A

MAJOR PROJECT REPORT

ON

EFFICIENT VLSI DESIGN OF NETWORK PACKET SWITCHING UNIT FOR ETHERNET COMMUNICATION

Submitted in partial fulfilment of the requirement for the award of degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

SUBMITTED BY

RALLABANDI RAVITEJA	218R1A04H9
RAYUDU SAI KAMAL	218R1A04I0
SABAVAT RAHUL	218R1A04I1
SAMALA PRATHIK REDDY	218R1A04I2

Under the Esteemed Guidance of

Mrs. G. KALPANA

Associate Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Approved by AICTE, Affiliated to JNTU Hyderabad, Accredited by NBA)
Kandlakoya(V), Medchal (M), Telangana – 501401

(2024-2025)

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Approved by AICTE, Affiliated to JNTU Hyderabad, Accredited by NBA)

Kandlakoya(V), Medchal Road, Hyderabad – 501401

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the major-project work entitled **"EFFICIENT VLSI DESIGN OF NETWORK PACKET SWITCHING UNIT FOR ETHERNET COMMUNICATION"** is being submitted by **RALLABANDI RAVITEJA** bearing Roll No: **218R1A04H9, RAYUDU SAI KAMAL** bearing Roll No: **218R1A04I0, SABAVAT RAHUL** bearing Roll No**: 218R1A04I1, SAMALA PRATHIK REDDY bearing Roll No: 218R1A04I2** in B.Tech IV-II semester, Electronics and Communication Engineering is a record Bonafide work carried out during the academic year 2024-2025. The results embodied in this report have not been submitted to any other University for the award of any degree.

INTERNAL GUIDE:	**HEAD OF THE DEPARTMENT:**
Mrs. G. KALPANA	**Dr. SUMAN MISHRA**

EXTERNAL EXAMINER

ACKNOWLEDGEMENTS

We sincerely thank the management of our college **CMR Engineering Colleg**e for providing the required facilities during our project work. We derive great pleasure in expressing our sincere gratitude to our Principal **Dr. A. S. Reddy** for his timely suggestions, which helped us to complete the project work successfully. It is the very auspicious moment we would like to express our gratitude to **Dr. SUMAN MISHRA,** Head of the Department, ECE for his consistent encouragement during the progress of this project.

We take it as a privilege to thank our project coordinator **Dr. T. SATYANARAYANA,** Associate Professor, Department of ECE for the ideas that led to complete the project work and we also thank him for his continuous guidance, support, and unfailing patience, throughout the course of this work. We sincerely thank our project internal guide **Mr. S. SUDHAKAR,** Assistant Professor of ECE for guidance and encouragement in carrying out this project work.

DECLARATION

We hereby declare that the project entitled **"EFFICIENT VLSI DESIGN OF NETWORK PACKET SWITCHING UNIT FOR ETHERNET COMMUNICATION"** is the work done by us in campus at **CMR ENGINEERING COLLEGE,** Kandlakoya during the academic year 2024-2025 and is submitted as Major Project in partial fulfilment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY in ELECTRONICS AND COMMUNICATION ENGINEERING from JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSVERSITY, HYDERABAD .**

RALLABANDI RAVITEJA	**(218R1A04H9)**
RAYUDU SAI KAMAL	**(218R1A04I0)**
SABAVAT RAHUL	**(218R1A04I1)**
SAMALA PRATHIK REDDY	**(218R1A04I2)**

ABSTRACT

The rapid growth of Ethernet-based communication networks has necessitated the development of high-performance packet switching units to efficiently manage the flow of data packets. In this context, this research presents an innovative VLSI (Very Large Scale Integration) design of a Network Packet Switching Unit (NPSU) tailored for Ethernet communication.

The conventional systems employed in Ethernet networks often struggle to cope with the increasing demands for bandwidth, low latency, and reduced power consumption. These drawbacks include inefficient packet processing, high power consumption, and limited scalability. To address these issues, our proposed system leverages advanced VLSI techniques to optimize packet processing, reduce power consumption, and enhance scalability. The NPSU employs custom hardware accelerators, efficient memory architectures, and intelligent routing algorithms to maximize packet throughput while minimizing power consumption. Additionally, the proposed design facilitates easy integration into existing Ethernet network infrastructures.

This research represents a substantial step towards meeting the evolving demands of Ethernet-based communication systems, ensuring efficient and reliable data transfer in the face of ever-increasing data traffic. This project lays the foundation for future exploration in hardware acceleration for communication systems and encourages deeper research in integrating machine learning techniques for adaptive routing and power optimization in real-time. The proposed NPSU is thus a promising step toward the development of next-generation Ethernet switching solutions.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

On - Chip interchanges significantly affect the general zone, execution, and power utilization of present-day framework on-chips (SoCs). Expanding the correspondence over-head debases the speedup accomplished by parallel figuring as per Amdahl's law. In this manner, creating efficient superior on-chip interconnects has been of central significance for the parallel and elite figuring advances. Systems on-chips (NoCs) are the most versatile interconnection worldview that is equipped for tending to different application needs and meet distinctive performance prerequisites of substantial remaining tasks at hand, including inertness by means of versatile directing, throughput by means of enhanced way jumper sity, control dispersal by streamlining the NoC to focused outstanding tasks at hand, and adaptability by run-time design.

In NoCs, information is dealt with as parcels, while on-chip handling components (PEs) are considered as system hubs between associated by means of switches and switches. NoCs give an adaptable assumption and vast asset overheads. The NoC layering model parts the exchange into four layers: 1) application; transport; 3) system; and 4) physical layers. A crossbar is the fundamental building square of the NoC physical layer. A crossbar switch is a common correspondence medium receiving a numerous entrance procedure to empower physical bundle trade. The fundamental asset sharing systems embraced by existing NoC crossbars are time-division various access (TDMA), where the physical connection is time shared between the interconnected PEs, and space-division different access (SDMA), where a committed connection is built up between each match of interconnected PEs. The physical layer of a NoC switch likewise contains buffering and capacity gadgets.

Code-division numerous entrance (CDMA) is another medium sharing method that use the code space to empower concurrent medium access. In CDMA channels, each transmit– get (TX-RX) combine is allocated an exceptional bipolar spreading code and information spread from all transmitters are summed in an added substance correspondence channel. The spreading codes in established CDMA frameworks are symmetrical—cross relationship be tween’s symmetrical codes is zero—which empowers the CDMA beneficiary to appropriately disentangle the got entirety through a correlator decoder. Traditional CDMA frameworks depend on Walsh– Hadamard symmetrical codes to empower medium sharing. CDMA has been proposed as an on-chip interconnect sharing method for both transport and NoC interconnect models. Numerous focal points of utilizing CDMA for on-chip interconnects incorporate lessened power utilization, settled correspondence idleness, and decreased framework complexity. A CDMA switch has less wiring many-sided quality than a SDMA crossbar and less intervention overhead than a TDMA switch, and in this manner gives a decent trade off both. Be that as it may, just essential highlights of the CDMA innovation have been investigated in the on-chip interconnect writing.

Over-burden CDMA is an outstanding medium access method conveyed in remote interchanges where the quantity of clients sharing the correspondence channel is helped by expanding the quantity of usable spreading codes to the detriment of expanding numerous entrance obstruction (MAI). The over-burden CDMA idea can be connected to on-chip interconnects to build the interconnect limit.

1.1 OBJECTIVE OF THE PROJECT

This paper focuses on further reducing the communication latency and power consumption of NoCs, because the communication latency of NoCs directly influences the data access latency in many-core systems, and the power consumption of NoCs accounts for a high ratio of the total power consumption of the whole chip. In this paper, we propose a novel hybrid scheme, in which a novel switching mechanism, called virtual circuit switching, is first introduced to intermingle with circuit switching and packet switching. In virtual circuit switching, virtual channels (VCs) are exploited to form several virtual CS (VCS) connections by storing the interconnect information in routers. Flits can directly traverse the router with only the ST stage. The main advantage of virtual circuit switching is that it can have the similar router pipeline with circuit switching and can have multiple VCS connections to share a common physical channel. To support the proposed hybrid scheme, one modified router architecture is implemented based on the baseline with a tolerable overhead, and the corresponding switching mechanism is presented in this paper. Based on virtual circuit switching, a path allocation algorithm is proposed to determine VCS connections and CS connections on a mesh connected NoC under a given network traffic, so that both communication latency and power consumption are optimized. A set of synthetic traffic workloads and real traffic workloads are exploited to evaluate the effectiveness of the proposed hybrid scheme. The experimental results show that our proposed hybrid scheme can efficiently reduce both the communication latency and power consumption. Virtual circuit switching is first introduced, and the modified router architecture and its corresponding switching mechanism are presented to support the proposed hybrid scheme. Based on virtual circuit switching, this paper proposes a path allocation algorithm to optimize both communication latency and power consumption. The effectiveness of the proposed hybrid scheme is demonstrated by comparing with the baseline packed switched NoC design using a set of synthetic and real traffic workloads.

1.2 PROBLEM STATEMENT

In comparison with NoC, circuit switching can significantly lower the communication latency and power consumption, because routing and arbitration are not needed once circuits are set up. Only the ST stage is required on the circuit-switched (CS) connection when a flit traverses a node. However, circuit switching lacks flexibility. If several communications compete for a common physical channel, circuits will be set up in turn. Then, the long setup time will decrease the overall NoC performance. To address the problems of packet switching and circuit switching, the hybrid scheme that combines packet switching and circuit switching is proposed. It not only can provide high flexibility for communications but also optimize latency of NoCs by establishing CS connections between communication pairs. It had been also demonstrated that establishing CS connections on the PS network can reduce communication power.

Moreover, before circuits have been established, packets are transmitted on PS connections to offset the long setup delay of circuits. However, the fact that CS connections are not allowed to share a common physical channel restricts the number of CS connections. If several packet transmissions will compete for a common physical channel, only one packet transmission can be executed in circuit switching and other packets must travel on PS connections. For the traffic with light congestion, most of communications can be addressed through circuit switching. However, for the traffic with heavy congestion, a very low ratio of CS connections to communications may be incurred, which limits the optimization of latency and power for NoCs.

CHAPTER 2

LITERATURE SURVEY

This section deals with the detailed analysis of related work, there is considerable research work is done in the field of MCSoC-NoC and MPSoC-NoC, respectively. Then, CONNECT was developed with resolving the problems presented in the conventional NoCs, respectively. This CONNECT consisting of flow control, network buffer sizing, router pipeline depth, link width, and topology width, respectively. This was standard NoC design for prototyping on FPGAs. This method resulted in better performance as compared to the conventional ASIC based prototypes. Then, the Stand for NoC was developed by the Stanford University, which is an open source NoC. This NoC utilized the concepts of parallel VCs and parallels switching allocations along with the crossbar switching and reduced the resource utilization performance as compared to conventional CONNECT, respectively. Most of the research works are developed from the inspiration of these two NoC architectures and the proposed method is developed by following the principle design rules of both Stanford-NoC and CONNECT, respectively.

The research works initially focused on generalized NoC architectures. Initially, open source NoC router architecture has been designed and which is widely used in the various applications. But this method is applicable for only small scale-based applications, and it is suffering with throughput issues, while implementing in very large-scale applications. Thus, it is necessary to implement a standard platform for validating the issues generated in various applications and Hard NoC is the one of the references NoC design and validation platform, respectively. Initially, various buffers less NoC designs were developed by introducing the VCs in the conventional NoCs. These designs were performed effectively under the less traffic scenario, as the traffic was increased these routers consuming the more power and also data was loss during the transmission. Then, in authors developed the VC based NoC to meet the requirements of standard routers and mitigate the problems of buffer-less routers. But VCs overlapping problem was generated again as the traffic was increased, and data was transferred to the improper destinations. Thus, in authors developed the request masking method to avoid the congestion generated in the VCs. This method is used to control the VC allocation based on the source and destination address requests. But, as multiple numbers of sources requesting the data, more number of error controlling priorities was generated in the NoC router, so this method is failed to handle the multi priority system, respectively. In authors developed Flit-Oriented Lightweight Cycle-accurate network Simulator (FOLCS), which is used to connect and verify the various chips, cores, and processors and it is reduced the hardware utilization.

Then, the several researchers are focused on the implementation of MCSoC-NoC. These NoCs are capable of executing the single program with high speed as multiple cores are used to share the various tasks of single core, respectively. In authors introduced the NoC for approximate communications and named it as approx Noc. This method utilized the data approximation and quality control methods for reducing the size of packets and improved the speed of router. The error-resilient mechanisms were available to reduce the errors, but still the faults need to be optimized further to improve the performance. Thus, in authors focused on implementation fault tolerant NoC (FTNoC), which comprises of two stage network reducing the faults generated in the NoCs. Here, fault tolerant mechanisms were used to identify the faulty cores through static genetic task mapping procedure. As the number of paths were increased, it is very difficult to identify the optimal path through this mechanism. Then, in authors introduced the concept of dynamic clustering based NoC (DCNoC). These clustering can used to introduce the parallelization of tasks. This NoC equally shares the tasks among the various cores, but this method is suffering with priority related issues. Thus, in specifically introduced the centralized priority management allocation mechanism for NoC by using the virtual-channel allocation. In authors introduced the hybrid reconfigurable wireless NoC (Honey Win), which is implemented by using mesh topology. In this NoC priority-based task sharing concept is introduced at the cost of high-power consumption. Then, Pro NoC was developed to resolve the problems related to the latency, which is implemented by the concepts of prototyping and validation of faults by using the virtual network, low latency routing-based VC, respectively. These conventional MCSoC–NoCs are suffering with the routing speed problems, fault coverage issues, and synchronization issues in the parallel tasks, respectively.

