ABSTRACT

ON CHIP IMPLEMENTATION OF DEADLOCK AVOIDANCE IN WORMHOLE NETWORKS

by
Kiran K. Gururaj

This thesis gives a detailed description of the Application Specific Integrated Circuit (ASIC) design to avoid deadlocks in Wormhole Networks. Deadlock avoidance is the most critical issue while considering wormhole networks and should be avoided by any routing protocol and algorithm. A novel architecture for the Turn Prohibition Based Routing (TPBR) protocol has been proved to be efficient and was developed as a part of this work. This architecture for implementing the algorithm is divided into three parts. The first part determines the order of selection of the nodes in the network to run the algorithm. The second part deals with the prohibition of the turns through the node, which might possibly create a deadlock. The third part constructs a routing table, which will have the route from a source to a destination, considering the prohibited turns into account. A VHDL model was developed and simulated using IEEE numeric_std package for this architecture. This model was synthesized with Cadence tools and the post synthesis simulations verified the functionality of the architecture. The physical design was created using the standard gate cell libraries and implemented in 0.35-micron CMOS technology.

ON CHIP IMPLEMENTATION OF DEADLOCK AVOIDANCE IN WORMHOLE NETWORKS

by
Kiran K. Gururaj

A Thesis
Submitted to
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

August 2002

APPROVAL PAGE

ON CHIP IMPLEMENTATION OF DEADLOCK AVOIDANCE IN WORMHOLE NETWORKS

Kiran K. Gururaj

8/29/02

Dr. Durga Misra, Thesis Advisor — Date
Professor of Electrical and Computer Engineering, NJIT

08/29/02

Dr. Lev Zakrevski, Committee Member — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

08/29/02

Dr. Symeon Papavassiliou, Committee Member — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Kiran K. Gururaj

Degree: Master of Science

Date: August 2002

Date of Birth:

Place of Birth:

Education

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2002

- Bachelor of Engineering in Electronics and Communications Engineering,
  Bangalore University, Bangalore, KA, INDIA

Major: Electrical Engineering

Publications:

Kiran K. Gururaj, Lev Zakrevski and D. Misra, "VLSI Architecture for Deadlock Avoidance in Wormhole Networks", Proceedings of Fifteenth International Conference on Systems Engineering, ICSE 2002, pp. 322-328, Aug 2002.

To My Parents and the Almighty

ACKNOWLEDGEMENT

The author wishes to express his sincere gratitude to his Graduate Advisor and Thesis guide Professor D. Misra, for his guidance, moral support and encouragement throughout this work.

Special thanks to Assistant Professors Lev Zakrevski and Symeon Papavassiliou for serving as the members of the committee.

The author also thanks Prof. D. Misra and NJ center for Optoelectronics for the financial assistance during Fall 2001 and Spring 2002.

The author also thanks the Computing Services Division of NJIT for providing the necessary tools and also appreciates the services rendered by them.

Finally, the author would like to thank his beloved parents for giving the encouragement and moral support throughout his course of study at NJIT.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Applications and Importance of NOW

Over the past few years, networks with irregular topologies like the Network Of Workstations (NOW) have become an alternative to the parallel multiprocessors. In order to reduce the communication latency, memory requirements and achieve high bandwidth in data communications, the NOW use wormhole routing. However wormhole routing is very much prone to deadlocks as packets are allowed to hold one node while requesting for others. Deadlock avoidance is the most critical issue while considering wormhole networks. Deadlock is an undesired feature that should be avoided by any routing protocol and algorithm because a lot of network resources will be wasted and the performance of the network will be degraded due to the abuse of network resource by deadlock routing. Therefore, design of deadlock-free routing protocols is important to the performance of a NOW. Although NOW do not provide the computing power available in multi-computers and multiprocessors, they meet the needs of a great variety of parallel computing problems at a lower cost. By using the Turn Prohibition Based Routing algorithm (TPBR), deadlocks in the networks can be avoided by eliminating the turns in the networks. In wormhole routing each packet consists of a sequence of elementary flow control units called flits. As long as the transmission route is free, flits are forwarded to their destination in a pipelined manner. The main advantage compared to the store and forward switching is that the latency is reduced since there is no need to wait for the end of the packet before transmitting flits to the next node. However these contiguous flits in

a packet are always contained in the same or adjacent nodes of the network. This can cause difficulties, as possibility of deadlock arises. Deadlock in the interconnection network occurs when no message can advance towards its destination because the queues of the message system are full. As the header flit advances along a specified route, the subsequent flits follow up in a pipelined fashion. This technique provides for low communication latency, almost independent of the distance between the source and the destination. Whenever the header flit is blocked at an intermediate node by another message, the remaining flits stop advancing thus blocking each other. When considering wormhole routing it must either have an effective procedure of recovery after deadlocks or must be deadlocks free which is implemented in the form of a protocol. To prevent deadlocks the necessary and sufficient condition is to eliminate the cycles in a channel dependency graph. This can be achieved by eliminating the turns in the graph, which will form closed loops in the network. Even though there exists other algorithms that implement the wormhole routing, the TPBR algorithm proves to be a better solution for eliminating the deadlocks in the networks.

Networks of Workstations (NOW) are comprised of a collection of routers, communication links and workstations in an irregular topology. They have been applied as an alternative to parallel multiprocessors. In a NOW, the message routed through the network is subject to *deadlock*, which means the path between source and destination forms a loop. Therefore, by applying TPBR algorithm, deadlock can be broken and the flexibility of routing can still be maintained because of the small number of prohibited turns. However, in TPBR algorithm works as a centralized algorithm which needs knowledge of the network topology to determine the prohibited turns. For decentralized

network equipment such as a router, it is difficult for it to get knowledge of the network topology, especially for large-scale networks. In this thesis a protocol for TPBR algorithm to be performed in a decentralized manner is shown, which is implemented on a VLSI chip.

1.2 Design Flow of the TPBR Chip

In this thesis, a novel architecture of the TPBR algorithm is proposed, which will make the Network Of Workstations deadlock free. This architecture will be the basis for implementing the Turn Prohibition Based Routing algorithm in VHDL and designing an ASIC. The design flow for implementing the algorithm as an ASIC is as shown in Figure. 1.1. Mostly Cadence tools were used, except Model Sim for VHDL simulations and HSPICE for spice simulations.

The VHDL model for the algorithm is compiled and simulated using Model Sim. The VHDL model is synthesized using Cadence Ambit Bulidgates. The design is synthesized, mapped to the cells in the library for timing calculations. The output from this tool gives us a Verilog file, which has gate level design information, a GCF file, which has timing information and a DEF file which has placement information.

The placement and Routing of the standard cells is done using Cadence Silicon Ensemble. The design, placement and timing information is read from the above produced files and timing driven routing is done. The DRC and parasitic extraction is done in Cadence Virtuoso environment. The extracted spice file is simulated using HSPICE.

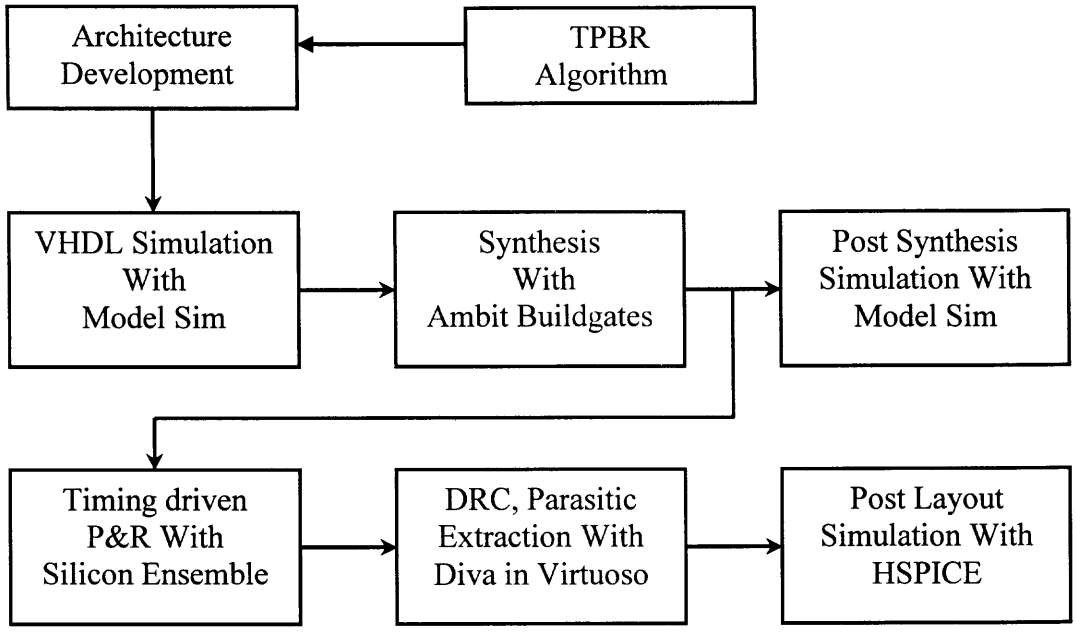

Figure 1.1 Design flow for implementing the TPBR algorithm on a chip.

