gate level circuit into a more robust asynchronous one [30] [31]. Synchronous elasticization is one approach to transform an ordinary clocked design into a latency insensitive design or elastic design [32].

APPENDIX

OCP INTERFACE SIGNALS

Figures A.1 and A.2 list the complete OCP interface signals.

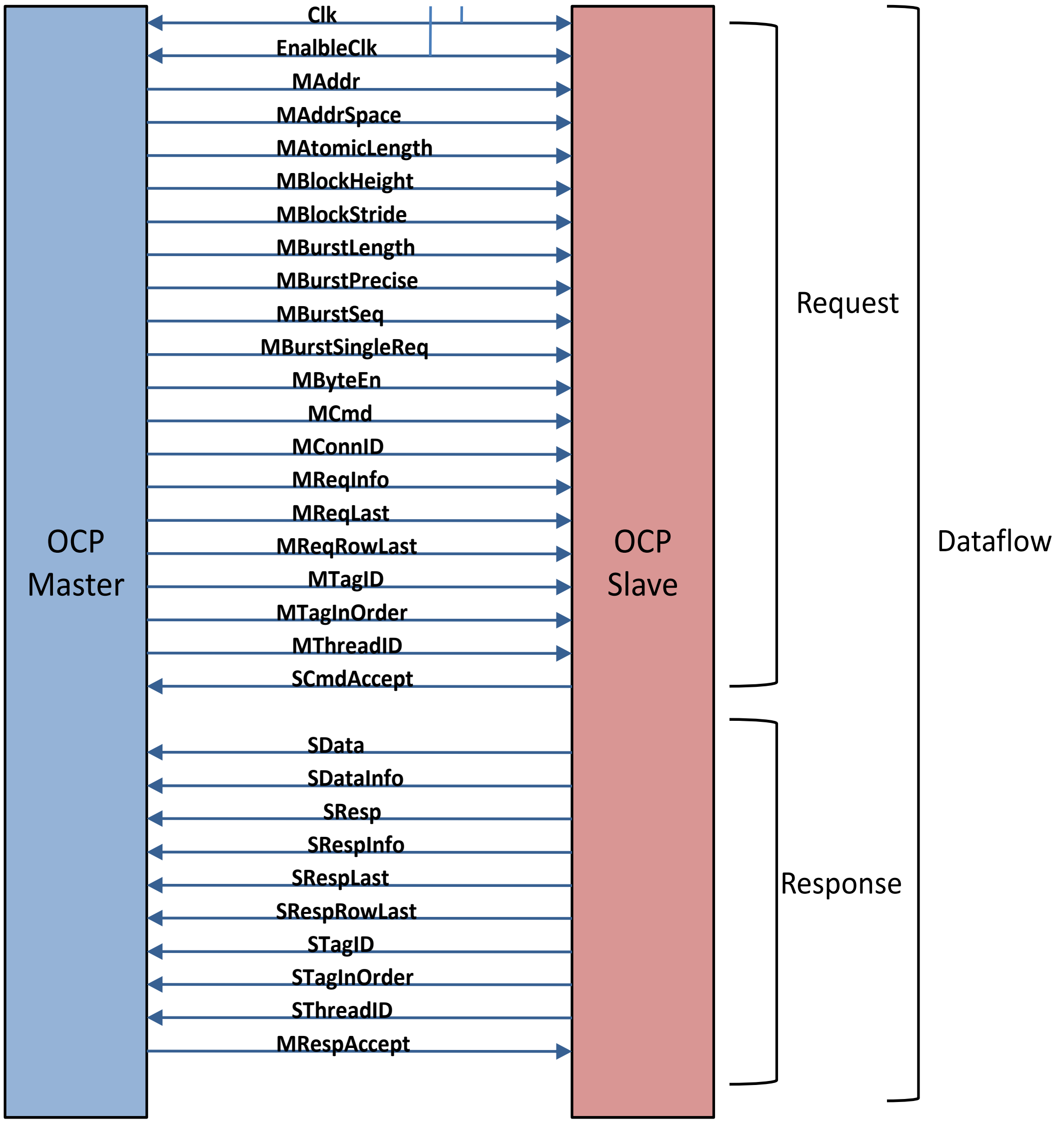


Figure A.1. OCP Dataflow Signals

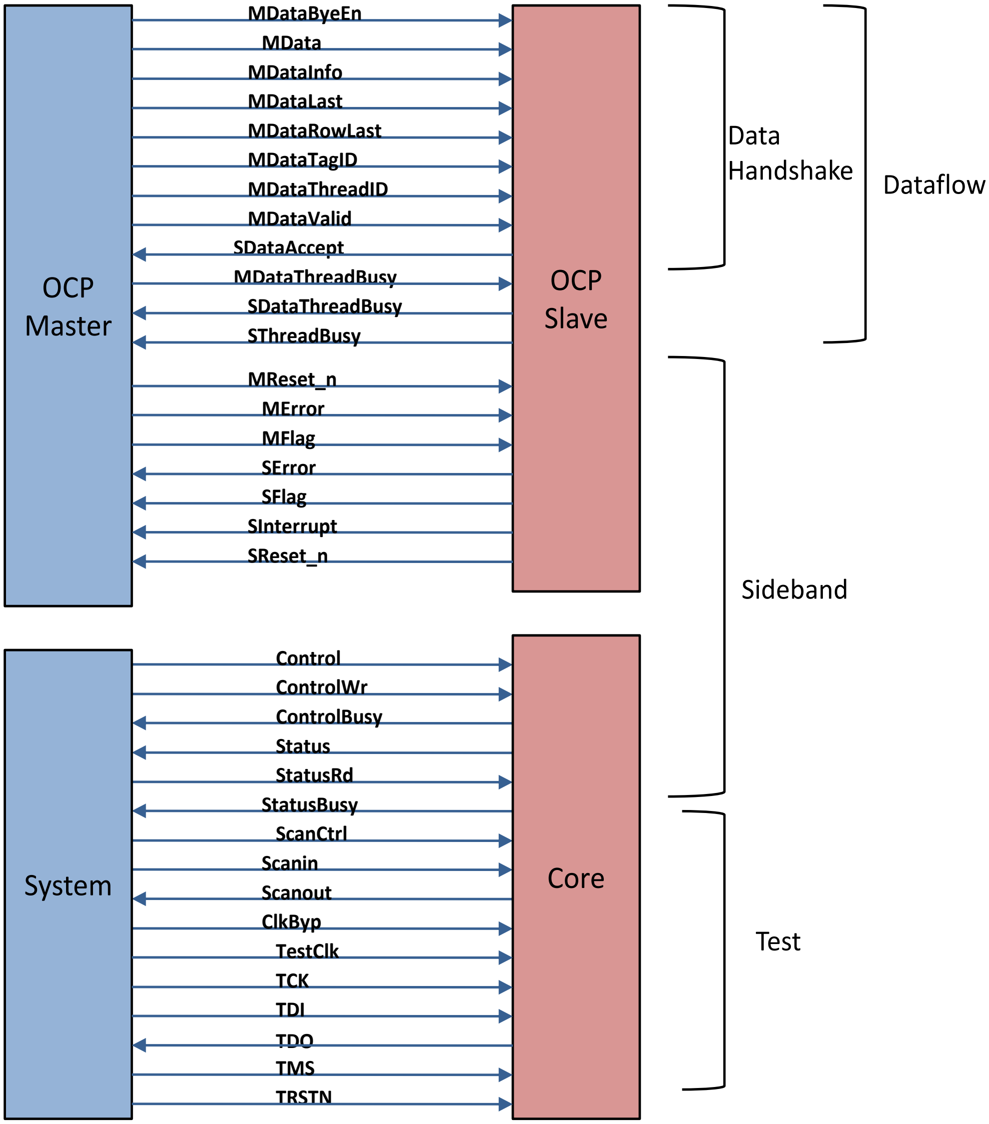


Figure A.2. OCP Sideband and Test Signals

REFERENCES

[1] OCP-IP, “The Importance of Sockets in SoC Design,” http://www.ocpip.org/white_papers.php.

[2] S. Parischa and N. Dutt, *On-Chip Communication Architectures: System on Chip Interconnect*. Georgia: Morgan Kaufmann, 2008.

[3] *Open Core Protocol Specification Ver 2.2*, http://www.ocpip.org/, Open Core Protocol - International Partnership (OCP-IP), 2008.

[4] C. Wang, C. Lai, S. Hwang, and Y. Lin, “On-Chip Interconnection Design and SoC Integration with OCP,” in *VLSI-DAT*. OCP-IP, 2008.