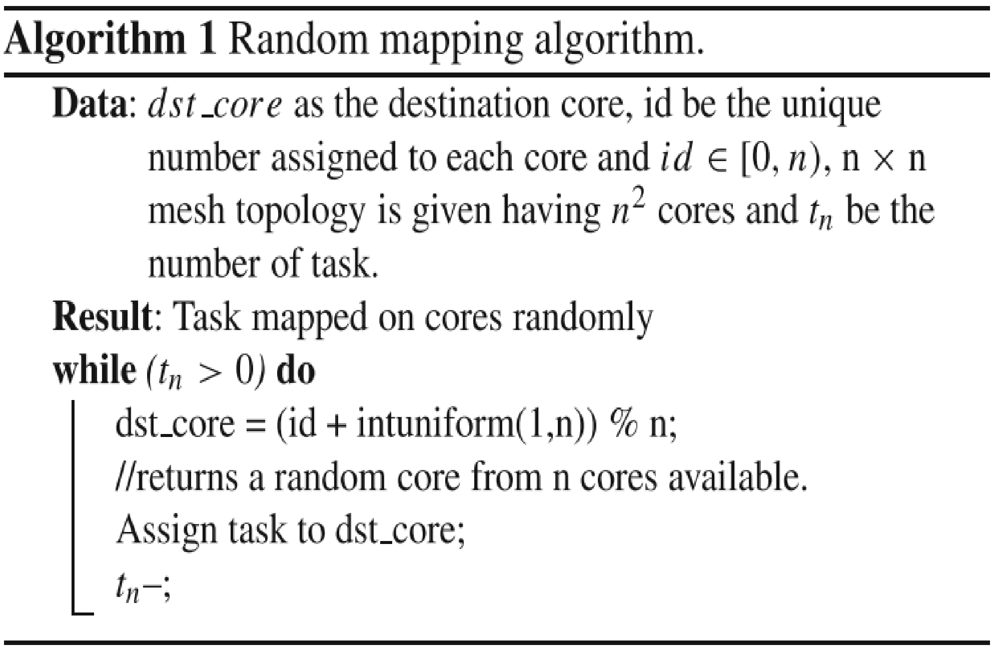
The researchers are started focusing on implementation of MPSoC–NoCs to solve the problems of conventional MCSOC–NoCs. These MPSoC–NoCs are more reliable as failure in one processor does not affect the routing functionality of another processor, respectively. Yet Another NoC (YaNoC) is developed for prototyping on FPGA devices, which is developed by using standard routing concepts on diagonal mesh topology. This method improved the speed as compared to conventional NoCs, but failed to reduce the hardware utilization. In authors focused on implementation of task mapping methods for heterogeneous MPSoCs, which is used to control the contention and energy of the system. This method finishes the tasks using earliest latest finish time first and latest finish time-based error controlling priorities, respectively. In authors developed the smart routing and secured protocol for MPSoC-NoC and named as SRSNoC, respectively. This smart routing makes the routes to non-overlapping each other; this method was used to improve the error controlling priorities of paths with higher security. In authors developed the path collision localization and the Denial-of-Service attack detection based NoC, which is named as DoS NoC, respectively. This NoC is specifically designed to reduce the packet loss ratio and the routing complexity is increased as the number of countermeasures to attack preventions was increased, respectively. In authors developed the LB NoC, which predicts the paths by using look ahead bypass manner. This method effectively reduces the area, power and latencies generated, but it requires additional routing algorithms for further improving the speed of router.

Thus, later on researchers started introducing the routing algorithms in both MCSoC and MPSoC-NoCs. But these routing algorithms resulted in better performance for MPSoC-NoCs as compared to MCSoC-NoCs, respectively. In authors introduced the round robin arbiter (RRA) for routing the packets effectively in the NoCs, which is implemented on the 4x4 mesh topology. Here, routing of the packets takes place through the priority-based crossbar switching. These error controlling priorities are perfectly assigned to the routes, if there is node to node synchronization is presented in the mesh topology. If the synchronization among the nodes is failed, then the error controlling priorities also failed. To overcome the problems of RRA, Adaptive hybrid arbiter (AHA) is developed for real time traffic controlling in NoC. The AHA unit develops the multi requests with low, medium and higher error controlling priorities, and then the routes are assigned with the error controlling priorities based on traffic load. The major problem of this work is error controlling priorities are introduced in the buffers, but most of the works are suggested buffer less designs can improve the throughput. In authors introduced the XY routing algorithm in NoC, which gives the superior performance as compared to the RRA, AHA and Odd-even (OE) routing, respectively. This XY can be utilized only in mesh topology effectively, it is failed to provide the optimal performance in different network conditions with various network topologies. In authors introduced the Iterative serial line internet protocol (iSLIP) for NoC. Here, the routes are assigned based on the dependency of the paths with respect to the head of line blocking and non-blocking limits, respectively. The iSLIP is consisting of cyclic redundancy check (CRC) with multiple parities. Here, the parities are not useful for assigning the multiple error controlling priorities in scheduling the tasks, respectively. In authors introduced the partition-based congestion control routing for NoCs and named as ParRouting. This method is used to control the local congestions, such as area is divided into multiple parts and each part is assigned with one unique priority. The major problem of this approach, this method is not appropriate for global congestions with multiple error controlling priorities. Thus, most of the conventional routing protocols are not considered the multiple priority scheme, thus this work is focused on implementation of multiple priority based iSLIP routing algorithm.

CHAPTER 3

EXISTING METHOD

There are many issues involved using random mapping algorithm, such as load balancing, latency, service time and queuing time are not handled by random algorithm for NoC. The worst case of the algorithm is, when every time the same core is chosen for mapping the task. As all tasks are mapped on the same core, so, the new tasks to be mapped will remain in the queue and wait for an infinite period of time till the core is not ready to process the new task. Once the core is available task is mapped on the core. In the best case of random algorithm for mapping, the randomly chosen cores will have an equal probability to be chosen, and task will be mapped on to these cores uniformly. There are rare chances to obtain the best case of the random algorithm. Let us consider a scenario that every time the last core of the grid is chosen to map the tasks. If such a case exists then latency involved to map the tasks on the cores will be very high. So, mapping the task on to the cores in case of random algorithm consumes a large amount of latency, service time, queuing time and the energy consumption. To improve the performance of the mapping algorithm in this paper, the horological, rotational and divide and conquer mapping algorithms are proposed.



Algorithm 1 Random mapping algorithm.

```
Data: $dst\_core$ as the destination core, id be the unique
      number assigned to each core and $id \in [0, n)$, n × n
      mesh topology is given having $n^2$ cores and $t_n$ be the
      number of task.
Result: Task mapped on cores randomly
while ($t_n > 0$) do
    dst_core = (id + intuniform(1,n)) % n;
    //returns a random core from n cores available.
    Assign task to dst_core;
    $t_n$–;
```

CHAPTER 4

PROPOSED METHOD

4.1 BASIC PRINCIPLE

The basic principle of the proposed hybrid scheme is that VCs are exploited in virtual circuit switching to form several VCS connections and multiple VCS connections can share a common physical channel. In this hybrid scheme, VCS connections cooperate with PS connections and CS connections to transmit packets. It is shown in Fig. 1. Fig. 1(a) shows an example of traffic, in which physical channels (1, 2), (7, 11), and (8, 4) are shared by more than one communication, respectively. (x, y) denotes the physical channel from node x to node y. shows CS connections and PS connections after using the conventional hybrid scheme. A CS connection is configured by recording in each router which input port should be connected to which output port. It is composed of physical channels and routers. However, routers on a PS connection are configured during the (BW, RC, VA, and SA) stages when flits require passing through. A physical channel can be shared by one CS connection and multiple PS connections. Once flits on CS connections arrive at routers, crossbar switches are immediately configured so that the CS flits can bypass directly to the ST stage. When there is no CS flit, the corresponding ports of crossbar switches are released to PS connections. shows VCS, CS, and PS connections of the proposed hybrid scheme. A VCS connection comprises VCs and routers that have been configured by recording in each router which input VC should be connected to which downstream VC. Crossbar switches of routers are preconfigured during the SA stage before VCS flits require passing through. Because VCS connections are established over VCs, a physical channel can be shared by n VCS connections at most (n is equal to the VC number). Other communications competing for that physical channel must be executed in packet switching, such as the communication from node 8 to node 4. A CS connection is configured by recording in each router which input port should be connected to which output port. It is composed of physical channels and routers. A CS connection is configured by recording in each router which input port should be connected to which output port. It is composed of physical channels and routers. A CS connection is configured by recording in each router which input port should be connected to which output port.

It is composed of physical channels and routers. A CS connection is configured by recording in each router which input port should be connected to which output port. It is composed of physical channels and routers.

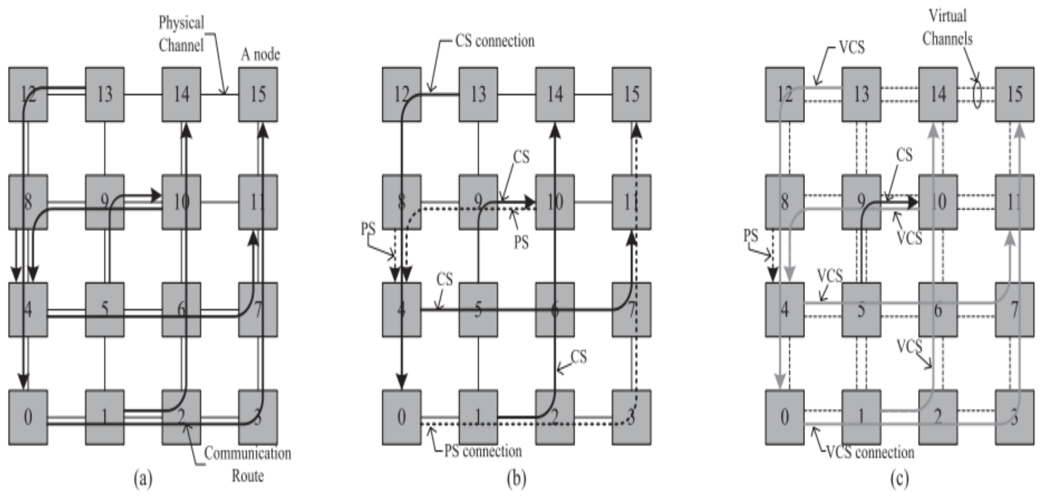


Fig 4.1 Illustration of the proposed hybrid scheme in a 4 × 4 mesh with two VCs per input port. (a) Simple traffic with communication routes in a 4 × 4 mesh. (b) CS and PS connections of the conventional hybrid scheme. (c) VCS, CS, and PS connections of the proposed hybrid scheme.

4.2 ROUTER ARCHITECTURE

In order to support VCS, PS, and CS connections at the same time, a modified router architecture with five ports is proposed, as shown in Fig. 2. Compared with the baseline router, the additional hardware of the proposed router includes the bypass path, the circuit configuration, and the VCS state. First, the bypass path is added in each input unit for allowing flits to go directly to the crossbar switch. Second, each input unit contains a PS state and a VCS state. The PS state corresponds to the VC state of the baseline PS router, and the VCS state is used to support VCS connections Third, the circuit configuration unit is to store the interconnect information for CS connections. In this paper, both the PS and the VCS states have n fields corresponding to n VCs. In addition, these n VCs are shared by VCS connections and PS connections. Information of the VC in the downstream router is stored in the VCS state to denote which downstream VC is connected to the corresponding VC. Incoming flits can directly traverse the crossbar switch according to the corresponding field of VCS state. The VCS signal is used to preconfigure the crossbar switch for VCS connections. It can be transmitted simultaneously with the transmission of flits. The VCS signal is (log2 n + 1)-bit wide, including a VC identifier and a flag for representing its validity. The VCS signal does not traverse the crossbar switch, but is generated by the router. It is output when the crossbar switch just completes the configuration for the VCS connection during the SA stage. The overhead caused by VCS signal can be negligible. First, the VCS signal is only issued when crossbar switches of the VCS connection wait to be preconfigured. Due to the low activity of VCS signal, the power overhead caused by VCS signal can be much less than the power saving by bypassing buffer writing, routing, and arbitration of routers. Second, in the network with two VCs, the width of VCS signal is 2 bits.

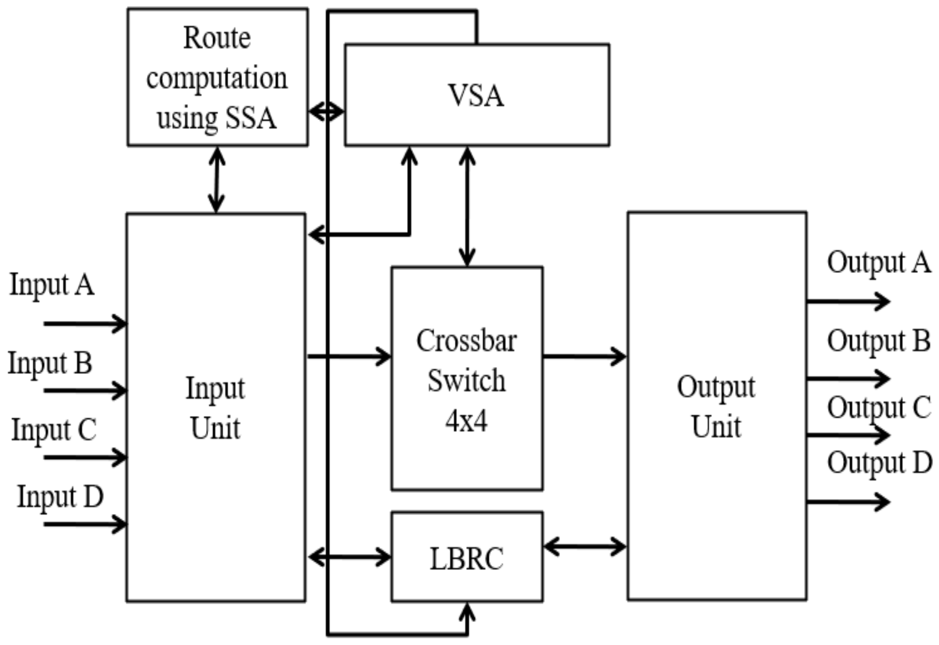


Fig 4.2 Architecture of the router

Compared with the 1-mm flit channel and the 128-bit router, the increased area is less than 1.5% under the estimation of Orion. Moreover, the proposed router architecture has been implemented in Verilog HDL. The synthesis result shows that the modified architecture increases the area by 1.34% without incurring any additional latency on the critical path, when compared with the baseline PS router. As for the data overhead, since route information can be stored in the VCS state and circuit configuration, packets on VCS and CS connections do not need route information. However, packets on PS connections need route information in the data of head flit.