1.3 Outline of the Thesis

The remainder of the thesis is organized as follows. Chapter 2 provides an overview of Wormhole routing, Deadlocks in a network, various routing strategies present till date and also the Up/Down routing technique. Chapter 3 provides a detailed description of the TPBR algorithm and its advantages over Up/Down routing technique. Chapter 4 describes the proposed architecture of the chip and its various components. Chapter 5 describes the VHDL model and different implementation issues, and it includes the test and the simulation results. Chapter 6 addresses the synthesis issues and problems encountered during the synthesis and modifications of the actual VHDL model to make it synthesizable. Then the simulations of the gate level verilog netlist after synthesis are

listed. Chapter 7 describes the physical layout and device level simulation of the final layout. Chapter 8 concludes this thesis by summarizing the results of the work and discussing alternative implementations for reducing the gate count.

CHAPTER 2

OVERVIEW OF ROUTING CONCEPTS

2.1 Wormhole Routing

Packets in a network can be transmitted in many ways, including one called wormhole routing. In wormhole routing, packets are divided into smaller parts of equal size called flits (= flow-control digit), which are then transmitted one by one, instead of transmitting the packet as a whole. All the flits of the same packet follow the same path and cannot overtake each other.

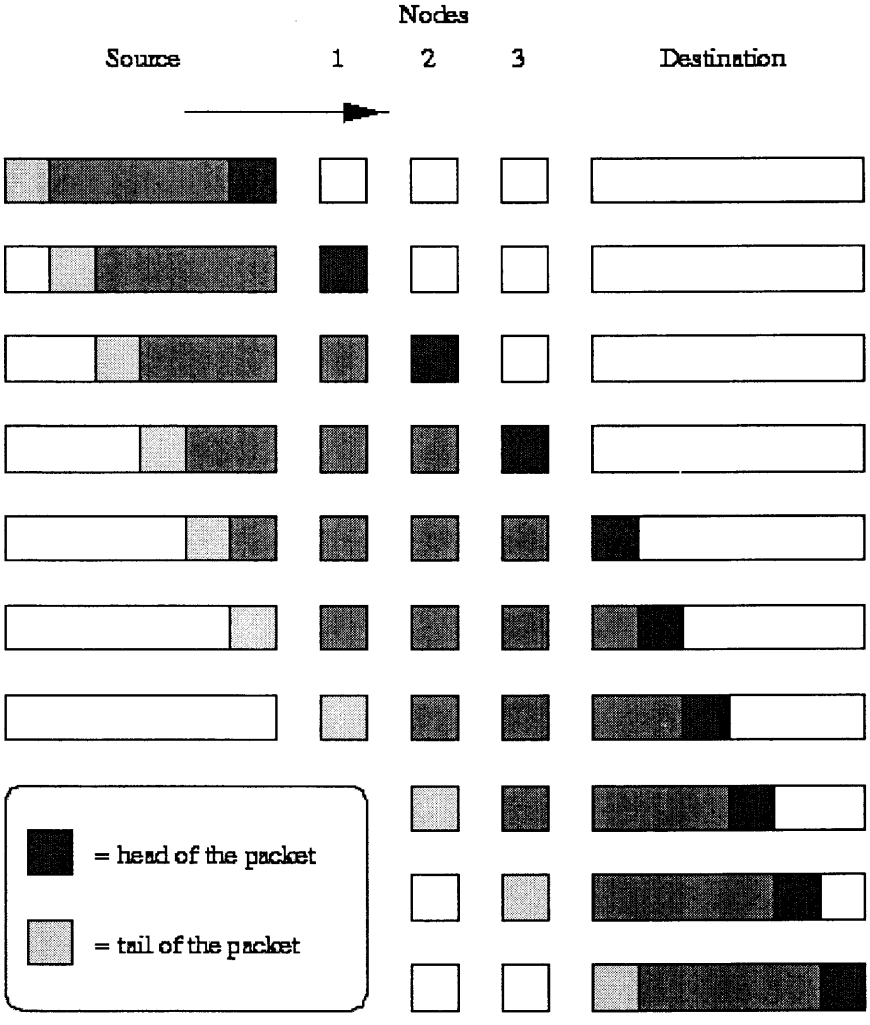

Figure 2.1 Wormhole Routing in a Network.

Furthermore, each node on the path of the packet can contain only one flit of that packet and will always try to transmit it to the next node. Because the movement of the packets resembles the movement of a worm, the packets sometimes are called worms themselves. An illustration of how a packet moves through a network is given in Figure. 2.1

In wormhole routing the following is done:

1. The source node starts transmitting the flits one by one to the next node, beginning with the head of the packet.
2. The nodes on the path of the packet receive the flits, but can store only one flit at a time. If a flit is present in a node, other flits cannot enter this node, which means that the previous node cannot send its flit.
3. A node always tries to send the flit which it receives on to the following node, unless the node itself is the destination node.
4. The destination node receives the flits one by one until the last flit of the packet has been received.

It must be noted that, in wormhole routing, it is impossible for flits of another packet to cross the path of the current packet. This means, that even if a node is empty, flits of the other packet cannot use this node, unless the last flit of the current packet has passed it. This characteristic of wormhole routing introduces a problem called deadlock.

2.2 Deadlock in a Network

Deadlock in a network occurs when packets are blocking each other, thus effectively preventing them from moving to their desired destination.

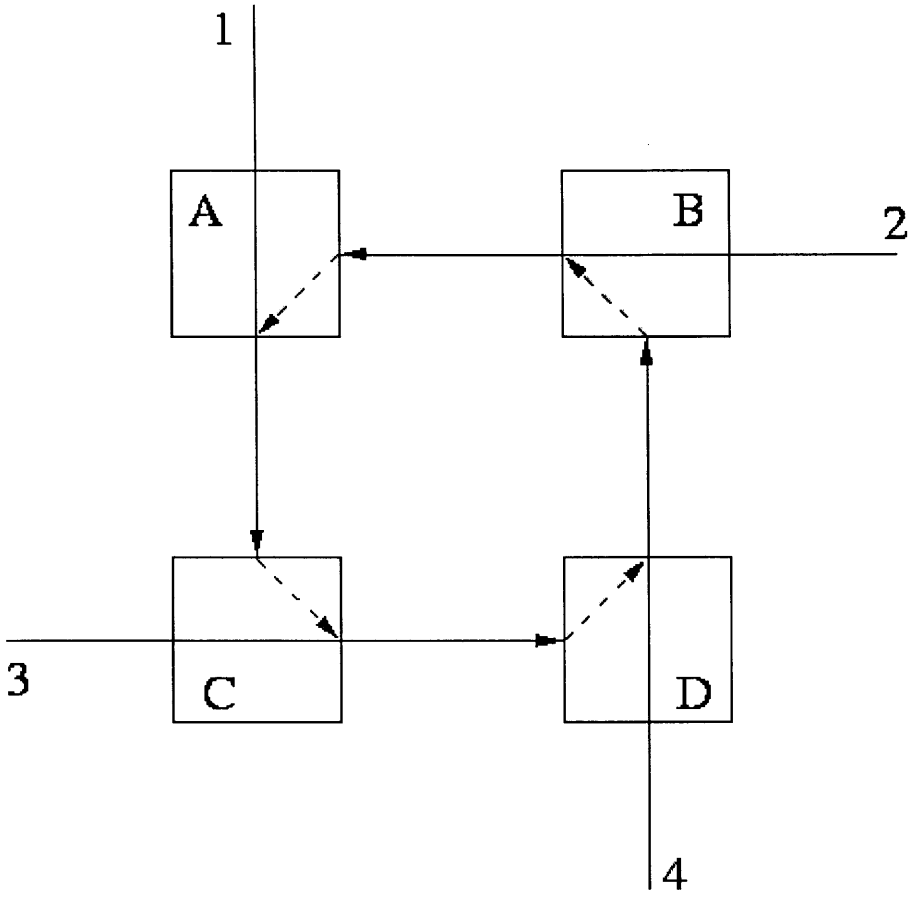

Figure 2.2 Deadlock in a network.