[5] Design and R. Incorp., “Bus Protocols Limits the Design Reuse of IP,” http://www.design-reuse.com/articles/.

[6] OCP-IP, “Socket-Centric IP Core Interface Maximizes IP Applications,” http://www.ocpip.org/white_papers.php.

[7] ARM, “AMBA Overview,” http://www.arm.com/, 2008.

[8] IBM, “CoreConnect Overview,” http://www.chips.ibm.com/products/coreconnect/, 2006.

[9] S. Corp, “Wishbone System-on-chip (soc) Interconnection Architecture for portable IP cores,” http://www.silicore.net/pdfiles/wishbone.pdf, 2002.

[10] R. Usselmann, “Open Cores SoC Bus Review,” http://www.opencores.org, Silicore Corp, Jan. 2001.

[11] *Technical Information on Open Core Protocol*, http://www.ocpip.org/, Open Core Protocol - International Partnership (OCP-IP).

[12] VSIA, “Virtual Component Interface Standard,” http://www.vsi.org/.

[13] E. Brunvand, *Digital VLSI Chip Design with Cadence and Synopsys CAD Tools*. Addison-Wesley, Jan. 2009.

[14] J. M. Rabaey, *Digital Integrated Circuits: A Design Perspective*, 2nd ed. Pearson Education, 2003.

[15] S. Palnitkar, *Verilog HDL : A Guide to Digital Design and Synthesis*. Upper Saddle River, New Jersey: Prentice Hall, Jan. 1996.

[16] *ModelSim SE Tutorial 6.6d*, http://www.model.com/, Mentor Graphics, 2010.

[17] *NC-Verilog Simulator Tutorial 5.1*, http://www.cadfamily.com/, Cadence, 2003.

[18] OpenCores, "Open Cores," http://http://opencores.org/projects/.

[19] J. You, D. Gebhardt, and K. S. Stevens, "Bandwidth Optimization in Asynchronous NoCs by Customizing Link Wire Length," in *International Conference on Computer Design*, Amsterdam, Oct. 2010.

[20] Synopsys, "PrimeTime PX: Methodology for Power Analysis," http://www.synopsys.com, Aug. 2006.

[21] "CoreCreator II Tool: IP Cores OCP Compliance Validation Environment," http://www.ocpip.org, OCP-IP, Aug. 2010.

[22] R.Ginosar, "Fourteen ways to fool your synchronizer," in *9th International Symposium on Asynchronous Circuits and Systems.* VLSI Systems Research Center, Haifa, Israel, May 2003.

[23] C. E. Cummings and I. Sunburst Design, "Simulation and Synthesis Techniques for Asynchronous FIFO Design," in *SNUG*, San Jose, Jun. 2002.

[24] G. D. Micheli and L. Benini, *Networks on Chips: Technology and Tools*, 1st ed., ser. Systems on Silicon series. Elsevier Science, Jul. 2006.

[25] C. E. Cummings and P. Alfke, "Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons," in *SNUG*, San Jose, Jun. 2002.

[26] J. You, Y. Xu, H. Han, and K. S. Stevens, "Performance Evaluation of Elastic GALS Interfaces and Network Fabric," in *Elsevier Electronic Notes in Theoretical Computer Science*, vol. 200, no. 1, Feb. 2008, pp. 17–32.

[27] T. T. Ye, L. Benini, and G. D. Micheli, "Packetization and Routing Analysis of On-Chip Multiprocessor Networks," in *Elsevier Science*, Sep. 2003.

[28] W. J. Bainbridge, "Asynchronous System-on-Chip Interconnect," Ph.D. dissertation, University of Manchester, Department of Computer Science, 2000.

[29] Sonics, "Sonics OCP Library for Verification 1.9," http://www.sonicsinc.com/, 2008.

[30] N. Andrikos, L. Lavagnot, D. Pandinit, and C. P. Sotiriou, "A Fully-Automated Desynchronization Flow for Synchronous Circuits," in *DAC 2007*, Jun. 2007.

[31] K. S. Stevens, Y. Xu, and V. Vij, "Characterization of Asynchronous Templates for Integration into Clocked CAD Flows," in *15th International Symposium on Asynchronous Circuits and Systems*, Washington, pp. 151–161.

[32] E. Kilada, S. Das, and K. Stevens, "Synchronous Elasticization: Considerations for Correct Implementation and MiniMIPS Case Study," in *VLSI-SOC*, Sep. 2010.