4.3 SETUP NETWORK

In the proposed hybrid scheme, CS connections are constructed by setting circuit configuration units in routers and VCS connections are constructed by setting VCS states. A lightweight setup network which has a very small area and power overhead, is exploited to establish and tear down CS and VCS connections. Its main task is to set circuit configuration units and corresponding fields of VCS states in data network for storing the interconnect information. A CS or VCS connection is set up when the single-flit control message arrives at the destination node of the setup network. In comparison with the conventional hybrid scheme, VC allocator is still needed in the setup network. Besides the route information, the control flit in the setup network of the proposed hybrid scheme contains a VC identifier and one bit for informing the type of connection being constructed. However, these additional components only increase the NoC area by less than 1.5% (estimated by Orion when compared with the conventional hybrid scheme in a 4 × 4 mesh-connected NoC with 128-bit channels and two VCs per input port. Moreover, the power overhead of the setup network can be also much less than the power saving of the proposed hybrid scheme, because the traffic load of setup network is very low and the setup network is idle after constructing VCS/CS connections.

4.4 VIRTUAL CIRCUIT SWITCHING

The proposed hybrid scheme supports the intermingling of packet switching, circuit switching, and virtual circuit switching. Two extra bits are added to each flit (shown in Fig. 3) to denote the switching type of the flit. When a flit enters the router, these extra bits are checked at first. Then, the corresponding router pipeline is executed according to the switching type of the flit. Operations of circuit switching in this paper are similar. This section mainly describes operations of virtual circuit switching. Fig. 3 shows a detailed illustration of virtual circuit switching. Labels 1, 2, 3, and 4 represent the order of time step. Note that VCS connections must be constructed in advance before the flit traveling in virtual circuit switching.

1) Packet Transmission Without Blocking: Fig. 3(a) shows operations of virtual circuit switching without blocking. At first, a VCS signal is being input to the router at 1 time before the link traversal (LT) of the first flit. When the VCS signal arrives, SA is directly executed at 2 time since the information of route and VC allocation is already stored in the

VCS state. At this time, the first flit of the packet does not arrive and is being input to the router during the LT stage. Since the transmission is not blocked, the crossbar switch is successfully configured. A notification is given to the input unit for informing that the flits can go directly to the crossbar switch through the bypass path. The switch allocator also commands the corresponding output port of the router to issue a VCS signal to the downstream router. When the flit arrives at 3 time, it directly traverses the crossbar switch to the output port. In addition, the output VCS signal is transmitted directly to the downstream router, since it does not need to traverse the crossbar switch. From 4 time, the following flits also pass through the router with only the ST stage and the crossbar switch is reserved until the traversal of the tail flit or the cease of packet transmission. The VCS signal is not needed to configure the crossbar switch.

2) Packet Transmission With Blocking: Fig. 3(b) shows operations of virtual circuit switching when flits are blocked. In Fig. 3(b), the white flits lost the competition for the output port of the router and the crossbar switch is occupied by the dark flits. At 1 time, a VCS signal is being input before the LT of the first white flit. At 2 time, the switch allocator fails to allocate the crossbar switch to the white flits. A notification is given to the corresponding input unit for informing the blocking. In addition, the output VCS signal will not be issued. The corresponding field of the VCS state is set to mark the output of VCS signal for preconfiguring the remaining routers. From 3 time, when the first white flit arrives at the router, the white flits are stored in buffers and the transmission of white flits is ceased. Then, the following white flits are buffered in upstream routers controlled by backpressures.

3) Packet Recovering From Blocking: Fig. 3(c) shows operations of virtual circuit switching when packet recovering from blocking. The white flits are buffered since the conflicted output port is allocated to the dark flits. Once the dark flits are blocked or there is no dark flit being input, the crossbar switch will be allocated to the white flits. At this time, which is represented by label 1 , a dark flit is traversing the crossbar switch and the switch allocator successfully allocates the crossbar switch to the white flits. At 2 time, since the corresponding field of VCS state marks the output of VCS signal, a VCS signal is transmitted to the downstream router. It is to guarantee that the white flits can still directly traverse the crossbar switch when arriving at the downstream router. This VCS signal is output simultaneously with the LT of the dark flit. Meanwhile, the first white flit in buffers is traversing the crossbar switch. From 3 time, the following white flits in buffers continue to traverse the crossbar switch. In addition, the backpressure will call white flits in upstream routers to restart the transmission on the VCS connection. This VCS signal is output simultaneously with the LT of the dark flit. Meanwhile, the first white flit in buffers is traversing the crossbar switch.

It is to guarantee that the white flits can still directly traverse the crossbar switch when arriving at the downstream router. This VCS signal is output simultaneously with the LT of the dark flit. Meanwhile, the first white flit in buffers is traversing the crossbar switch.

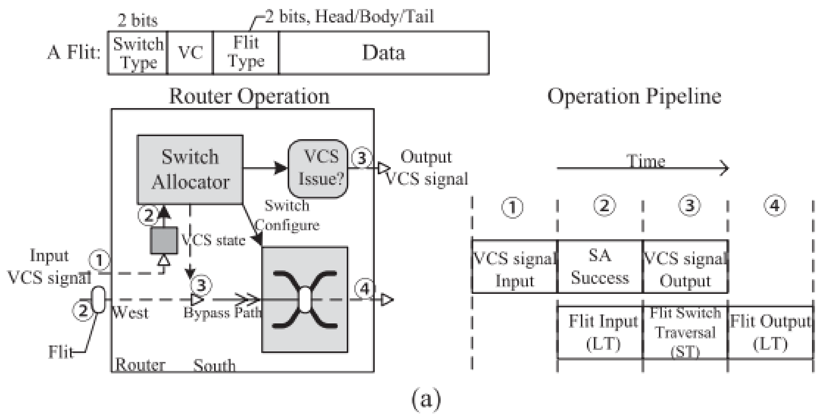


Fig 4.3 Operations of virtual circuit switching. (a) Without blocking

The white flits are buffered since the conflicted output port is allocated to the dark flits. Once the dark flits are blocked or there is no dark flit being input, the crossbar switch will be allocated to the white flits. At this time, which is represented by label 1, a dark flit is traversing the crossbar switch and the switch allocator successfully allocates the crossbar switch to the white flits. At 2 time, since the corresponding field of VCS state marks the output of VCS signal, a VCS signal is transmitted to the downstream router. It is to guarantee that the white flits can still directly traverse the crossbar switch when arriving at the downstream router. This VCS signal is output simultaneously with the LT of the dark flit. Meanwhile, the first white flit in buffers is traversing the crossbar switch. From 3 time, the following white flits in buffers continue to traverse the crossbar switch. In addition, the backpressure will call white flits in upstream routers to restart the transmission on the VCS connection. This VCS signal is output simultaneously with the LT of the dark flit. Meanwhile, the first white flit in buffers is traversing the crossbar switch.

It is to guarantee that the white flits can still directly traverse the crossbar switch when arriving at the downstream router. Once the dark flits are blocked or there is no dark flit being input, the crossbar switch will be allocated to the white flits. At this time, which is represented by label 1, a dark flit is traversing the crossbar switch and the switch allocator successfully allocates the crossbar switch to the white flits.

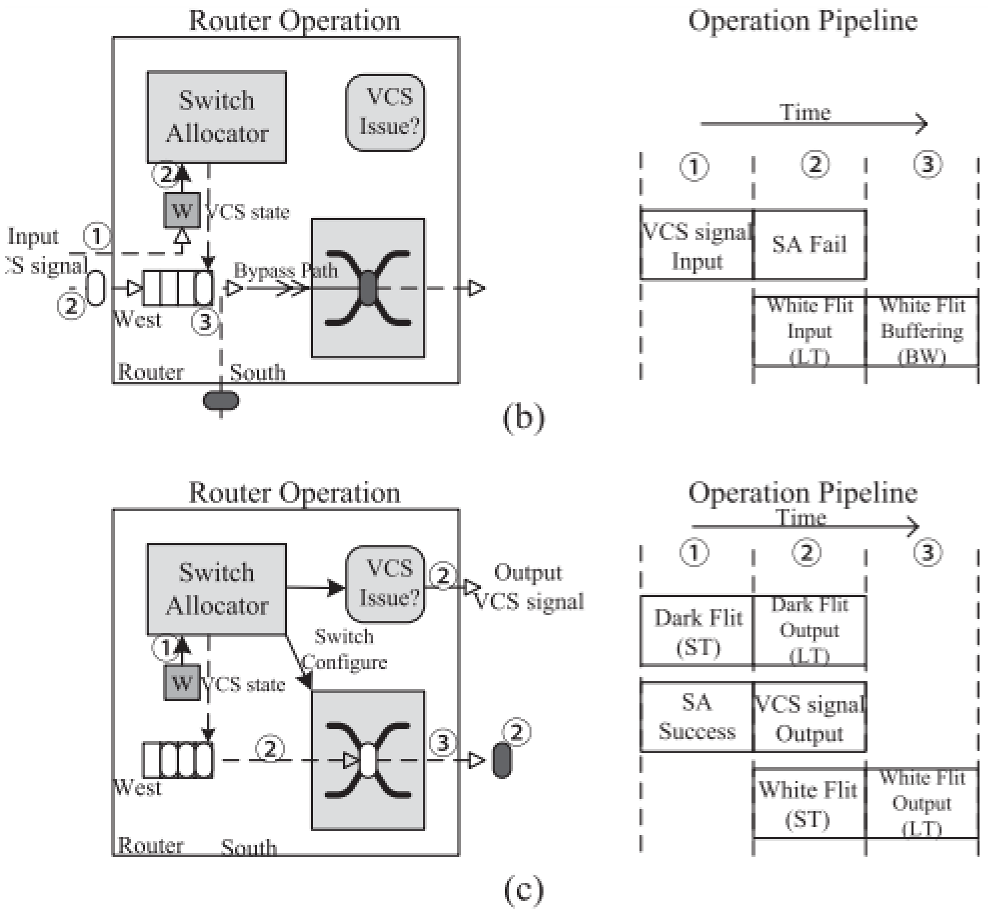


Fig 4.3.1 Operations of virtual circuit switching (b) With blocking. (c) Recovering from blocking.

4) Backpressure: The backpressure of packet switching, including credit-based flow control and ON/OFF flow control, can be applied to virtual circuit switching. This paper adopts credit-based flow control. A credit is consumed when a flit traverses the crossbar switch. At the same time, the router sends a credit back to the upstream router. Once there is no credit, flits are blocked and buffered. Being the same with packet switching, virtual circuit switching also has round-trip delay trt (in unit of clock cycles). However, trt of virtual circuit switching is smaller than that of packet switching due to the smaller number of router pipelines. If the flit buffer number on a VC is smaller than the cycle number of trt, VCS flits in upstream can be blocked due to the lack of credits. When the backpressure calls the continued VCS transmission in upstream, the crossbar switch in the downstream router may be allocated to another VCS transmission. This scenario may also happen when the flit buffer number on a VC is large enough. Thus, to guarantee the VCS transmission in upstream, once the backpressure calls the cease of VCS transmission, the corresponding VCS state in upstream router will be set to mark that VCS signal needs to be issued again. Fig. 4 shows two instances of VCS operations in upstream when recovering from blocking. The transmission direction is from router 0 to router 1. Since VCS flits in router 0 is blocked by backpressure, the output of VCS signal from router 0 to router 1 is required.

In Fig. 4(a), the flit buffer number on a VC is not larger than trt. After backpressure calls the continued VCS transmission in router 0, all VCS flits buffered in router 1 have left. When router 1 receives the VCS signal from router 0, the switch allocator of router 1 reconfigures the crossbar switch and commands the continued transmission of VCS signal. Then flits from router 0 can go directly to the crossbar switch of router 1 once they arrive. In Fig. 4(b), the flit buffer number on a VC is larger than trt. When router 1 receives the VCS signal from router 0, some VCS flits in router 1 are still buffered. The switch allocator in router 1 will not command the output of VCS signal since flits in router 0 can follow the VCS transmission. However, flits from router 0 need to be buffered in router 1 temporarily. In virtual circuit switching, the crossbar switch is configured before the arrival of flits so that VCS flits can directly traverse the crossbar switch without other router pipeline stages. Compared with packet switching, virtual circuit switching can significantly reduce communication latency and power. Although transmissions of VCS signals are required, the power overhead is much less than the power saving of virtual circuit switching. Compared with circuit switching, virtual circuit switching has small power and latency overheads. One extra cycle is needed to send the VCS signal before the VCS transmission of packet. This VCS signal is generated by the network interface at the source node. However, VCS connections are allowed to share a common physical channel. Assuming that there are two conflicted communications, two VCS connections can cost less average latency and power than one CS connection plus one PS connection.

E. Switching Intermingling and Arbitration Policy: As presented above, configured crossbar switches in routers only serve the whole VCS packet until the traversal of the tail flit or the cease of packet transmission. However, flits on CS connections do not need any arbitration and go directly to the ST stage even if the crossbar switch is allocated to other flits. So, the CS connection and the VCS connection are not allowed to compete for one physical channel in the proposed hybrid scheme. Therefore, considering advantages and disadvantages of CS and VCS connections, our proposed hybrid scheme to establish CS and VCS connections can be concluded as follows: first, establish VCS connections if there are other communications competing for the same physical channel; second, establish CS connections if there is no VCS connection competing for the same physical channel. As for the arbitration policy, the winner-take-all bandwidth arbitration, which allocates all of the bandwidth to one packet until it is finished or blocked before serving other packets, is exploited in the proposed hybrid scheme. Data traveling on CS or VCS connections have higher priority than PS flits, because CS flits cannot be ceased and packet transmissions on VCS connections will cost buffer read/write energy when being blocked. However, the higher priority given to CS or VCS flits can lead to a starvation scenario. If a node along the path of a CS or VCS connection always has incoming flits, PS flits buffered locally may never get a chance to use the conflicted port of crossbar switch.

Therefore, starvation avoidance is necessary. For CS connections, we use the method mentioned in. For VCS and PS connections, we use the 0–1 age to control the priority for balancing the starvation and energy. The initial age of the PS transmission is 0. VCS flits have the higher priority than the PS flits with age 0. Even if the crossbar switch is allocated to the PS flits with age 0, VCS flits are still able to occupy the conflicted port of crossbar switch and force the router to stop the PS transmission. Then, the buffer read/write energy can be saved. After PS transmission is blocked for a predetermined number of cycles, the age will increase. PS flits with age 1 have the same priority with VCS flits. Then, the starvation is prevented. This starvation control mechanism with 0–1 age incurs a very small number of priority levels and can be easily implemented.

However, the higher priority given to CS or VCS flits can lead to a starvation scenario. If a node along the path of a CS or VCS connection always has incoming flits, PS flits buffered locally may never get a chance to use the conflicted port of crossbar switch.