In this Figure. 2.2, packet 4 tries too make a left turn at node B, but is being blocked by packet 2, which tries to make a left turn at node A and is being blocked by packet 1 and so on. It becomes clear, that these four packets are stuck and cannot move, until something radically is being done, like randomly removing one of the four packets. However, this cannot be the solution to the problem.

The packets in Figure. 2.2 block each other, because they all are using the same kind of channel. A possible solution to the problem in Figure. 2.2 is given in Figure. 2.3.

In this Figure, the problem is being solved by introducing two kinds of channels. First, there are the channels (dotted), which will be used by packet, which have made a turn, and second, there are the channels (solid), which will be used by packets, which do not have made a turn yet. Because the packets in the nodes A through D are assigned to different kinds of channels, they will not block each other any longer.In the example above, the solution was to introduce a new kind of channel, namely, the one if a packet has to make a turn. However, the previous solution only works, if other parts of the network can be neglected, and, if the following assumption holds: the packets in the network only have a maximum of ONE turn to make.

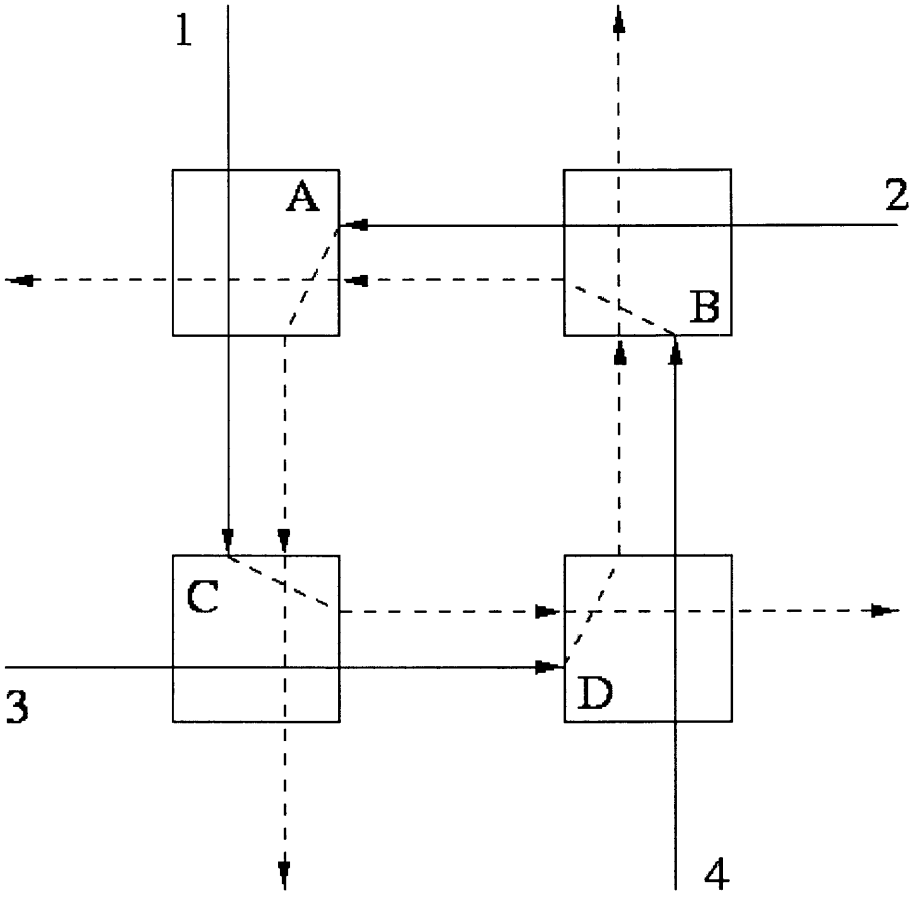

Figure 2.3 Possible solutions for avoiding Deadlock [3].

2.3 Routing Strategies

In an intercommunication network, routing algorithms that are used for determining the path to the destination node can be classified according to their:

- Number of destinations- **Unicast:** packets may have a single destination, Multicast: packets may hay multiple destinations
- Place where routing decisions are taken- **Centralized**: by centralized controller, Source: by the source node, Distributed: determined in a distributed manner while the packet travels, Multiphase: hybrid, source node computes some destinations, path established in a distributed manner. **De-centralized**: The source and destination is not pre-determined.
- Way of implementation- Table-Lookup: looking at a routing table, **Finite-State Machine:** executing a routing algorithm in software or hardware according to a finite-state machine
- Adaptivity- Deterministic: always supply the same path between a source/destination pair,Adaptive: use information about network traffic and/or channel status to avoid congested or faulty regions of the network. **Non-deterministic:** can be of any path
- Progressiveness- **Progressive**: move the header forward, reserving a new channel at each routing operation, Backtracking: allow header to backtrack, releasing previously reserved channels (used for fault-tolerant routing)
- Minimality- **Profitable (minimal)**: supply channels that bring the packet closer to its destination, Misrouting(non-minimal): may also supply channels that send the packet away from its destination
- Number of alternative paths- **Fully Adaptive, Partially Adaptive**, N**on -Adaptive.**

2.4 UP/DOWN routing technique

In Up/Down routing, one of the nodes is chosen arbitrarily as the root of a tree and all links of the topology are designated as `up' or `down' links with respect to this root. The `up'/`down' state of a link is relative to a spanning tree computed in the background by a distributed algorithm. A link is `up' if it points from a lower to a higher level node in the tree (i.e, to a node at a lesser distance from the root). Otherwise, it is `down'. For nodes at the same level, node IDs breaks the tie. The routing from a source to a destination is done in such a fashion that zero or more `up' links (towards the root) are traversed before zero or more `down' links are traversed (away from the root) in order to reach the destination. This prevents circular waits and thus the routing is deadlock-free.

Myrinet runs a `mapping' algorithm that computes the current topology in the background. Myrinet uses up/down routing to build these paths [9]. Although the original distributed up/down routing scheme provides partial adaptivity, in Myrinet only one of the routes is selected to be included into the routing table, thus resulting in a deterministic routing algorithm. On the other hand, many paths provided by the up/down routing are non-minimal on certain networks. Myrinet uses up/down routing to build network routes. Up/down routing is based on an assignment of direction to the operational links. To do so, spanning tree is computed and then, the "up" end of each link is defined as:

- The end whose switch is closer to the root in the spanning tree;
- The end whose switch has the lower ID, if both ends are at switches at the same tree level.

The result of this assignment is that each cycle in the network has at least one link in the "up" direction and one link in the "down" direction. To eliminate deadlocks while still allowing all links to be used, this routing uses the following up/down rule:

A legal route must traverse zero or more links in the "up" direction followed by zero or more links in the "down" direction. Thus, cyclic dependencies between channels are avoided because a message cannot traverse a link along the "up" direction after having traversed one in the "down" direction.

Up/down routing can supply several valid routes between two network hosts and, in some cases,there exist more than one shortest up/down route. Therefore, it is stored in the table a set of routes for each source-destination pair, including the best ones, and selecting one of them according to some criteria [5].

The following path selection algorithms:

- *OSUD (One Shortest Up/Down path)*: Always the same shortest up/down path. This is the current routing policy used in Myrinet.
- *RSUD (Random Shortest Up/Down path)*: Randomly selection among all the shortest up/down paths.
- *RRSUD (Round-Robin Shortest Up/Down path)*: Round-robin selection among all the shortest up/down paths.
- *PUD (Probabilistic Up/Down path)*: 80% of selected paths using RSUD, and 20% randomly chosen among up/down paths that are one hop longer than the shortest up/down path. Even though Up/Down routing technique is being used nowadays and is proved advantageous in regular networks, it fails to eliminate deadlocks when used in Wormhole networks.

CHAPTER 3

TPBR ALGORITHM AND ITS MATHEMATICAL MODEL

3.1 The Algorithm

In wormhole routing, a router begins forwarding a packet as soon as the header is received and the required channel buffer in the next router can accept one or more flits of the packet. Flits are transmitted from one router to the next in a pipelined fashion and may occupy several channels along the path from source to destination. Only the header flit of a packet contains information required for routing. If the header flit is blocked because the required buffer in the next router along it's path is full, all of the flits in the packet are blocked. Design of efficient deadlock-free routing algorithms in irregular topologies introduces new challenges.

The steps involved in the algorithm are:

1) Finding out the number of turns in the network.

2) Prohibiting some of the turns in a network over the nodes, this still can make the tree connected.

3) Drawing a spanning tree based on the nodes through which turns are not eliminated.

4) Getting the route from source to the destination.

Now let us consider the prohibition of the some of the turns in a network. Here it is assumed that a network consists of N nodes connected by E edges, constituting a graph G. A node is considered as the router component of the processor-router pair In order to tackle the problem of deadlock, the TPBR algorithm is used to eliminate the number of turns in a network. Every routing algorithm prohibits some of the turns in network graph . A turn (a,b, c) is a three-tuple of nodes such that (a, b) and (b, c) are edges in the

network graph *G*. In order to correctly model existing switch based networks such as Myrinet, it is assumed that *G* is symmetric, i.e. if (*a, b*) is an edge in *G*, then (*b, a*) is also an edge in the network.

At the first step, a node with the minimal degree is selected and labeled as node1. If after deletion from *G* of node1 and all edges neighboring node1, the remaining graph *G*-1 is still connected, then all d (d -1)/2 turns like *(a,* 1, b) are prohibited and all turns (1, *b,* c) are permitted where d is the degree of the node. If after deletion from *G* of node1 and all edges adjacent to node1 the remaining graph *G*-1 consists of disconnected sub graphs $G_1 \ldots G_S$, (this procedure is used also for s=1) then select nodes *a*1... *as* (called *tree* nodes) such that *ai* is a node of G_i and *(ai,* 1*)* is an edge in *G.* All edges *(ai,* 1*)* are added to the spanning tree. All constructed tree nodes, except one, are added to the set of *basic* nodes *B* (initially B=0). At the next step the procedure is repeated to the remaining graph G-1, labeling non-basic node with a minimal degree by 2. At each step, basic nodes are selected in such a way, that every component of connectivity has only one basic node. Process is completed, when all nodes are labeled.

Implementing TPBR algorithm in a distributed way which is a requirement of today's' network equipments comes into two aspects. Firstly start the turn prohibition from the node which has minimal $(d_a^2-2)/\sum (d_i-1)$ where d_a is the degree of the node that is to be considered d_i is the degree of the i^{th} surrounding node. However, each node does not have any idea on the degree of other nodes in the network. Secondly each node does not have a big picture of the topology of the network, therefore, it is unable to know if a turn is removed from that node, the rest of the network is still connected or not.

To address these problems, a protocol based on the TPBR algorithm is developed, which involves three steps.

Step 1: *Calculation of* $(d_a^2-2)/\sum(d_i-1)$.

a. Each node broadcasts its degree to its neighboring nodes.

b. When a node receives the degree value from other nodes, it stores it.

c. When a node gets knowledge of all the nodes neighboring to it, then it will perform a sorting algorithm and determine which turn has to be prohibited.

Step 2: *Turn prohibition unit.*

a. Each time there is only one node in the entire network, which is trying to determine its prohibited turn.

b. Components of connectivity are constructed in the graph without the selected node. Then it makes an edge special if any, by checking if there is a discontinuity in the graph. The node, which is connected to special edge other than selected node, is marked as special node.

c. TPBR algorithm doesn't select the special node.

d. Likewise the algorithm is carried out recursively until all the nodes are evaluated.

Step 3: Routing table construction.

a. In our TPBR protocol, the Bell-Ford algorithm is extended because it has better decentralized feature than the Dijkstra's algorithm.

b. After each node determines its prohibited turn, then a routing table is constructed. This can be achieved by extending any routing algorithm such as Dijkstra's algorithm or Bell-Ford algorithm. The prohibited turns should be take into account, therefore some paths

cannot be taken even though they have a shorter distance, due to the existence of prohibited turns.

3.2 Mathematical Model

Assume that the original network consists of N nodes, connected by E edges. Also assume that all nodes are connected (for any two nodes there exists a path between them). In the general case network graph G can be considered as a multigraph with several edges between two nodes. In particular, if V virtual networks are used, any two nodes are connected either by 0, or by V edges.

The total number of turns in $T=\sum(di\ (di - 1))/2$, where di is a degree (number of neighbors) of node i. For example, for the up/down routing a spanning tree for G is constructed, nodes are labeled preserving the partial order defined by the tree (the root has label 1) and turn (a,b,c) is prohibited if b>a and b>c.

It is seen that the results on lower and upper bounds on fractions z=z(N,E) of turns which have to be prohibited to prevent deadlocks in a given network graph [2,3,7]. The proposed upper bound is constructive, i.e. its proof generates a simple algorithm (its is called z-algorithm) for construction of a tree and labeling of nodes by $1,2,...,N$ such that turn (a,b,c) is prohibited iff at least one of the edges (a,b) or $(b,$c) does not belong to the tree and $a>b$ and $c>b$.

Denote by Z(N,E) a minimal fraction of prohibited turns for prevention of deadlocks in network graph G with N nodes and E edges. Let di be a number of neighbors (degree) of node i and T be the total number of turns 2E = S(i=1 to N)d(i),

T=S(i=1 to N)d(i)*(d(i-1)/2

The following lower bounds for Z(N,E) [10] will be useful to estimate performances of the routing strategies that will be described later. First note, that *Z(N,E)≥ (E-N+1)/T (Bound A).* This bound follows from the fact that there are b = E-N+1 linearly independent cycles in G and each one of these cycles has to contain at least one prohibited turn to prevent deadlocks.

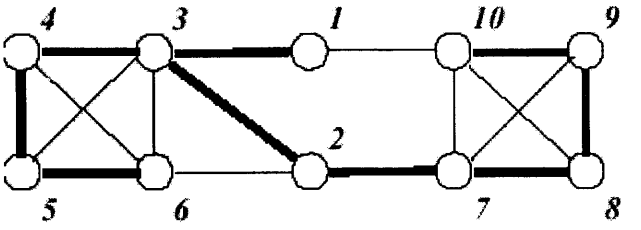

Figure 3.1 Network showing the connectivity to show continuity.

For the example shown in Figure. 3.1 it can be seen that N=10, E=17, T=44 and by Z(N,E)>=8/44. Let C={C1, ...*C*R} be a system of cycles in *G* and *m* is a maximal number of cycles from *C* containing the same turn[1,2].

Then Z(N,E) ≥ R/*mT (Bound B).* If m=1, then Z(N,E) ≥ R/T, where *R* is a maximal number of cycles in the network graph such that every turn belongs to at most one cycle. one can select as *C* the system of all 9 triangles and one cycle of length 5 (R=10). In this case cycles don't have common turns (m=1), and by Z(N,E)≥ 10/44. Bound A is useful when a number of cycles in *G* is small and Bound B can be used for a networks with large numbers of cycles.