Data traveling on CS or VCS connections have higher priority than PS flits, because CS flits cannot be ceased and packet transmissions on VCS connections will cost buffer read/write energy when being blocked. However, the higher priority given to CS or VCS flits can lead to a starvation scenario. If a node along the path of a CS or VCS connection always has incoming flits, PS flits buffered locally may never get a chance to use the conflicted port of crossbar switch. Therefore, starvation avoidance is necessary. For CS connections, we use the method mentioned in. For VCS and PS connections, we use the 0–

1 age to control the priority for balancing the starvation and energy. The initial age of the PS transmission is 0.

If a node along the path of a CS or VCS connection always has incoming flits, PS flits buffered locally may never get a chance to use the conflicted port of crossbar switch.

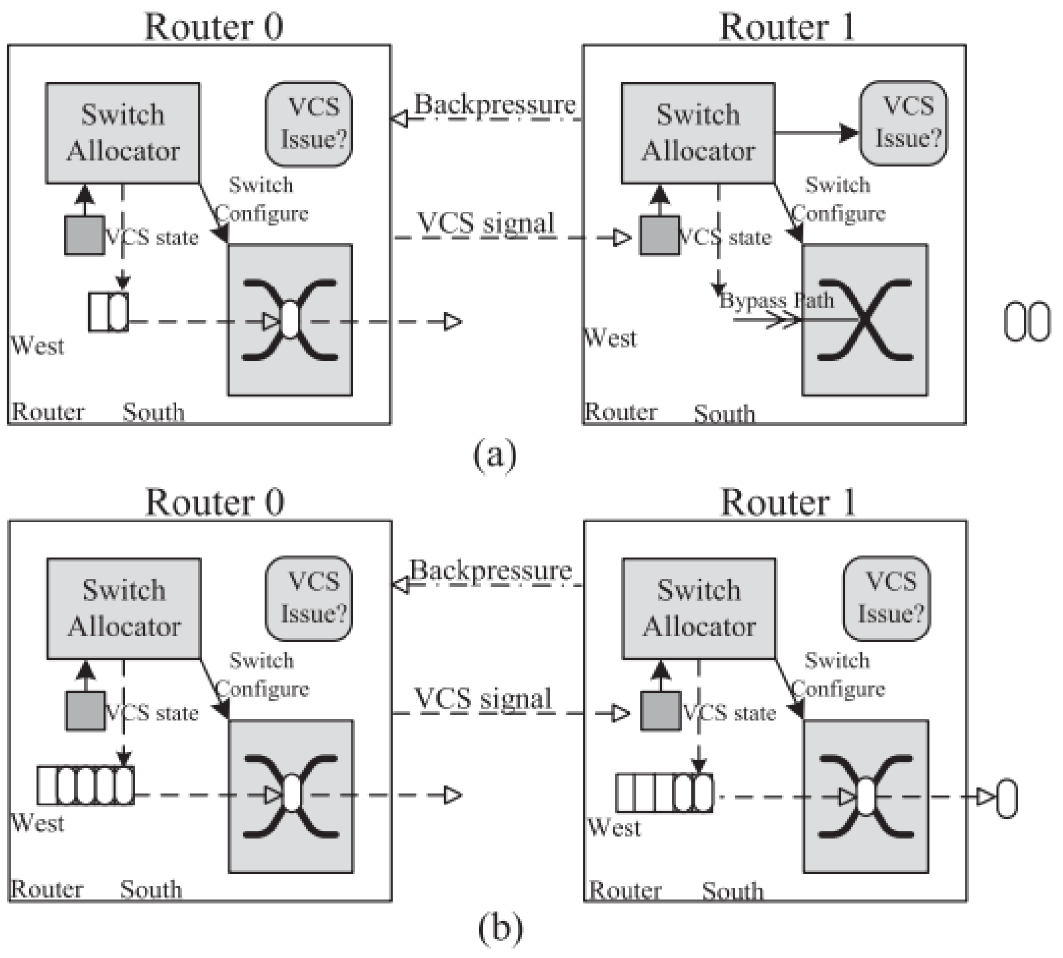


Fig 4.4 Operations of virtual circuit switching in upstream when recovering from blocking. (a) Flit buffer number on a VC is not larger than trt. (b) Flit buffer number on a VC.

CHAPTER 5

INRODUCTION TO VLSI

Very-large-scale integration (VLSI) is the process of creating integrated circuits by combining thousands of transistor-based circuits into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. The microprocessor is a VLSI device. The term is no longer as common as it once was, as chips have increased in complexity into the hundreds of millions of transistors.

5.1 OVERVIEW OF THE PROJECT

The first semiconductor chips held one transistor each. Subsequent advances added more and more transistors, and, as a consequence, more individual functions or systems were integrated over time. The first integrated circuits held only a few devices, perhaps as many as ten diodes, transistors, resistors and capacitors, making it possible to fabricate one or more logic gates on a single device. Now known retrospectively as "small-scale integration" (SSI), improvements in technique led to devices with hundreds of logic gates, known as large-scale integration (LSI), i.e. systems with at least a thousand logic gates. Current technology has moved far past this mark and today's microprocessors have many millions of gates and hundreds of millions of individual transistors.

At one time, there was an effort to name and calibrate various levels of large-scale integration above VLSI. Terms like Ultra-large-scale Integration (ULSI) were used. But the huge number of gates and transistors available on common devices has rendered such fine distinctions moot.

Terms suggesting greater than VLSI levels of integration are no longer in widespread use. Even VLSI is now somewhat quaint, given the common assumption that all microprocessors are VLSI or better.

As of early 2008, billion-transistor processors are commercially available, an example of which is Intel's Montecito Itanium chip. This is expected to become more commonplace as semiconductor fabrication moves from the current generation of 65 nm processes to the next 45 nm generations (while experiencing new challenges such as increased variation across process corners). Another notable example is NVIDIA’s 280 series GPU.

This microprocessor is unique in the fact that its 1.4 Billion transistor count, capable of a teraflop of performance, is almost entirely dedicated to logic (Itanium's transistor count is largely due to the 24MB L3 cache). Current designs, as opposed to the earliest devices, use extensive design automation and automated logic synthesis to lay out the transistors, enabling higher levels of complexity in the resulting logic functionality. Certain high-performance logic blocks like the SRAM cell, however, are still designed by hand to ensure the highest efficiency (sometimes by bending or breaking established design rules to obtain the last bit of performance by trading stability).

5.2 WHAT IS VLSI?

VLSI stands for "Very Large-Scale Integration". This is the field which involves packing more and more logic devices into smaller and smaller area.

Simply we say Integrated circuit is many transistors on one chip. Design/manufacturing of extremely small, complex circuitry using modified semiconductor material. Integrated circuit (IC) may contain millions of transistors, each a few mm in size

Applications wide ranging: most electronic logic devices

5.3 HISTORY OF SCALE INTEGRATION

1. late 40s Transistor invented at Bell Labs
2. late 50s First IC (JK-FF by Jack Kilby at TI)
3. early 60s Small Scale Integration (SSI)
4. 10s of transistors on a chip a late 60s Medium Scale Integration (MSI)
5. 100s of transistors on a chip
6. early 70s Large Scale Integration (LSI)
7. 1000s of transistor on a chip
8. early 80s VLSI 10,000s of transistors on a
9. chip (later 100,000s & now 1,000,000s)
10. Ultra LSI is sometimes used for 1,000,000s

SSI - Small-Scale Integration (0-102)

11. MSI - Medium-Scale Integration (102-103)
12. LSI - Large-Scale Integration (103-105)

13. VLSI - Very Large-Scale Integration (105-107)
14. ULSI - Ultra Large-Scale Integration (>=107)

5.4 ADVANTAGES OF ICs

While we will concentrate on integrated circuits, the properties of integrated circuits-what we can and cannot efficiently put in an integrated circuit-largely determine the architecture of the entire system. Integrated circuits improve system characteristics in several critical ways. ICs have three key advantages over digital circuits built from discrete components:

Size:

Integrated circuits are much smaller-both transistors and wires are shrunk to micrometre sizes, compared to the millimetre or centimetre scales of discrete components. Small size leads to advantages in speed and power consumption, since smaller components have smaller parasitic resistances, capacitances, and inductances.

Speed:

Signals can be switched between logic 0 and logic 1 much quicker within a chip than they can between chips. Communication within a chip can occur hundreds of times faster than communication between chips on a printed circuit board. The high speed of circuits on-chip is due to their small size-smaller components and wires have smaller parasitic capacitances to slow down the signal.

Power consumption:

Logic operations within a chip also take much less power. Once again, lower power consumption is largely due to the small size of circuits on the chip-smaller parasitic capacitances and resistances require less power to drive them.

5.5 VLSI AND SYSTEMS:

These advantages of integrated circuits translate into advantages at the system level:

Smaller physical size. Smallness is often an advantage in itself-consider portable televisions or handheld cellular telephones.

Lower power consumption:

Replacing a handful of standard parts with a single chip reduces total power consumption. Reducing power consumption has a ripple effect on the rest of the system: a smaller, cheaper power supply can be used; since less power consumption means less heat, a fan may no longer be necessary; a simpler cabinet with less shielding for electromagnetic shielding may be feasible, too.

Reduced cost:

Reducing the number of components, the power supply requirements, cabinet costs, and so on, will inevitably reduce system cost. The ripple effect of integration is such that the cost of a system built from custom ICs can be less, even though the individual ICs cost more than the standard parts they replace.

Understanding why integrated circuit technology has such profound influence on the design of digital systems requires understanding both the technology of IC manufacturing and the economics of ICs and digital systems.

Applications

- Electronic system in cars.
- Digital electronics control VCRs
- Transaction processing system, ATM
- Personal computers and Workstations
- Medical electronic systems.

  Etc….

5.6 APPLICATIONS OF VLSI:

Electronic systems now perform a wide variety of tasks in daily life. Electronic systems in some cases have replaced mechanisms that operated mechanically, hydraulically, or by other means; electronics are usually smaller, more flexible, and easier to service. In other cases, electronic systems have created totally new applications. Electronic systems perform a variety of tasks, some of them visible, some more hidden:

- Personal entertainment systems such as portable MP3 players and DVD players perform sophisticated algorithms with remarkably little energy.
- Electronic systems in cars operate stereo systems and displays; they also control fuel injection systems, adjust suspensions to varying terrain, and perform the control functions required for anti-lock braking (ABS) systems.
- Digital electronics compress and decompress video, even at high-definition data rates, on-the-fly in consumer electronics.
- Low-cost terminals for Web browsing still require sophisticated electronics, despite their dedicated function.
- Personal computers and workstations provide word-processing, financial analysis, and games. Computers include both central processing units (CPUs) and special-purpose hardware for disk access, faster screen display.
- Medical electronic systems measure bodily functions and perform complex processing algorithms to warn about unusual conditions. The availability of these complex systems, far from overwhelming consumers, only creates demand for even more complex systems.

The growing sophistication of applications continually pushes the design and manufacturing of integrated circuits and electronic systems to new levels of complexity. And perhaps the most amazing characteristic of this collection of systems is its variety-as systems become more complex, we build not a few general-purpose computers but an ever-wider range of special-purpose systems. Our ability to do so is a testament to our growing mastery of both integrated circuit manufacturing and design, but the increasing demands of customers continue to test the limits of design and manufacturing

5.7 ASIC:

An Application-Specific Integrated Circuit (ASIC) is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use. For example, a chip designed solely to run a cell phone is an ASIC. Intermediate between ASICs and industry standard integrated circuits, like the 7400 or the 4000 series, are application specific standard products (ASSPs).

As feature sizes have shrunk and design tools improved over the years, the maximum complexity (and hence functionality) possible in an ASIC has grown from 5,000 gates to over 100 million. Modern ASICs often include entire 32-bit processors, memory blocks including ROM, RAM, EEPROM, Flash and other large building blocks. Such an ASIC is often termed a SoC (system-on-a-chip). Designers of digital ASICs use a hardware description language (HDL), such as Verilog or VHDL, to describe the functionality of ASICs.

Field-programmable gate arrays (FPGA) are the modern-day technology for building a breadboard or prototype from standard parts; programmable logic blocks and programmable interconnects allow the same FPGA to be used in many different applications. For smaller designs and/or lower production volumes, FPGAs may be more cost effective than an ASIC design even in production.

An application-specific integrated circuit (ASIC) is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use. A Structured ASIC falls between an FPGA and a Standard Cell-based ASIC. Structured ASIC’s are used mainly for mid-volume level design. The design task for structured ASIC’s is to map the circuit into a fixed arrangement of known cells.

CHAPTER 6

INTRODUCTION TO XILINX

6.1 MIGRATING PROJECTS FROM PREVIOUS ISE SOFTWARE:

When you open a project file from a previous release, the ISE® software prompts you to migrate your project. If you click Backup and Migrate or Migrate Only, the software automatically converts your project file to the current release. If you click Cancel, the software does not convert your project and, instead, opens Project Navigator with no project loaded.

Note: After you convert your project, you cannot open it in previous versions of the ISE software, such as the ISE 11 software. However, you can optionally create a backup of the original project as part of project migration, as described below.

6.1.1 To Migrate a Project

In the ISE 12 Project Navigator, select File > Open Project.
In the Open Project dialog box, select the .xise file to migrate.
Note You may need to change the extension in the Files of type field to display .npl
(ISE 5 and ISE 6 software) or .ise (ISE 7 through ISE 10 software) project files.
In the dialog box that appears, select Backup and Migrate or Migrate Only.
The ISE software automatically converts your project to an ISE 12 project.
Note If you chose to Backup and Migrate, a backup of the original project is created at
project_name_ise12 migration.zip.
Implement the design using the new version of the software.
Note Implementation status is not maintained after migration.

6.2 PROPERTIES:

For information on properties that have changed in the ISE 12 software, see ISE 11 to ISE 12 Properties Conversion.

6.3 IP MODULES:

If your design includes IP modules that were created using CORE Generator™ software or Xilinx® Platform Studio (XPS) and you need to modify these modules, you may be required to update the core. However, if the core netlist is present and you do not need to modify the core, updates are not required and the existing netlist is used during implementation.

6.4 OBSOLETE SOURCE FILE TYPES:

The ISE 12 software supports all of the source types that were supported in the ISE 11 software. If you are working with projects from previous releases, state diagram source files (.dia), ABEL source files (.abl), and test bench waveform source files (.tbw) are no longer supported. For state diagram and ABEL source files, the software finds an associated HDL file and adds it to the project, if possible. For test bench waveform files, the software automatically converts the TBW file to an HDL test bench and adds it to the project. To convert a TBW file after project migration, see Converting a TBW File to an HDL Test Bench.