Let us now consider graph *G*=(*V,E*), with $|V|=N$ nodes (denoted by *a,b,*...), connected by $|E|=M$ edges (denoted by (*a,b*) etc.) Considering the non-oriented case, so (*a,b*)=(*b,a*). Without losing generality, assume that all nodes are connected (for any two nodes there exist a path between them). If this is not the case, also consider all

components of connectivity separately. A *path* $P=(v_0,v_1,\ldots,v_L)$ of length L ($L\geq 1$) from node a to node b in G is a sequence of (not necessarily different) nodes $v_i \in V$, such that $v_0=a$, $v_L=b$, and every two subsequent nodes are connected by an edge, and every *ordered pair of subsequent nodes* appears exactly once. Length L of the path is the number of edges, included in it. Set W of turns in G is called *cycle breaking* (11,12,13) if every cycle in G is covered by at least one turn from W. A solution is presented below a for the problem of finding in a given graph a minimal cycle-breaking set of turns. Cycle-breaking set of turns W is called *irreducible*, if there are no cycle-breaking subsets of W, which are not equal to W. (Also note that not every irreducible cycle-breaking set is a minimal one).

For example, for 2*d* $p\times p$ meshes $N=p^2$, $M=2(p-1)p$,

$$T(\boldsymbol{G}) = 6(p-2)^2 + 12(p-2) + 4,$$

$z(\boldsymbol{G}) \geq \dfrac{p^2-2p+1}{6p^2-12p+4}$, and the lower bound for $z(G)$ in this case is close to 1/6 for large meshes [1]. (Note that for the North-Last algorithm and 2-d meshes the fraction of prohibited turns is equal to lower bound 1, which proves optimality of the North-Last algorithm.)

For the example as in Figure. 3.1, $N=10$, $M=17$, $T(G)=44$ and by bound 1, $z(G)\geq 8/44$.

If $r=1$, then according to bound 2, $z(G) \geq R/T(G)$, where R is a maximal number of cycles in the G with disjoint sets of turns. For the example shown in Figure. 3.1 one can select as C the system of all 9 triangles and one cycle of length 5 ($R=10$). In this case cycles have no common turns ($r=1$), and by bound 2, $z(G)\geq 10/44$.

Bound 1 is useful when a number of cycles in G is small and bound 2 can be used for graphs with large numbers of cycles. To generalize bound 2, Denote for system C of R cycles by r_j, the number of cycles covered by turn j and assume that $r_1 \geq r_2 \geq \ldots \geq r_T$. Then $Z(G)$ should satisfy the following condition: $\sum_{i=1}^{Z(G)} r_i \geq R$ [3]. By this it can be said that the TPBR algorithm gives us a optimal set of prohibited turns which will be good enough to keep the network deadlock free. This model helps us to determine the set of turns, but the set of turns got is not irreducible but is optimal.

CHAPTER 4

ARCHITECTURAL DESIGN OF THE CHIP

4.1 Overview of the System

The proposed VLSI architectural design of the whole chip is discussed in this section. Figure. 4.1 gives an overview of the whole system, which can be considered as a single node in an irregular network..

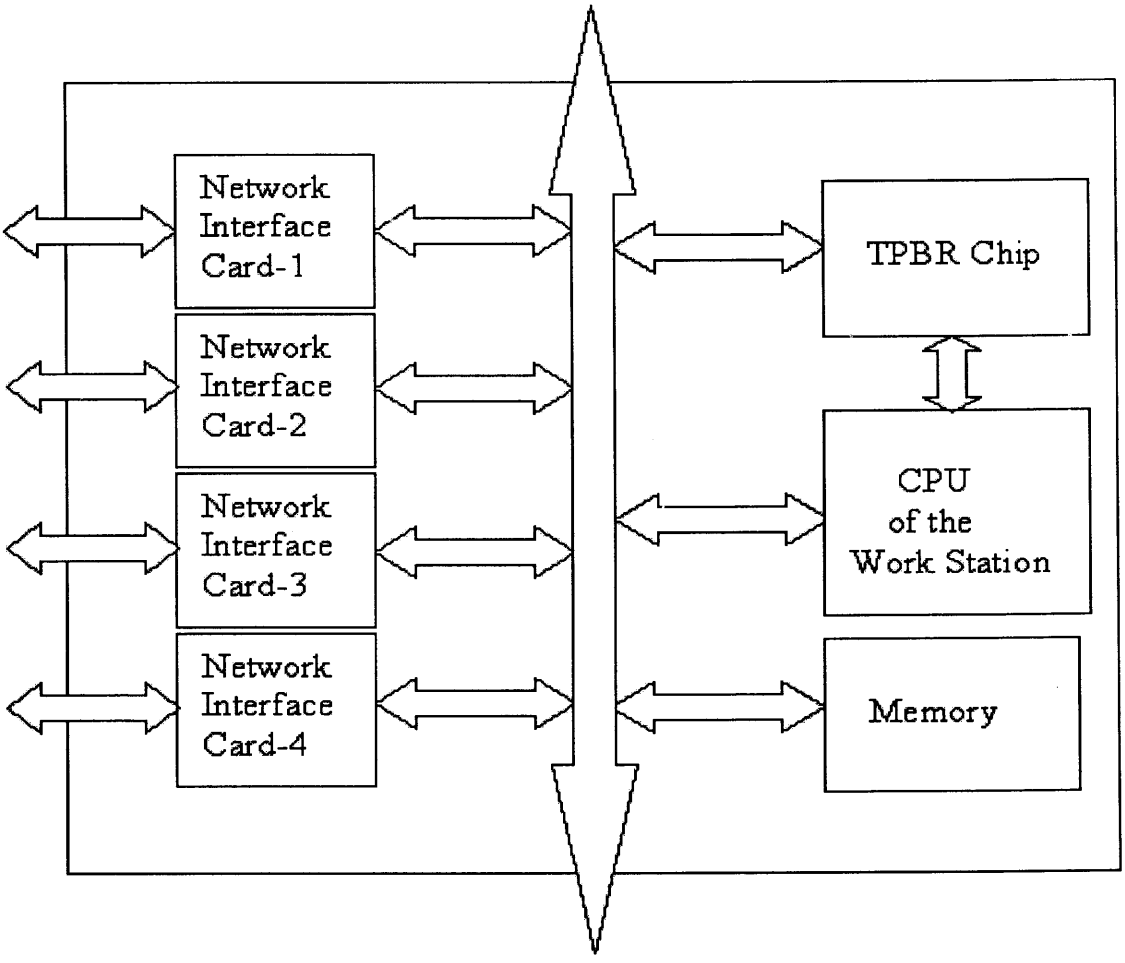

Figure 4.1 Overview of the Workstation in a Network.

As 4-port router is taken into consideration, therefore four network interface cards for each router is taken, which means that four systems can be connected to each of the workstation thus enabling each of the workstation to perform in a parallel fashion thus using the concept of the parallel multiprocessors. The Central Processing Unit (CPU) of the workstation is linked to the Turn Prohibition Based Routing (TPBR) chip, which decides the route the packet should take from the source to the destination keeping the prohibited turns into consideration.

The TPBR chip is responsible for running the proposed TPBR protocol. When the protocol running is finished, the routing table will be stored in the memory. Then CPU can utilize the routing information to receive and transmit message in the networks. A typical network composed of such routers is shown in Figure. 4.2

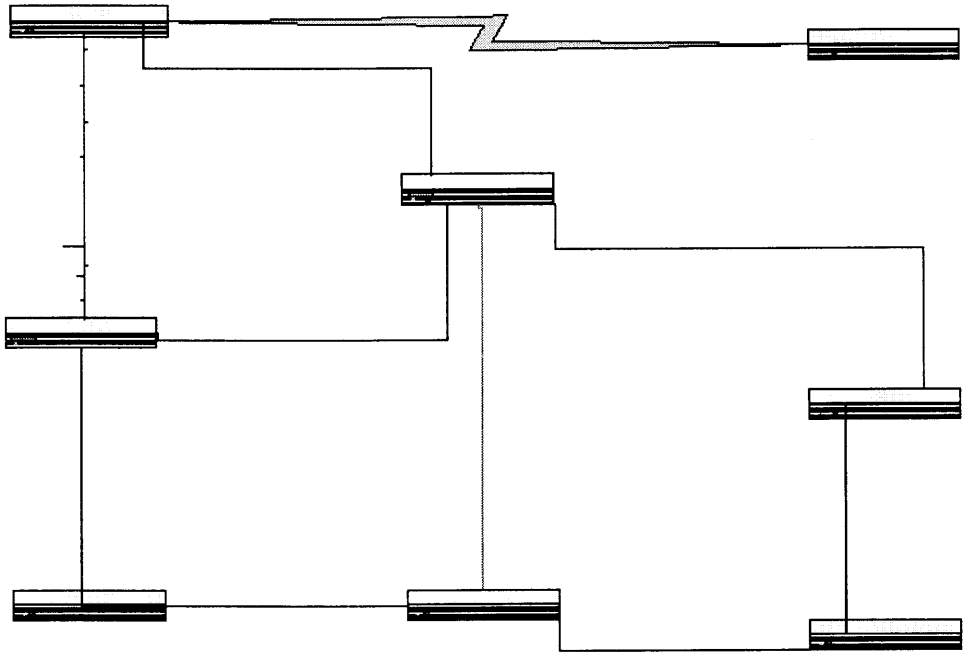

Figure 4.2 A Network composed of TPBR routers.

4.2 Architecture of the chip

This chip is the controller of the router, so this chip has data bus, address bus, clock signal and some control signals such as reset, chip select (cs), interrupt signals (int_req) and interrupt acknowledge signals (int_ack), read and write signals (rw) as shown in Figure. 4.3. In order to get the local information of the decentralized router, the chip has port status input signals, it can be used to detect which port of the router, (also assume each router has four ports) is connected with other neighbor routers. Since each router has its own node id which is used to be identified by itself and located by others. Assign 4 bit input signal for node_id (3..0). Reset signal is used only during initialization or system failure. Figure. 4.3 demonstrates the architecture of the TPBR chip, which is mainly divided into three sub parts, which individually takes care of a specific function for determining the prohibited turns and thus making the network deadlock free.

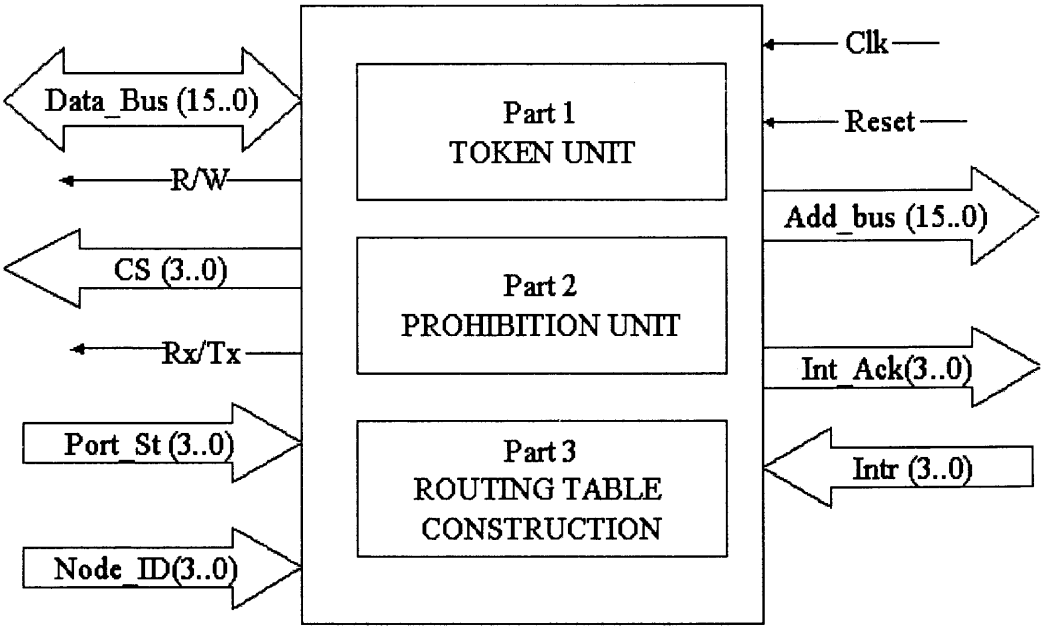

Figure 4.3 Architecture of the chip.

These three parts are invoked sequentially one after the other starting from the token unit then the prohibition unit and finally the routing table construction unit. After the process is completed then the routing table will have the route from each particular source to all the other nodes in the network.

4.3 Token Unit Design

The architecture of the token unit is shown in Figure. 4.4. In protocol step 1, each node needs to claim its degree in the network. This information needs to be broadcasted in the entire network. This is the important step in the process of Turn Prohibition, as each node should be aware when and on what all nodes the TPBR has to be run. By giving each node an identity, the nodes will run the algorithm in the way they were assigned the numbers. Only the 'node 0' should be able to claim its degree in a packet type so that is set as 1. Other nodes simply wait for the first packet. The data structure of data field in type 1 packet is shown in Figure. 4.5.

Each node in type 1 packet has a fixed slot for it to claim its own degree information. Whenever it receives a type 1 packet, it checks whether every node has claimed its degree in the packet. This is accomplished by checking the *counter* field, which is incremented by the node when it provides its degree information in its slot as shown in Figure. 4.5. The *W* bit is used to check whether a node has claimed its degree in this packet. If every node has claimed the degree in the packet, the node that receives the packet will check the *R* bit in its slot. Based on the functionality of the token unit, it is divided it into several sub-parts, such as interrupt_unit, port controller, core_logic unit, transmit unit and receive unit as shown in Figure. 4.4. Core_logic unit is an essential unit

of this token unit. It is used as the main frame of the design. The transmit and receive process is used to send and receive the packets which will be broadcasted in the network.

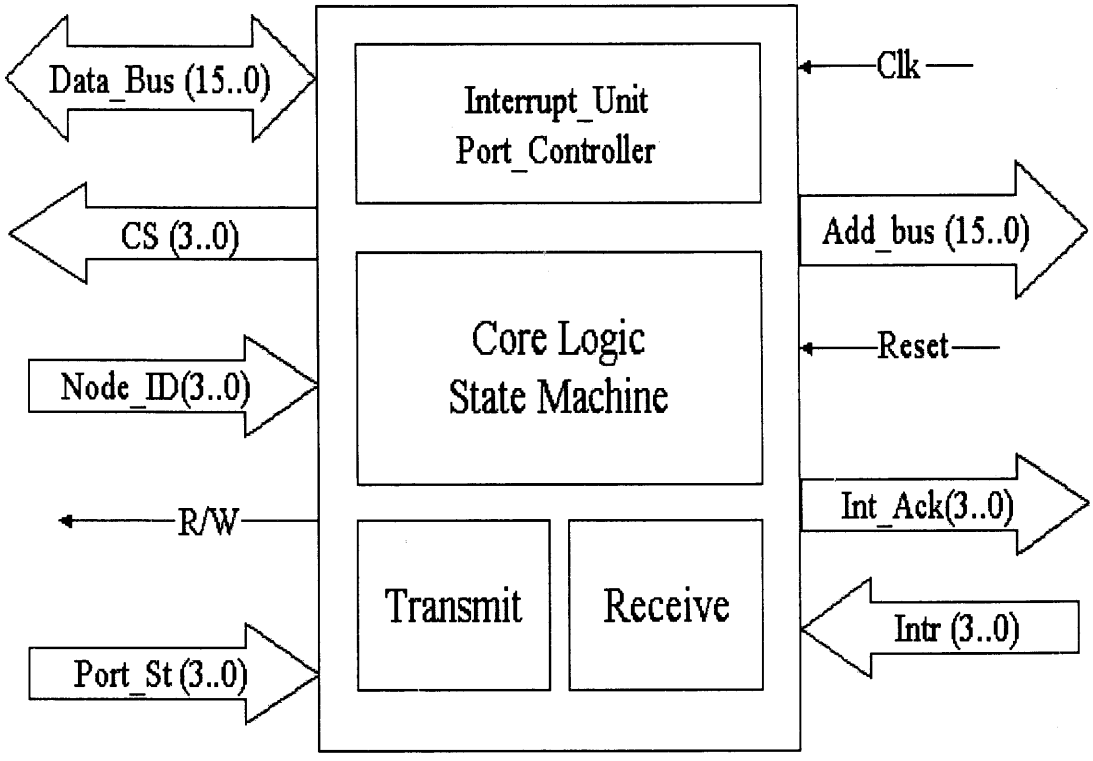

Figure 4.4 Architecture of the Token Unit.