6.5 USING ISE EXAMPLE PROJECTS:

To help familiarize you with the ISE® software and with FPGA and CPLD designs, a set of example designs is provided with Project Navigator. The examples show different design techniques and source types, such as VHDL, Verilog, schematic, or EDIF, and include different constraints and IP.

6.5.1 To Open an Example

1. Select File > Open Example.
2. In the Open Example dialog box, select the Sample Project Name.

Note To help you choose an example project, the Project Description field describes each project. In addition, you can scroll to the right to see additional fields, which provide details about the project.

3. In the Destination Directory field, enter a directory name or browse to the directory.
4. Click OK.

The example project is extracted to the directory you specified in the Destination Directory field and is automatically opened in Project Navigator. You can then run processes on the example project and save any changes.

Note If you modified an example project and want to overwrite it with the original example project, select File > Open Example, select the Sample Project Name, and specify the same Destination Directory you originally used. In the dialog box that appears, select Overwrite the existing project and click OK.

6.6 CREATING A PROJECT:

Project Navigator allows you to manage your FPGA and CPLD designs using an ISE® project, which contains all the source files and settings specific to your design. First, you must create a project and then, add source files, and set process properties. After you create a project, you can run processes to implement, constrain, and analyse your design. Project Navigator provides a wizard to help you create a project as follows.

Note If you prefer, you can create a project using the New Project dialog box instead of the New Project Wizard. To use the New Project dialog box, deselect the Use New Project wizard option in the ISE General page of the Preferences dialog box.

To Create a Project

1. Select File > New Project to launch the New Project Wizard.
2. In the Create New Project page, set the name, location, and project type, and click Next.
3. For EDIF or NGC/NGO projects only: In the Import EDIF/NGC Project page, select the input and constraint file for the project, and click Next.
4. In the Project Settings page, set the device and project properties, and click Next.
5. In the Project Summary page, review the information, and click Finish to create the project

Project Navigator creates the project file (project_name. xise) in the directory you specified. After you add source files to the project, the files appear in the Hierarchy pane of the Design source files, excluding generated files, are copied and placed in a specified directory.

Design source files, excluding generated files, are copied and placed in a specified directory. Project Navigator creates the project file (project name. xise) in the directory you specified.

6.7 DESIGN PANEL:

Project Navigator manages your project based on the design properties (top-level module type, device type, synthesis tool, and language) you selected when you created the project. It organizes all the parts of your design and keeps track of the processes necessary to move the design from design entry through implementation to programming the targeted Xilinx® device.

Note For information on changing design properties, see Changing Design Properties.

- You can now perform any of the following:
- Create new source files for your project.
- Add existing source files to your project.
- Run processes on your source files.

6.8 CREATING A COPY OF A PROJECT:

You can create a copy of a project to experiment with different source options and implementations. Depending on your needs, the design source files for the copied project and their location can vary as follows:

Design source files are left in their existing location, and the copied project points to these files. Design source files, including generated files, are copied and placed in a specified directory. Design source files, excluding generated files, are copied and placed in a specified directory.

Copied projects are the same as other projects in both form and function. For example, you can do the following with copied projects:

- Open the copied project using the File > Open Project menu command.
  View, modify, and implement the copied project.

- Use the Project Browser to view key summary data for the copied project and then, open the copied project for further analysis and implementation, as described in Design source files, excluding generated files, are copied and placed in a specified directory.

Using the Project Browser:

Alternatively, you can create an archive of your project, which puts all of the project contents into a ZIP file. Archived projects must be unzipped before being opened in Project Navigator. For information on archiving, see Creating a Projec**t** Archive**.**

To Create a Copy of a Project

1. Select File > Copy Project.
2. In the Copy Project dialog box, enter the Name for the copy.

Note The name for the copy can be the same as the name for the project, as long as you specify a different location.

3. Enter a directory Location to store the copied project.
4. Optionally, enter a Working directory.

By default, this is blank, and the working directory is the same as the project directory. However, you can specify a working directory if you want to keep your ISE® project file (. xise extension) separate from your working area.

5. Optionally, enter a Description for the copy.

The description can be useful in identifying key traits of the project for reference later.

6. In the Source options area, do the following:

Select one of the following options:

- **Keep sources in their current locations:** to leave the design source files in their existing location.

If you select this option, the copied project points to the files in their existing location. If you edit the files in the copied project, the changes also appear in the original project, because the source files are shared between the two projects.

- **Copy sources to the new location:** to make a copy of all the design source files and place them in the specified Location directory.

If you select this option, the copied project points to the files in the specified directory. If you edit the files in the copied project, the changes do not appear in the original project, because the source files are not shared between the two projects.
Optionally, select Copy files from Macro Search Path directories to copy files from the directories you specify in the Macro Search Path property in the translate Properties dialog box. All files from the specified directories are copied, not just the files used by the design.

Note: If you added a netlist source file directly to the project as described in Working with Netlist-Based IP, the file is automatically copied as part of Copy Project because it is a project source file. Adding netlist source files to the project is the preferred method for incorporating netlist modules into your design, because the files are managed automatically by Project Navigator.

Optionally, click Copy Additional Files to copy files that were not included in the original project. In the Copy Additional Files dialog box, use the Add Files and Remove Files buttons to update the list of additional files to copy. Additional files are copied to the copied project location after all other files are copied. To exclude generated files from the copy, such as implementation results and reports.

6.9EXCLUDE GENERATED FILES FROM THE COPY:

When you select this option, the copied project opens in a state in which processes have not yet been run.

To automatically open the copy after creating it, select Open the copied project.
Note By default, this option is disabled. If you leave this option disabled, the original project remains open after the copy is made.
Click OK.

6.10 CREATING A PROJECT ARCHIVE:

A project archive is a single, compressed ZIP file with a .zip extension. By default, it contains all project files, source files, and generated files, including the following:
User-added sources and associated files
Remote sources
Verilog `include files
Files in the macro search path

Generated files

Non-project files

A CS connection is configured by recording in each router which input port should be connected to which output port. It is composed of physical channels and routers.

6.11 TO ARCHIVE A PROJECT:

Select Project > Archive.
In the Project Archive dialog box, specify a file name and directory for the ZIP file.
Optionally, select Exclude generated files from the archive to exclude generated files and non-project files from the archive.
Click OK.

A ZIP file is created in the specified directory. To open the archived project, you must first unzip the ZIP file, and then, you can open the project.
Note Sources that reside outside of the project directory are copied into a remote_sources subdirectory in the project archive. When the archive is unzipped and opened, you must either specify the location of these files in the remote_sources subdirectory for the unzipped project, or manually copy the sources into their original location.

CHAPTER 7

INTRODUCTION TO VERILOG

In the semiconductor and electronic design industry, Verilog is a hardware description language(HDL) used to model electronic systems. Verilog HDL, not to be confused with VHDL (a competing language), is most commonly used in the design, verification, and implementation of digital logic chips at the register-transfer level of abstraction. It is also used in the verification of Analog and mixed-signal circuits.

7.1 OVERVIEW

Hardware description languages such as Verilog differ from software programming languages because they include ways of describing the propagation of time and signal dependencies (sensitivity). There are two assignment operators, a blocking assignment (=), and a non-blocking (<=) assignment. The non-blocking assignment allows designers to describe a state-machine update without needing to declare and use temporary storage variables (in any general programming language we need to define some temporary storage spaces for the operands to be operated on subsequently; those are temporary storage variables). Since these concepts are part of Verilog's language semantics, designers could quickly write descriptions of large circuits in a relatively compact and concise form. At the time of Verilog's introduction (1984), Verilog represented a tremendous productivity improvement for circuit designers who were already using graphical schematic capture software and specially-written software programs to document and simulate electronic circuits.

The designers of Verilog wanted a language with syntax similar to the C programming language, which was already widely used in engineering software development. Verilog is case-sensitive, has a basic preprocessor (though less sophisticated than that of ANSI C/C++), and equivalent control flow keywords (if/else, for, while, case, etc.), and compatible operator precedence. Syntactic differences include variable declaration (Verilog requires bit-widths on net/reg types demarcation of procedural blocks (begin/end instead of curly braces {}), and many other minor differences. A Verilog design consists of a hierarchy of modules. Modules encapsulate design hierarchy, and communicate with other modules through a set of declared input, output, and bidirectional ports. Internally, a module can contain any combination of the following: net/variable declarations (wire, reg, integer, etc.), concurrent and sequential statement blocks, and instances of other modules (sub-hierarchies). Sequential statements are placed inside a begin/end block and executed in sequential order within the block. But the blocks themselves are executed concurrently, qualifying Verilog as a dataflow language.

Verilog's concept of 'wire' consists of both signal values (4-state: "1, 0, floating, undefined") and strengths (strong, weak, etc.). This system allows abstract modeling of shared signal lines, where multiple sources drive a common net. When a wire has multiple drivers, the wire's (readable) value is resolved by a function of the source drivers and their strengths.

A subset of statements in the Verilog language is synthesizable. Verilog modules that conform to a synthesizable coding style, known as RTL (register-transfer level), can be physically realized by synthesis software. Synthesis software algorithmically transforms the (abstract) Verilog source into a net list, a logically equivalent description consisting only of elementary logic primitives (AND, OR, NOT, flip-flops, etc.) that are available in a specific FPGA or VLSI technology. Further manipulations to the net list ultimately lead to a circuit fabrication blueprint (such as a photo mask set for an ASIC or a bitstream file for an FPGA).

7.2 HISTORY

Verilog was the first modern hardware description language to be invented. It was created by Phil Moorby and Prabhu Goel during the winter of 1983/1984. The wording for this process was "Automated Integrated Design Systems" (later renamed to Gateway Design Automation in 1985) as a hardware modeling language. Gateway Design Automation was purchased by Cadence Design Systems in 1990. Cadence now has full proprietary rights to Gateway's Verilog and the Verilog-XL, the HDL-simulator that would become the de-facto standard (of Verilog logic simulators) for the next decade. Originally, Verilog was intended to describe and allow simulation; only afterwards was support for synthesis added.

Verilog-95

With the increasing success of VHDL at the time, Cadence decided to make the language available for open standardization. Cadence transferred Verilog into the public domain under the Open Verilog International (OVI) (now known as Accellera) organization.

Verilog was later submitted to IEEE and became IEEE Standard 1364-1995, commonly referred to as Verilog-95.

In the same time frame Cadence initiated the creation of Verilog-A to put standards support behind its analog simulator specter. Verilog-A was never intended to be a standalone language and is a subset of Verilog-AMS which encompassed Verilog-95.

Verilog 2001

Extensions to Verilog-95 were submitted back to IEEE to cover the deficiencies that users had found in the original Verilog standard. These extensions became IEEE Standard 1364-2001 known as Verilog-2001.

Verilog-2001 is a significant upgrade from Verilog-95. First, it adds explicit support for (2's complement) signed nets and variables. Previously, code authors had to perform signed operations using awkward bit-level manipulations (for example, the carry-out bit of a simple 8-bit addition required an explicit description of the Boolean algebra to determine its correct value). The same function under Verilog-2001 can be more succinctly described by one of the built-in operators: +, -, /, *, >>>. A generate/end generate construct (similar to VHDL's generate/end generate) allows Verilog-2001 to control instance and statement instantiation through normal decision operators (case/if/else). Using generate/end generate, Verilog-2001 can instantiate an array of instances, with control over the connectivity of the individual instances. File I/O has been improved by several new system tasks. And finally, a few syntax additions were introduced to improve code readability (e.g. always @*, named parameter override, C-style function/task/module header declaration).

Verilog-2001 is the dominant flavor of Verilog supported by the majority of commercial EDA software packages.

Verilog 2005

Not to be confused with System Verilog, Verilog 2005 (IEEE Standard 1364-2005) consists of minor corrections, spec clarifications, and a few new language features (such as the unwire keyword).

A separate part of the Verilog standard, Verilog-AMS, attempts to integrate analog and mixed signal modeling with traditional Verilog.

System Verilog

System Verilog is a superset of Verilog-2005, with many new features and capabilities to aid design verification and design modeling. As of 2009, the System Verilog and Verilog language standards were merged into System Verilog 2009 (IEEE Standard 1800-2009).

The advent of hardware verification languages such as Open Vera, and Verisity's e language encouraged the development of super log by Co-Design Automation Inc. Co-Design Automation Inc was later purchased by Synopsys. The foundations of super log and Vera were donated to Accellera, which later became the IEEE standard P1800-2005: System Verilog.

In the late 1990s, the Verilog Hardware Description Language (HDL) became the most widely used language for describing hardware for simulation and synthesis. However, the first two versions standardized by the IEEE (1364-1995 and 1364-2001) had only simple constructs for creating tests. As design sizes outgrew the verification capabilities of the language, commercial Hardware Verification Languages (HVL) such as Open Vera and were created. Companies that did not want to pay for these tools instead spent hundreds of man-years creating their own custom tools. This productivity crisis (along with a similar one on the design side) led to the creation of Accellera, a consortium of EDA companies and users who wanted to create the next generation of Verilog. The donation of the Open-Vera language formed the basis for the HVL features of System Verilog. Accellera goal was met in November 2005 with the adoption of the IEEE standard P1800-2005 for System Verilog, IEEE (2005).

The most valuable benefit of System Verilog is that it allows the user to construct reliable, repeatable verification environments, in a consistent syntax, that can be used across multiple projects

Some of the typical features of an HVL that distinguish it from a Hardware Description Language such as Verilog or VHDL are

1.Constrained-random stimulus generation

2.Functional coverage

3.Higher-level structures, especially Object Oriented Programming

4.Multi-threading and inter process communication

5.Support for HDL types such as Verilog’s 4-state values

6.Tight integration with event-simulator for control of the design

There are many other useful features, but these allow you to create test benches at a higher level of abstraction than you are able to achieve with an HDL or a programming language such as C.

System Verilog provides the best framework to achieve coverage-driven verification (CDV). CDV combines automatic test generation, self-checking testbenches, and coverage metrics to significantly reduce the time spent verifying a design. The purpose of CDV is to:

- Eliminate the effort and time spent creating hundreds of tests.
- Ensure thorough verification using up-front goal setting.
- Receive early error notifications and deploy run-time checking and error analysis to simplify debugging.