The above protocol design for the Token Unit can be seen in Figure. 4.7 in the form of state transition diagram. In startup state, the chip will be initialized, which includes initializing the registers inside the chip, getting the node_id and also the port_status. In trans_ini state, a packet will be formed according to the format of the packet, which is 16-byte long, and then transmitting it to all the nodes, which are connected with 'node 0'. Only 'node 0' will enter this state. In trans_spec state, after receiving a packet whose type code is 0, the state machine will enter the trans_spec state. It will transmit a packet to other nodes. In the transmit state, it is in charge of transmitting

packets to the outside node. In the waiting state, the node will simply wait for an incoming packet and when a packet comes, check its type. If packet is type 0, it will go to trans_spec state; if type1, it will go to Receive state. In the Receive state, the process will perform the data field check and perform the protocol as described in Figure.4.5 and Figure. 4.6.

Counter (8 bit)	Degree of Node 0	Degree of Node 1	Degree of Node 2	...

Figure 4.5 Structure of a data field in a type 1 packet.

R (Read) (1 bit)	W (Write) (1 bit)	Degree of the Node (2 bits)

Figure 4.6 Data structure of each slot in a packet.

In the Token Unit, the Interrupt Unit controls the four interrupts from the outside port and generates the interrupt signal to the receive unit and also generates port number for receive unit. It receives the interrupt_ack signal from the receive unit. After interrupt unit gets the interupt_ack signal it begins to wait for a new interrupt from the other port. The Receive Unit gets signal from Interrupt unit and outputs the signal and data to Core Logic unit. After getting the interrupt signal from interrupt unit, receive unit checks which port has data and controls the address bus, data bus , r/w and cs signal to receive data from that port. After receiving data it generates the receive_data and port number signal for the Core Logic and then waits until Core Logic unit sends the rxd_ack signal

back to receive unit, and sets the mode to active. Transmit Unit get signal from Core Logic and transmit data to the outside port. After transmitting the data it gives back a txd_ack signal to the Core Logic. After the process is done then the transmit unit will wait for new data to transmit.

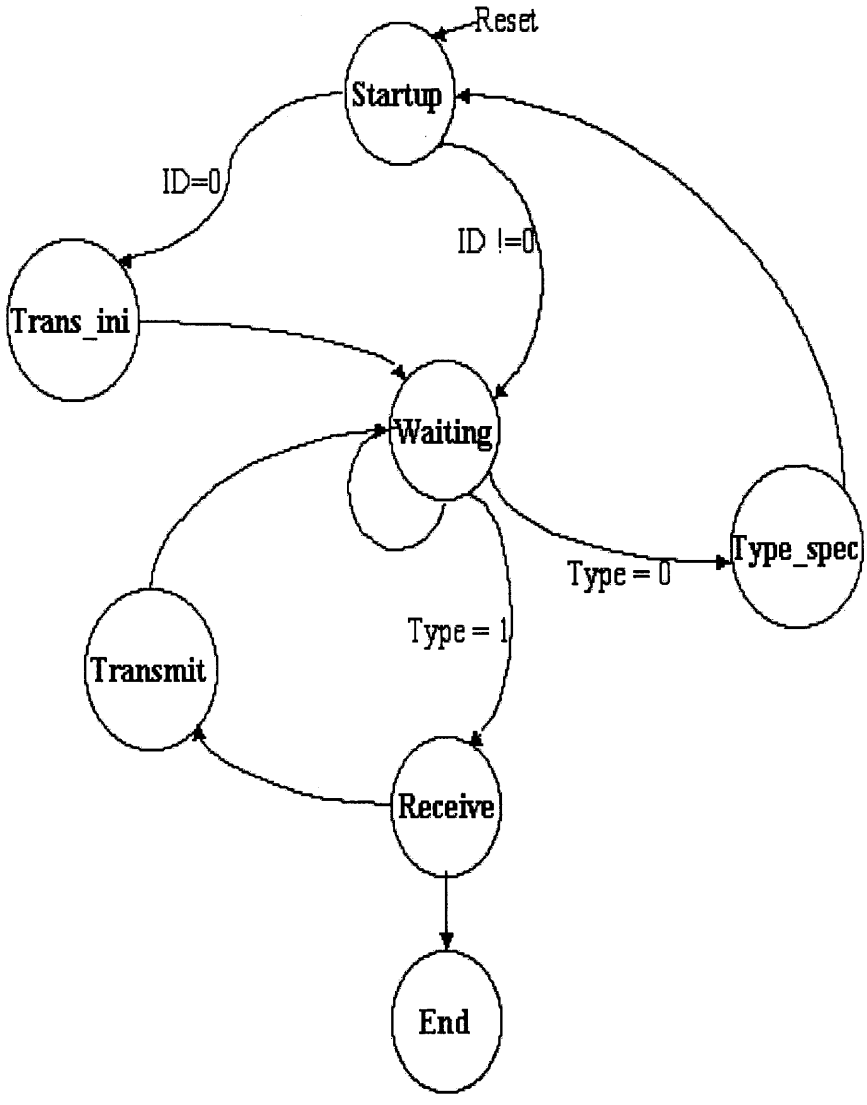

Figure 4.7 State Transition Diagram for the Token Unit.

4.4 Turn Prohibition Unit

In this block, as in the case of most routing algorithms certain turns in the network graph are prohibited. Consider a turn (a, b, c). The turns (a, b, c) and (c, b, a) are prohibited if some other path exists between nodes 'a' and 'c'. To implement this, a flat packet is broadcasted from all the ports of the node. Components of connectivity are constructed in the graph without the selected node. Then it makes an edge special if any, by checking if there is a discontinuity in the graph. The node, which is connected to special edge other than the selected node, is marked as special node. As seen in the Figure. 4.8, this block gets the 'enable' signal and 'token' from block1, which is the Token Unit. The final output of this block is the P-matrix, which is written to the memory.

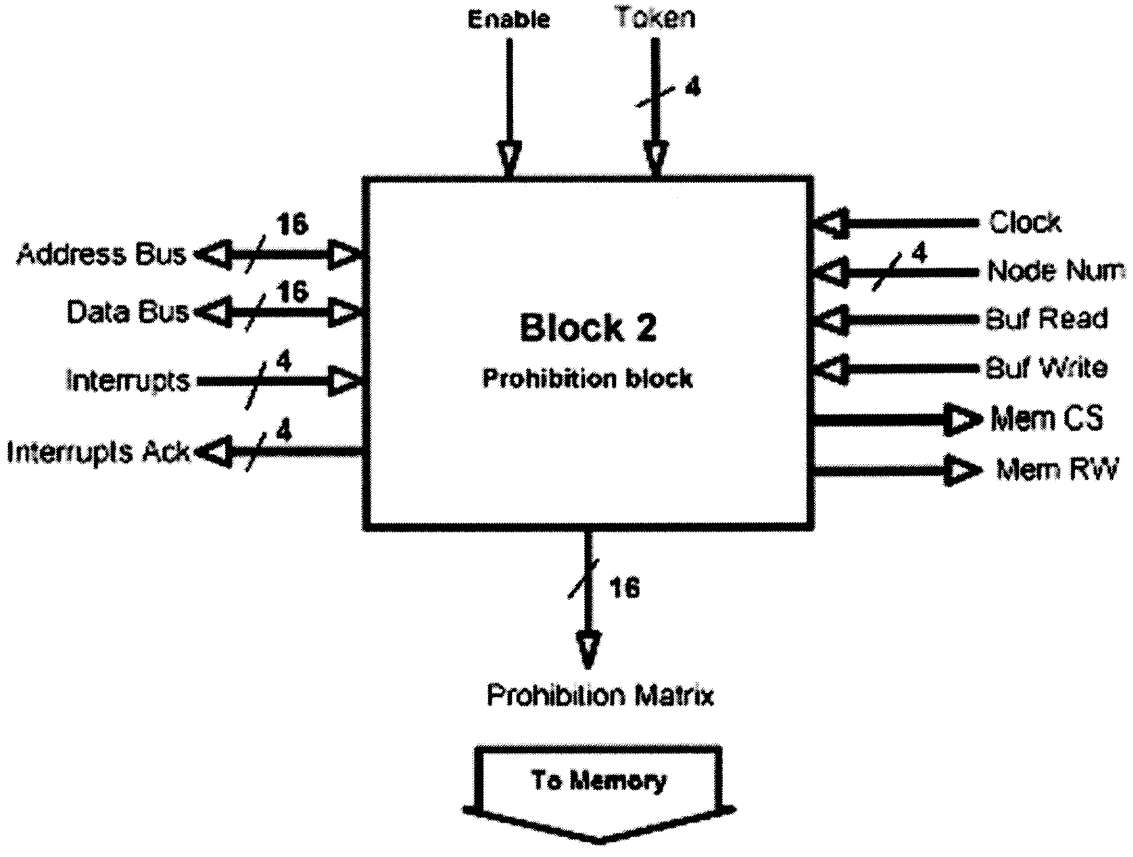

Figure 4.8 Entity of the Turn Prohibition Unit.