Examples

Ex1: A hello world program looks like this:

```
module main;
initial
begin
$display("Hello world!");
$finish;
end
end module
```

Ex2: A simple example of two flip-flops follows:

```
Moduletoplevel(clock,reset);
input clock;
input reset;

reg flop1;
reg flop2;

always@(pos edge reset or pos edge clock)
if(reset)
begin
flop1 <=0;
```

```
flop2 <=1;
end
else
begin
flop1 <= flop2;
flop2 <= flop1;
end
end module
```

The "<=" operator in Verilog is another aspect of its being a hardware description language as opposed to a normal procedural language. This is known as a "non-blocking" assignment. Its action doesn't register until the next clock cycle. This means that the order of the assignments is irrelevant and will produce the same result: flop1 and flop2 will swap values every clock.

The other assignment operator, "=", is referred to as a blocking assignment. When "=" assignment is used, for the purposes of logic, the target variable is updated immediately. In the above example, had the statements used the "=" blocking operator instead of "<=", flop1 and flop2 would not have been swapped. Instead, as in traditional programming, the compiler would understand to simply set flop1 equal to flop2 (and subsequently ignore the redundant logic to set flop2 equal to flop1.)

Ex3: An example counter circuit follows:

```
module Div20x (rst,clk,cet,cep, count,tc);
// TITLE 'Divide-by-20 Counter with enables'
// enable CEP is a clock enable only
// enable CET is a clock enable and
// enables the TC output
// a counter using the Verilog language

parameter size =5;
parameter length =20;

input rst;// These inputs/outputs represent
input clk;// connections to the module.
```

```
Input cet;
Input cep;
output[size-1:0] count;
output tc;

reg[size-1:0] count;// Signals assigned
// within an always
// (or initial)block
// must be of type reg

wiretc;// Other signals are of type wire
// The always statement below is a parallel
// execution statement that
// executes any time the signals
// rst or clk transition from low to high

always@(posedgeclkorposedgerst)
if(rst)// This causes reset of the cntr
count<={size{1'b0}};
else
if(cet&&cep)// Enables both  true
begin
if(count == length-1)
count<={size{1'b0}};
else
count<= count +1'b1;
end
// the value of tc is continuously assigned
// the value of the expression
assigntc=(cet&&(count == length-1));
end module
```

Ex4: An example of delays:

```
...
reg a, b, c, d;
wire e;
...
always@(b or e)
begin
   a = b & e;
   b = a | b;
#5 c = b;
   d =#6 c ^ e;
end
```

The always clause above illustrates the other type of method of use, i.e. the always clause executes any time any of the entities in the list change, i.e. the b or e change. When one of these changes, immediately a is assigned a new value, and due to the blocking assignment b is assigned a new value afterward (taking into account the new value of a.) After a delay of 5 time units, c is assigned the value of b and the value of c ^ e is tucked away in an invisible store. Then after 6 more time units, d is assigned the value that was tucked away.

Signals that are driven from within a process (an initial or always block) must be of type reg. Signals that are driven from outside a process must be of type wire. The keyword reg does not necessarily imply a hardware register.

7.3 Constants

The definition of constants in Verilog supports the addition of a width parameter. The basic syntax is:

<Width in bits>'<base letter><number>

Examples:

- 12'h123 - Hexadecimal 123 (using 12 bits)
- 20'd44 - Decimal 44 (using 20 bits - 0 extension is automatic)
- 4'b1010 - Binary 1010 (using 4 bits)
- 6'o77 - Octal 77 (using 6 bits)

7.4 SYNTHESIZABLE CONSTRUCTS

There are several statements in Verilog that have no analog in real hardware, e.g. $display. Consequently, much of the language can not be used to describe hardware. The examples presented here are the classic subset of the language that has a direct mapping to real gates.

```
// Mux examples - Three ways to do the same thing.
// The first example uses continuous assignment
wire out;
assign out =sel?a : b;
// the second example uses a procedure
// to accomplish the same thing.
reg out;
always@(a or b orsel)
begin
case(sel)
1'b0: out = b;
1'b1: out = a;
endcase
end
// Finally - you can use if/else in a
// procedural structure.
reg out;
always@(a or b orsel)
if(sel)
out= a;
else
out= b;
```

The next interesting structure is a transparent latch; it will pass the input to the output when the gate signal is set for "pass-through", and captures the input and stores it upon transition of the gate signal to "hold". The output will remain stable regardless of the input signal while the gate is set to "hold". In the example below the "pass-through" level of the gate would be when the value of the if clause is true, i.e. gate = 1. This is read "if gate is true, the din is fed to latch out continuously." Once the if clause is false, the last value at latch out will remain and is independent of the value of din.

```
EX6: // Transparent latch example
reg out;
always@(gate or din)
if(gate)
out= din;// Pass through state
// Note that the else isn't required here. The variable
// out will follow the value of din while gate is high.
// When gate goes low, out will remain constant.
```

The flip-flop is the next significant template; in Verilog, the D-flop is the simplest, and it can be modeled as:

```
reg q;
always@(posedgeclk)
  q <= d;
```

The significant thing to notice in the example is the use of the non-blocking assignment. A basic rule of thumb is to use **<=** when there is a **posedge** or **negedge** statement within the always clause.

A variant of the D-flop is one with an asynchronous reset; there is a convention that the reset state will be the first if clause within the statement.

```
reg q;
always@(posedgeclkorposedge reset)
if(reset)
   q <=0;
else
   q <= d;
```

The next variant is including both an asynchronous reset and asynchronous set condition; again the convention comes into play, i.e. the reset term is followed by the set term.

```
reg q;
always@(posedgeclkorposedge reset or posedge set)
if(reset)
  q <=0;
else
if(set)
```

```
  q <=1;
else
  q <= d;
```

Note: If this model is used to model a Set/Reset flip flop then simulation errors can result. Consider the following test sequence of events. 1) reset goes high 2) clk goes high 3) set goes high 4) clk goes high again 5) reset goes low followed by 6) set going low. Assume no setup and hold violations.

In this example the always @ statement would first execute when the rising edge of reset occurs which would place q to a value of 0. The next time the always block executes would be the rising edge of clk which again would keep q at a value of 0. The always block then executes when set goes high which because reset is high forces q to remain at 0. This condition may or may not be correct depending on the actual flip flop. However, this is not the main problem with this model. Notice that when reset goes low, that set is still high. In a real flip flop this will cause the output to go to a 1. However, in this model it will not occur because the always block is triggered by rising edges of set and reset - not levels. A different approach may be necessary for set/reset flip flops.

Note that there are no "initial" blocks mentioned in this description. There is a split between FPGA and ASIC synthesis tools on this structure. FPGA tools allow initial blocks where reg values are established instead of using a "reset" signal. ASIC synthesis tools don't support such a statement. The reason is that an FPGA's initial state is something that is downloaded into the memory tables of the FPGA. An ASIC is an actual hardware implementation.

7.5 INITIAL VS ALWAYS:

There are two separate ways of declaring a Verilog process. These are the always and the initial keywords. The always keyword indicates a free-running process. The initial keyword indicates a process executes exactly once. Both constructs begin execution at simulator time 0, and both execute until the end of the block. Once an always block has reached its end, it is rescheduled (again). It is a common misconception to believe that an initial block will execute before an always block. In fact, it is better to think of the initial-block as a special-case of the always-block, one which terminates after it completes for the first time.

```
//Examples:
initial
begin
    a =1;// Assign a value to reg a at time 0
#1; // Wait 1 time unit
    b = a;// Assign the value of reg a to reg b
end

always@(a or b)// Any time a or b CHANGE, run the process
begin
if(a)
    c = b;
else
    d =~ b;
end// Done with this block, now return to the top (i.e. the @ event-control)

always@(posedge a)// Run whenever reg a has a low to high change
  a <= b;
```

These are the classic uses for these two keywords, but there are two significant additional uses. The most common of these is an **always** keyword without the **@(...)** sensitivity list. It is possible to use always as shown below:

```
always
begin// Always begins executing at time 0 and NEVER stops
clk=0;// Set clk to 0
#1;// Wait for 1 time unit
clk=1;// Set clk to 1
#1;// Wait 1 time unit
end// Keeps executing - so continue back at the top of the begin
```

The **always** keyword acts similar to the "C" construct **while (1) {..}** in the sense that it will execute forever.

The other interesting exception is the use of the **initial** keyword with the addition of the **forever** keyword.

7.6 RACE CONDITION

The order of execution isn't always guaranteed within Verilog. This can best be illustrated by a classic example. Consider the code snippet below:

```
initial
  a =0;
initial
  b = a;
initial
begin
#1;
$display("Value a=%b Value of b=%b", a, b);
end
```

What will be printed out for the values of a and b? Depending on the order of execution of the initial blocks, it could be zero and zero, or alternately zero and some other arbitrary uninitialized value. The $display statement will always execute after both assignment blocks have completed, due to the #1 delay.

System Tasks:

System tasks are available to handle simple I/O, and various design measurement functions. All system tasks are prefixed with **$** to distinguish them from user tasks and functions. This section presents a short list of the most often used tasks. It is by no means a comprehensive list.

- $display - Print to screen a line followed by an automatic newline.
- $write - Write to screen a line without the newline.
- $swrite - Print to variable a line without the newline.
- $sscanf - Read from variable a format-specified string. (*Verilog-2001)
- $fopen - Open a handle to a file (read or write)
- $fdisplay - Write to file a line followed by an automatic newline.
- $fwrite - Write to file a line without the newline.
- $fscanf - Read from file a format-specified string. (*Verilog-2001)
- $fclose - Close and release an open file handle.
- $readmemh - Read hex file content into a memory array.

- $readmemb - Read binary file content into a memory array.
- $monitor - Print out all the listed variables when any change value.
- $time - Value of current simulation time.
- $dumpfile - Declare the VCD (Value Change Dump) format output file name.
- $dumpvars - Turn on and dump the variables.
- $dumpports - Turn on and dump the variables in Extended-VCD format.
- $random - Return a random value.

Source code:

```
`timescale 1ns / 1ps

module tb;

// Inputs

reg CNFG;

reg RES;

reg LOAD;

reg CS;

reg [7:0] port_A;

reg [7:0] port_B;

reg [7:0] port_C;

reg [7:0] port_D;

reg [1:0] In_add;

reg [1:0] out_add;

reg en_A;

reg en_B;

reg en_C;

reg en_D;


// Outputs

wire [7:0] port_Ao;
```

```
);

wire [7:0] port_Bo;
wire [7:0] port_Co;
wire [7:0] port_Do;

// Instantiate the Unit Under Test (UUT)
NOC uut (
.CNFG(CNFG),
.RES(RES),
.LOAD(LOAD),
.CS(CS),
.port_A(port_A),
.port_B(port_B),
.port_C(port_C),
.port_D(port_D),
.In_add(In_add),
.out_add(out_add),
.en_A(en_A),
.en_B(en_B),
.en_C(en_C),
.en_D(en_D),
.port_Ao(port_Ao),
.port_Bo(port_Bo),
.port_Co(port_Co),
.port_Do(port_Do)
);

initial begin
```

```
CNFG = 0;

RES = 0;#10;

CNFG = 0;

RES = 1;

LOAD = 0;

CS = 1;

port_A = 0;

port_B = 0;

port_C = 0;

port_D = 0;

In_add = 0;

out_add = 0;

en_A = 0;

en_B = 0;

en_C = 0;

en_D = 0;


// Wait 100 ns for global reset to finish

#100;

CNFG = 1;

RES = 0;

LOAD = 1;

CS = 1;

port_A = 20;

port_B = 30;

port_C = 40;

port_D = 50;
```

```
In_add = 1;
out_add = 2;
en_A = 0;
en_B = 0;
en_C = 1;
en_D = 0;
#100;
port_A  = 25;
port_B  = 35;
port_C  = 45;
port_D  = 56;
In_add  = 2;
out_add = 2;
en_A = 1;
en_B = 1;
en_C = 1;
en_D = 1;
#100;
In_add = 2;
out_add = 1;
en_A = 1;
en_B = 1;
en_C = 1;
en_D = 1;
end
end module
```

TEST BENCH

```
module tb;

// Inputs

reg CNFG;

reg RES;

reg LOAD;

reg CS;

reg [7:0] port_A;

reg [7:0] port_B;

reg [7:0] port_C;

reg [7:0] port_D;

reg [1:0] In_add;

reg [1:0] out_add;

reg en_A;

reg en_B;

reg en_C;

reg en_D;

// Outputs

wire [7:0] port_Ao;

wire [7:0] port_Bo;

wire [7:0] port_Co;

wire [7:0] port_Do;

// Instantiate the Unit Under Test (UUT)

NOC uut (

.CNFG(CNFG),

.RES(RES),

.LOAD(LOAD),
.CS(CS),
.port_A(port_A),
.port_B(port_B),
.port_C(port_C),
.port_D(port_D),
.In_add(In_add),
.out_add(out_add),
.en_A(en_A),
.en_B(en_B),
.en_C(en_C),
.en_D(en_D),
.port_Ao(port_Ao),
.port_Bo(port_Bo),
.port_Co(port_Co),
.port_Do(port_Do)
);
initial begin
CNFG = 0;
RES = 0;#10;
CNFG = 0;
RES = 1;
LOAD = 0;
CS = 1;
port_A = 0;
port_B = 0;
port_C = 0;
port_D = 0;
```

```
In_add = 0;

out_add = 0;

en_A = 0;

en_B = 0;

en_C = 0;

en_D = 0;

// Wait 100 ns for global reset to finish

#100;

CNFG = 1;

RES = 0;

LOAD = 1;

CS = 1;

port_A = 20;

port_B = 30;

port_C = 40;

port_D = 50;

In_add = 1;

out_add = 2;

en_A = 0;

en_B = 0;

en_C = 1;

en_D = 0;

#100;

port_A  = 25;

port_B  = 35;

port_C  = 45;

port_D  = 56;
```

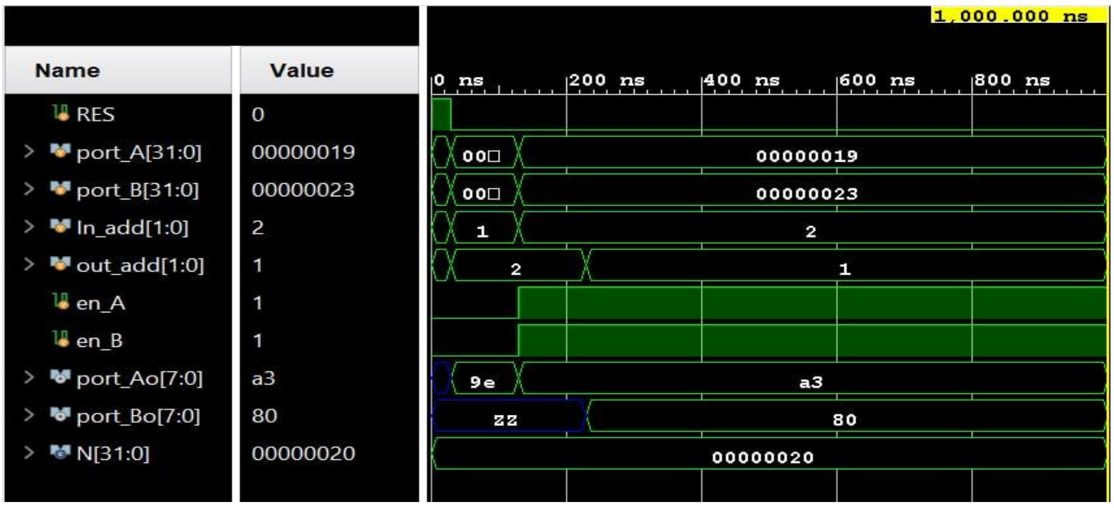
```
In_add  = 2;
out_add = 2;
en_A = 1;
en_B = 1;
en_C = 1;
en_D = 1;
#100;
In_add = 2;
out_add = 1;
en_A = 1;
en_B = 1;
en_C = 1;
en_D = 1;
end
end module
```

CHAPTER 8

RESULTS AND DISCUSSION

Existing results

Figure 6.1shows the existing method results of the simulation for N=32 ,here A,B are consider the inputs and a simulation with inputs A and B, and outputs A' and B', the results for when N = 32.when you input 1, it gives a certain output, and when you input 2, it gives a different output, implying a mismatch in the data.



Name	Value
RES	0
port_A[31:0]	00000019
port_B[31:0]	00000023
In_add[1:0]	2
out_add[1:0]	1
en_A	1
en_B	1
port_Ao[7:0]	a3
port_Bo[7:0]	80
N[31:0]	00000020

Figure 8.1 Existing Simulation Result for N=32

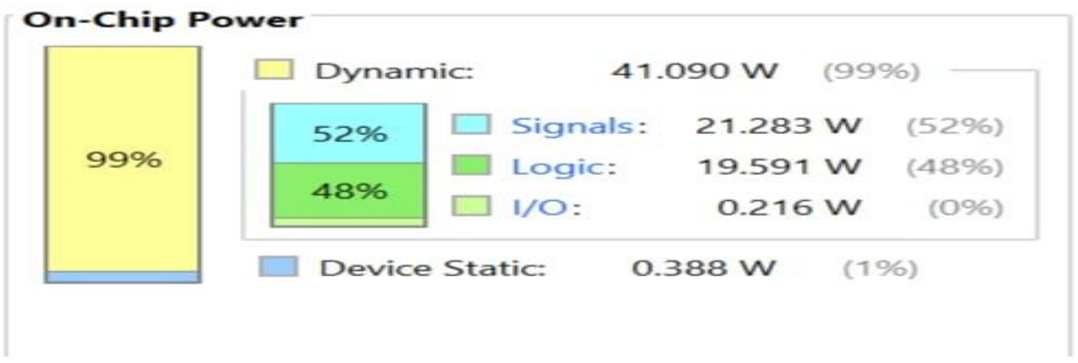
Figure 6.2 shows the existing Area measurements for N=32.Here,1964 lookup tables (LUT) are used out of available 134600, which consumes1.46% of utilization,125 number of IO's are used out of Available 500 IO's which consumes 25.00% of utilization.

Table 8.1 Existing Area for N=32

Resource	Estimation	Available	Utilization
LUT	1964	134600	1.46
IO	125	500	25.00

Figure 6.3 shows existing power measurements for N=32. Here, the total power is 41.478 W, Static power includes PL Static power of 0.388 W, Dynamic power includes signal power of 41.090 W, Logic power of 19.591 W, and I/O power of 0.216 W

A CS connection is configured by recording in each router which input port should be connected to which output port. It is composed of physical channels and routers..

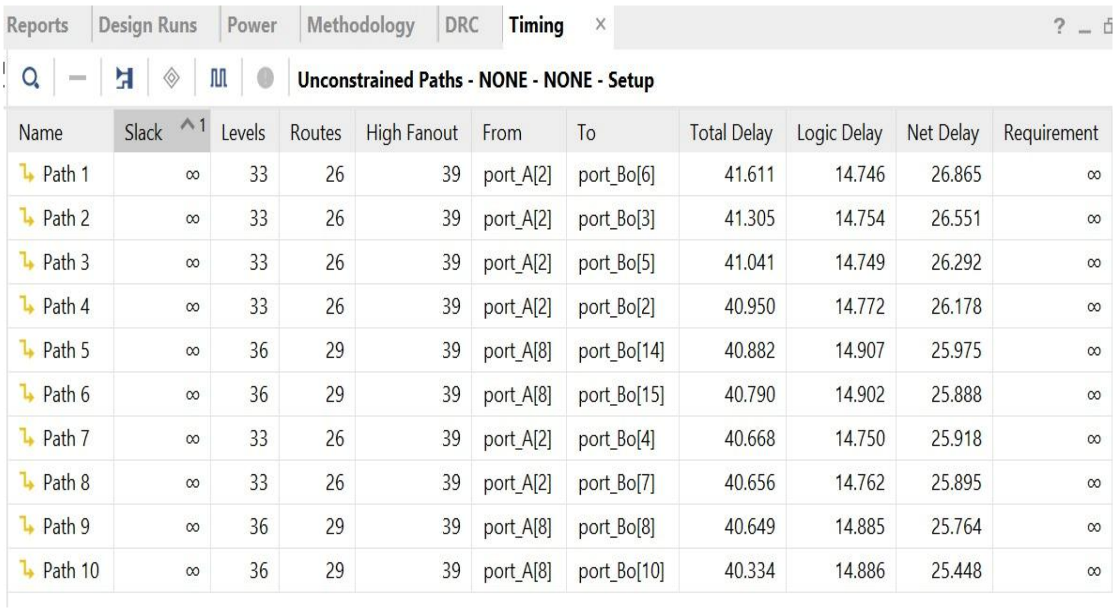


Dynamic:	41.090 W	(99%)
Signals:	21.283 W	(52%)
Logic:	19.591 W	(48%)
I/O:	0.216 W	(0%)
Device Static:	0.388 W	(1%)

Figure 8.2 Existing Power for N=32

Setup Delay

Figure 6.4 shows existing Setup delay for N=32. Here, maximum Total Delay is 41.611 ns, maximum Logic Delay is 14.907 ns, maximum Net Delay is 26.865 ns.



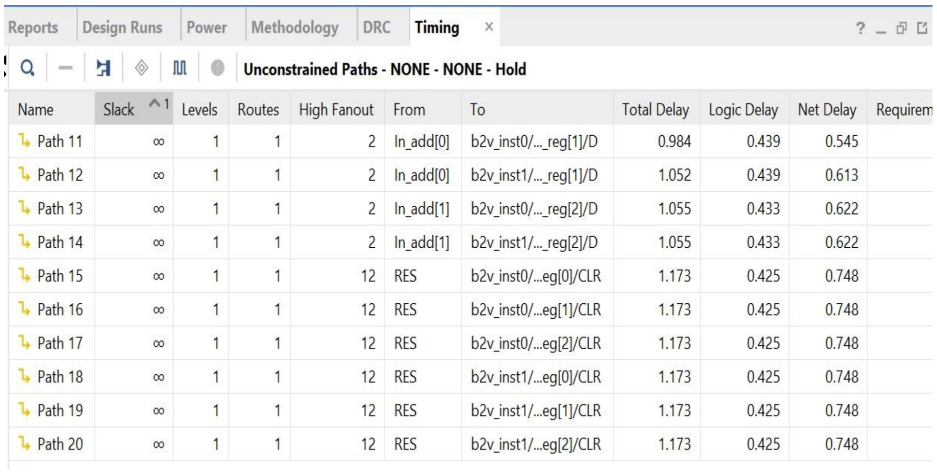
Unconstrained Paths - NONE - NONE - Setup

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	∞	33	26	39	port_A[2]	port_Bo[6]	41.611	14.746	26.865	∞
Path 2	∞	33	26	39	port_A[2]	port_Bo[3]	41.305	14.754	26.551	∞
Path 3	∞	33	26	39	port_A[2]	port_Bo[5]	41.041	14.749	26.292	∞
Path 4	∞	33	26	39	port_A[2]	port_Bo[2]	40.950	14.772	26.178	∞
Path 5	∞	36	29	39	port_A[8]	port_Bo[14]	40.882	14.907	25.975	∞
Path 6	∞	36	29	39	port_A[8]	port_Bo[15]	40.790	14.902	25.888	∞
Path 7	∞	33	26	39	port_A[2]	port_Bo[4]	40.668	14.750	25.918	∞
Path 8	∞	33	26	39	port_A[2]	port_Bo[7]	40.656	14.762	25.895	∞
Path 9	∞	36	29	39	port_A[8]	port_Bo[8]	40.649	14.885	25.764	∞
Path 10	∞	36	29	39	port_A[8]	port_Bo[10]	40.334	14.886	25.448	∞

Figure 8.3 Existing setup Delay for N=32

Hold Delay

Figure 6.5 shows existing Hold delay for N=32. Here, maximum Total Delay is 1.173 ns, maximum Logic Delay is 0.439 ns, maximum Net Delay is 0.748 ns.



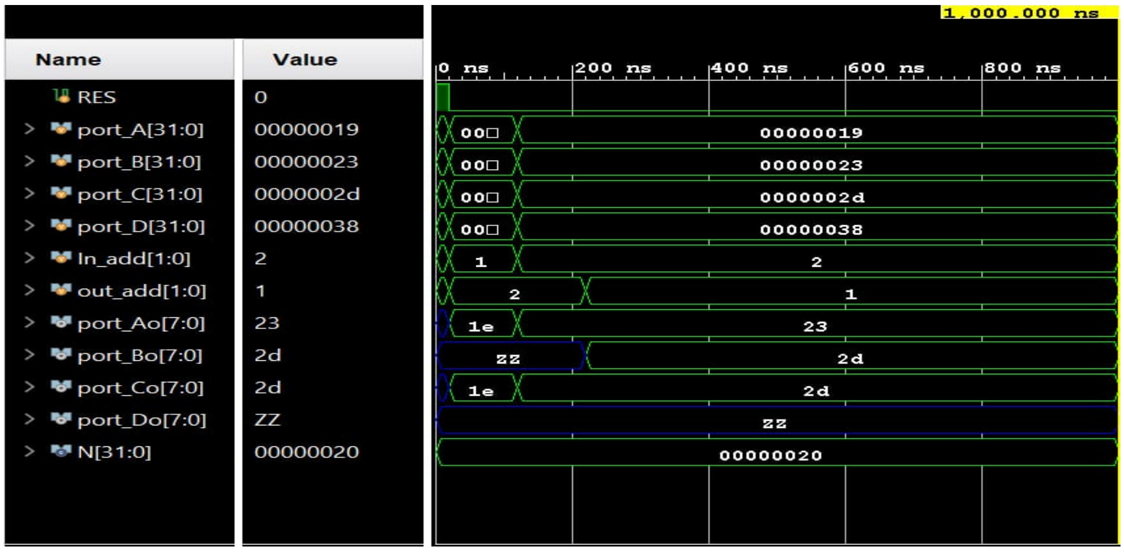
Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirem
Path 11	∞	1	1	2	In_add[0]	b2v_inst0/..._reg[1]/D	0.984	0.439	0.545	
Path 12	∞	1	1	2	In_add[0]	b2v_inst1/..._reg[1]/D	1.052	0.439	0.613	
Path 13	∞	1	1	2	In_add[1]	b2v_inst0/..._reg[2]/D	1.055	0.433	0.622	
Path 14	∞	1	1	2	In_add[1]	b2v_inst1/..._reg[2]/D	1.055	0.433	0.622	
Path 15	∞	1	1	12	RES	b2v_inst0/...eg[0]/CLR	1.173	0.425	0.748	
Path 16	∞	1	1	12	RES	b2v_inst0/...eg[1]/CLR	1.173	0.425	0.748	
Path 17	∞	1	1	12	RES	b2v_inst0/...eg[2]/CLR	1.173	0.425	0.748	
Path 18	∞	1	1	12	RES	b2v_inst1/...eg[0]/CLR	1.173	0.425	0.748	
Path 19	∞	1	1	12	RES	b2v_inst1/...eg[1]/CLR	1.173	0.425	0.748	
Path 20	∞	1	1	12	RES	b2v_inst1/...eg[2]/CLR	1.173	0.425	0.748	

Figure 8.4 Existing Hold delay for N=32

Proposed Results

shows the Proposed method results of the simulation for N=32, here A,B are consider the inputs and a simulation with inputs A and B, and outputs A' and B', the results for when N = 32when you input 1, it gives a certain output, and when you input 2, it gives same output, perfect match in the data.



Name	Value
RES	0
port_A[31:0]	00000019
port_B[31:0]	00000023
port_C[31:0]	0000002d
port_D[31:0]	00000038
In_add[1:0]	2
out_add[1:0]	1
port_Ao[7:0]	23
port_Bo[7:0]	2d
port_Co[7:0]	2d
port_Do[7:0]	ZZ
N[31:0]	00000020

Figure 8.5 Proposed Simulation Result for N=32

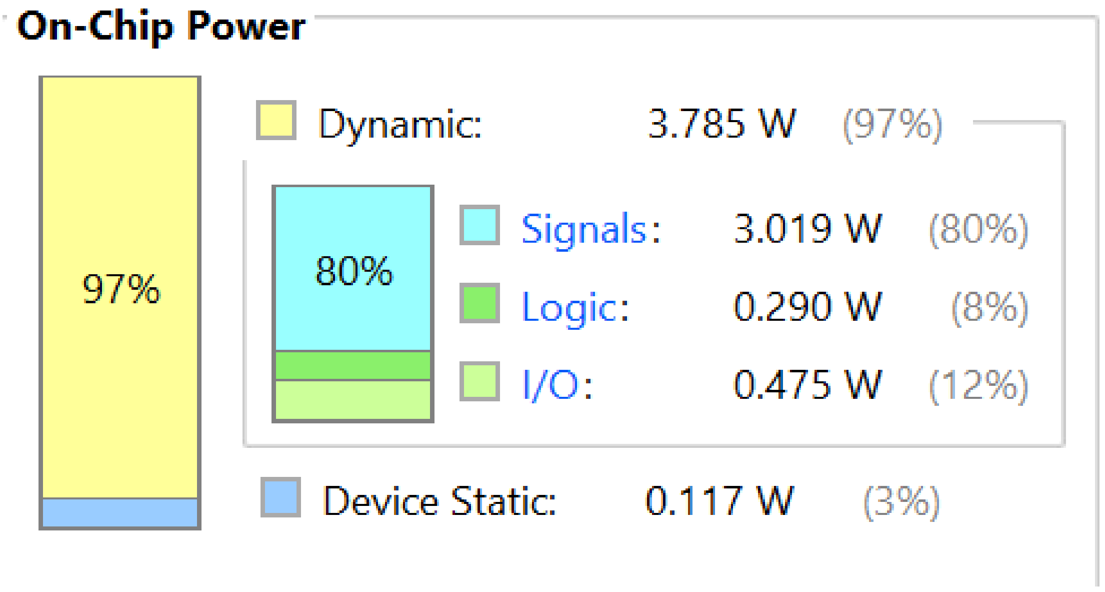
Figure 6.7 shows the Proposed Area measurements for N=32.Here, 135 Lookup tables (LUT)

are used out of available 134600, which consumes0.10% of utilization,261 number of IO's are used out of Available 500 IO's which consumes 52.20% of utilization.

Table 8.2 Proposed Area for N=32

Resource	Estimation	Available	Utilization
LUT	135	134600	0.10
IO	261	500	52.20

Figure 6.8 shows Proposed power measurements for N=32. Here, the total power is 3.902 W, Static power includes PL Static power of 0.117 W, Dynamic power includes signal power of 3.785 W, Logic power of 0.290 W, and I/O power of 0.475 W.

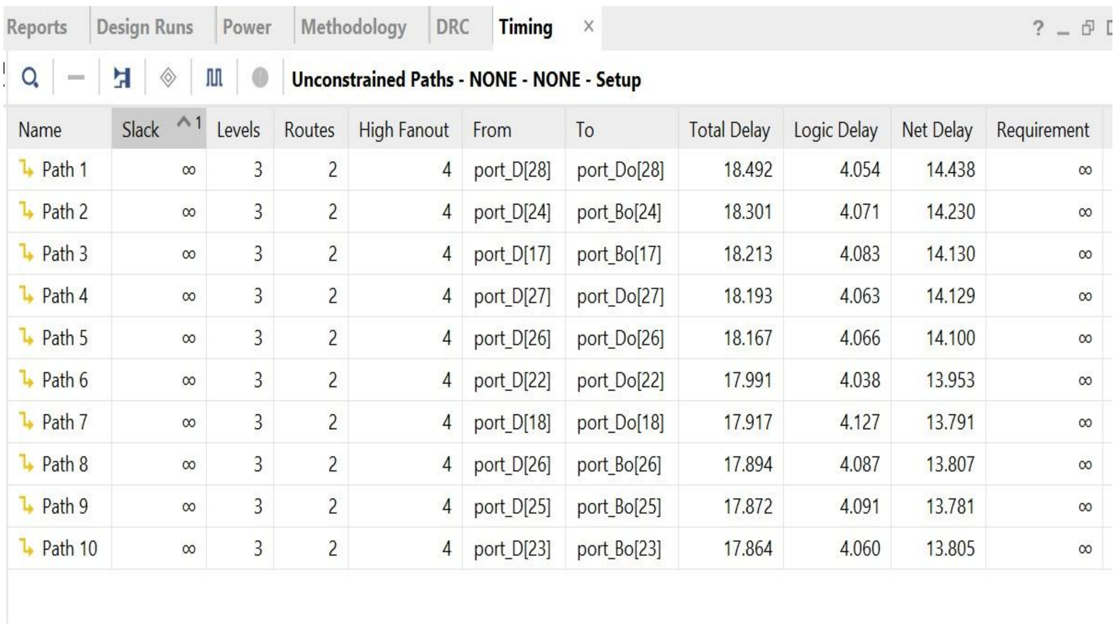


🟨	Dynamic:	3.785 W	(97%)
🟦	Signals:	3.019 W	(80%)
🟩	Logic:	0.290 W	(8%)
🟩	I/O:	0.475 W	(12%)
🟦	Device Static:	0.117 W	(3%)

Figure 8.6 Proposed power for N=32

Setup Delay

Figure 6.9 shows Proposed Setup delay for N=32. Here, maximum Total Delay is 18.492 ns, maximum Logic Delay is 4.091 ns, maximum Net Delay is 14.438 ns.

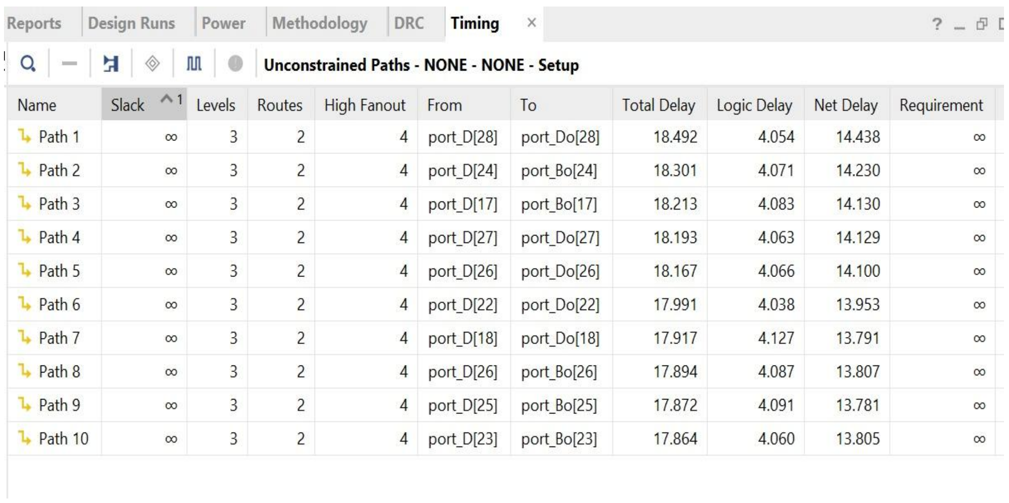


Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	∞	3	2	4	port_D[28]	port_Do[28]	18.492	4.054	14.438	∞
Path 2	∞	3	2	4	port_D[24]	port_Bo[24]	18.301	4.071	14.230	∞
Path 3	∞	3	2	4	port_D[17]	port_Bo[17]	18.213	4.083	14.130	∞
Path 4	∞	3	2	4	port_D[27]	port_Do[27]	18.193	4.063	14.129	∞
Path 5	∞	3	2	4	port_D[26]	port_Do[26]	18.167	4.066	14.100	∞
Path 6	∞	3	2	4	port_D[22]	port_Do[22]	17.991	4.038	13.953	∞
Path 7	∞	3	2	4	port_D[18]	port_Do[18]	17.917	4.127	13.791	∞
Path 8	∞	3	2	4	port_D[26]	port_Bo[26]	17.894	4.087	13.807	∞
Path 9	∞	3	2	4	port_D[25]	port_Bo[25]	17.872	4.091	13.781	∞
Path 10	∞	3	2	4	port_D[23]	port_Bo[23]	17.864	4.060	13.805	∞

Table 8.7 Proposed Setup Delay for N=32

Hold Delay

Figure 6.10 shows Proposed Hold delay for N=32. Here, maximum Total Delay is 18.492 ns, maximum Logic Delay is 4.091 ns, maximum Net Delay is 14.438 ns.



Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	∞	3	2	4	port_D[28]	port_Do[28]	18.492	4.054	14.438	∞
Path 2	∞	3	2	4	port_D[24]	port_Bo[24]	18.301	4.071	14.230	∞
Path 3	∞	3	2	4	port_D[17]	port_Bo[17]	18.213	4.083	14.130	∞
Path 4	∞	3	2	4	port_D[27]	port_Do[27]	18.193	4.063	14.129	∞
Path 5	∞	3	2	4	port_D[26]	port_Do[26]	18.167	4.066	14.100	∞
Path 6	∞	3	2	4	port_D[22]	port_Do[22]	17.991	4.038	13.953	∞
Path 7	∞	3	2	4	port_D[18]	port_Do[18]	17.917	4.127	13.791	∞
Path 8	∞	3	2	4	port_D[26]	port_Bo[26]	17.894	4.087	13.807	∞
Path 9	∞	3	2	4	port_D[25]	port_Bo[25]	17.872	4.091	13.781	∞
Path 10	∞	3	2	4	port_D[23]	port_Bo[23]	17.864	4.060	13.805	∞

Figure 8.8 proposed Hold Delay for N=32

CONCLUSION

In this work, we present a novel hybrid scheme based on virtual circuit switching to further reduce communication latency and power of NoCs. The basic principle of the proposed hybrid scheme is to intermingle virtual circuit switching with circuit switching and packet switching. Intermediate router pipelines are bypassed by establishing VCS connections and CS connections. A path allocation algorithm is also presented to smartly allocate VCS connections and CS connections for a given traffic in mesh connected NoCs, such that the average packet latency and energy consumption are both optimized. To demonstrate the effectiveness of the proposed hybrid scheme, a set of synthetic traffic workloads and real traffic loads are exploited for evaluation.

Future work:

Our future work will focus on extending the current work to support applications with unpredictable communication patterns. Other extensions include the fault tolerance, the quality of-service (QoS) operation, the multicast delivery service, and the mapping, scheduling of applications based on virtual circuit switching. In fact, due to the small area overhead, the proposed hybrid scheme can have the similar reliability and bit-error rate when compared with the baseline NoC and VIP design. In addition, some fault-tolerance techniques, such as error control codes, structural redundancy, and packet retransmission, can be utilized to increase the reliability of the proposed hybrid scheme. Moreover, the proposed hybrid scheme can be exploited to achieve the QoS operation. For example, VCS connections and CS connections can be limited to the class of communications that need guaranteed latency, and packet switching can be used to serve the best effort traffic.

REFERENCES

1. Shehzad, F., Rashid, M., Sinky, M. H., Alotaibi, S. S., & Zia, M. Y. I. (2024). A Scalable System-on-Chip Acceleration for Deep Neural Networks. IEEE Access, 9, 95412-95426.

2. Chen, Ying, Sae-Won Lee, and Rodney G. Vaughan. "Self-assembled triangular waveguide slot array for system-on-chip applications." IET Microwaves, Antennas & Propagation 11.14 (2020): 2035-2042.

3. Basak, A., Bhunia, S., Tkacik, T., & Ray, S. (2019). Security assurance for system-on-chip designs with untrusted IPs. IEEE Transactions on Information Forensics and Security, 12(7), 1515-1528.

4. Arulananth, T. S., Baskar, M., SM, U. S., Thiagarajan, R., Rajeshwari, P. R., Kumar, A. S., & Suresh, A. (2021). Evaluation of low power consumption network on chip routing architecture. Microprocessors and Microsystems, 82, 103809.

5. Vu, The H., Ogbodo Mark Ikechukwu, and Abderazek Ben Abdallah. "Fault-tolerant spike routing algorithm and architecture for three dimensional noc-based neuromorphic systems." IEEE Access 7 (2020): 90436-90452.

6. Wang, Ke, Hao Zheng, and Ahmed Louri. "Tsa-noc: Learning-based threat detection and mitigation for secure network-on-chip architecture." IEEE Micro 40.5 (2021): 56-63.

7. Zheng, Hao, and Ahmed Louri. "Ez-pass: An energy & performance-efficient power-gating router architecture for scalable nocs." IEEE Computer Architecture Letters 17.1 (2018): 88-91.

8. Bui, P. D., & Lee, C. (2020). Unified System Network Architecture: Flexible and Area-Efficient NoC Architecture with Multiple Ports and Cores. Electronics, 9(8), 1316.

9. Reza, M. F., & Ampadu, P. (2019, October). Energy-efficient and high-performance NoC architecture and mapping solution for deep neural networks. In Proceedings of the 13th IEEE/ACM International Symposium on networks-on-chip (pp. 1-8).

10. Xu, Changqing, Yi Liu, and Yintang Yang. "SRNoC: An ultra-fast configurable FPGA-based NoC simulator using switch–router architecture." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39.10 (2019): 2798-2811.

APPENDIX

Appendix A: Project Specifications

- **Project Title**: Efficient VLSI Design of Network Packet Switching Unit for Ethernet Communication
- **Domain**: VLSI Design / Computer Networks
- **Technology Used**:
    - VHDL / Verilog HDL
    - Xilinx Vivado / ModelSim
    - Simulation & Synthesis Tools
- **Target Device**: FPGA (e.g., Xilinx Spartan/Artix family)
- **Clock Frequency**: 50 MHz (customizable)
- **Data Width**: 8-bit / 16-bit (based on design)
- **Packet Format**: Custom Ethernet-style header with payload
- **Switching Type**: Store-and-Forward or Cut-Through (based on implementation)

Appendix B: Functional Blocks in NPSU

- **Input Port Buffer**
- **Packet Parser**
- **Routing Decision Logic**
- **Crossbar Switch**
- **Output Port Scheduler**
- **FIFO Memory Structures**
- **Control Unit for Flow Management**

Appendix C: Design Objectives

- Reduce overall **packet switching latency**
- Improve **throughput** using pipelining techniques
- Optimize **power consumption** with efficient logic gates
- Support **scalability** to higher port counts (e.g., 4-port, 8-port)
- Ensure **compatibility** with standard Ethernet packet structures

Appendix D: Simulation Results (If Available)

- **Testbench Scenarios**: Packet injection, collision test, buffer full condition
- **Waveform Screenshots**: From Vivado or ModelSim (to be inserted)
- **Timing Summary**: Worst-case delay, clock usage
- **Power Report**: Total dynamic and static power (from synthesis)

Appendix E: Tools & Libraries Used

- **Vivado Design Suite** (Version XX.XX)
- **ModelSim Simulator** (for functional verification)
- **Xilinx IP Cores** (if any used)
- **VHDL/Verilog Libraries**: IEEE Std Logic 1164, Numeric Std

Appendix F: Future Scope

- Integrate **AI-based routing algorithms** for intelligent switching
- Extend the architecture to support **QoS (Quality of Service)**
- Implement the design on **ASIC** for production-level efficiency
- Expand to **multi-layer switch architectures** for complex networks