This block also has interface with the 'address' and 'data' buses for reading and writing data. The 'interrupts' and 'interrupts ack' ports are for synchronizing this block with the IO drivers. This block gets its node number from the 'node num' port. The 'buf read' and 'buf write' will determine the type of IO operation to be performed.

TPBR algorithm doesn't select the special node. After a node recognizes that one of its neighbor is a special node, then it broadcasts a packet to that node indicating the node is special, and then the node will not run the TPBR algorithm. If the received packet is the token release packet from the node with the preceding token then it is checked whether the broadcasting of the flat packets is done previously. If not, it is completed and will wait again for the interrupts.

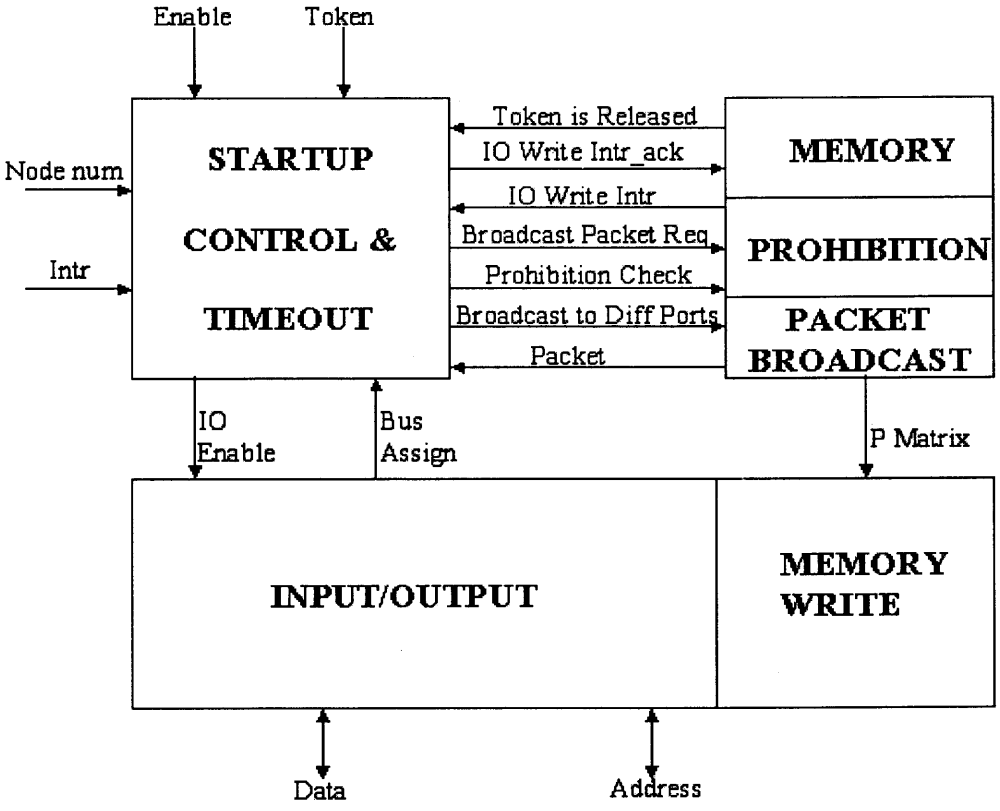

Figure 4.9 Architecture of the Turn Prohibition Unit.

As seen in Figure. 4.9 the Turn prohibition unit consists of 3 main parts start & control, the turn prohibition unit, I/O memory block. The startup unit receives the 'enable' and the 'token' signal. When 'enable' is 'high', making 'prohibit_enable' high turns on the whole block. Making 'prohibit_enable' low after the token is released again turns off the whole block. If prohibit_enable is '1', broadcast packet is request by making broadcast_packet_req 'high' and the flat packets are broadcasted by the appropriate process. After the flat packets are broadcasted the timeout process is enabled and waits for a time of 6 I/O operations and after the time is elapsed the 'timeout' signal is made 'high'. If after the IO is enabled, the interrupts are checked and the one with the highest priority is serviced first. The same process is used for memory write after the token release packet is broadcasted that is the token_relsd is 'high'. The third block of the Turn Prohibition unit is the core unit of the whole chip, which mainly deals with the Turn Prohibition. This block has three sub-parts, one is the packet broadcast part, which broadcasts the appropriate packets in response to the broadcast packet request, and broadcasts to different ports depending upon the request arrived. The second sub-part is the prohibition part, which runs the TPBR algorithm so as to find the turns to be prohibited. In the process of eliminating the turns, at some time the algorithm recognizes some of the nodes to be special which means that the node which is special, should not run the TPBR algorithm. In order to keep track of the special nodes identified by a particular node (system), a memory module is used which stores the identity of the special node. After the whole process of Turn Prohibition is completed, the special node identity stored in the memory is recalled and is intimated to that corresponding node, so that the special node does not run the TPBR algorithm.

4.5 Routing Table Construction

In this section, a decentralized algorithm is described for the construction of local routing tables (for a given set of prohibited turns Z (G)) minimizing average path length and average delivery time. Assume that Z (G) is already constructed. For a given source *s* and destination *d* our goal is to select the shortest routing path a1...*am* (a1=s, *a*m=d) among all paths, satisfying the routing restrictions (not including turns from Z (G)) . For any intermediate node *i* the algorithm estimates the length of the shortest permitted path between neighbors of *i* and the destination, and routes the message to neighbor *j* with the lowest estimated path's length (providing that the corresponding turns in *i* and *j* are permitted). Assume that every node has up to *d* neighbors ("node" here is the router component of the processor-router pair). Hence, it would have up to d+1 input/output buffer, including the buffers for the consumption channel to the processor.

Initially, every router knows the set of turns, which are permitted, and which are prohibited in this router. After the implementation of the TPBR algorithm, all the nodes in the network broadcasts their turns which are either permitted or prohibited through them to all the other nodes in the network by flooding a packet.

4.5.1 Mathematical model for Table Construction

This can be represented by (d+1)×(d+1) matrix P, such that P (*i*, j)=1 if the turn from input buffer *i* to the output buffer *j* is permitted and *i*{0,...,d} (case *i* (or j) =0 corresponds to the consumption channel). It follows from the *TPB*R-algorithm that matrix P has the following properties:

1) It is symmetrical, i.e. P(*i* ,j)= P(*j* ,i);

2) P (0,i)=1, P (i, 0)=1.

For every node routing matrices R(*i*,k) and D(*i*,k) are constructed, where i{ *0*,...,d}, k{*1*,...,N}, *N* is the number of nodes. R (*i*, k) = j, if a message, coming from input port *i* to destination node k, should be routed to output port j. Elements of *R* take values from 0 to d. D (*i*,k) is the length (number of hops) on the path from input buffer *i* to destination node k. Matrix *D* is used at the pre-routing stage only, while matrix *R* is used for on-line routing. The total memory required to store these matrices is of the order of 2(d+1) N.For d=4, N=1,000 it is around 10K; hence a hardware implementation for this algorithm is feasible. R_a and D_a for node *a* are initialized as: $R_a(i,j)=X$, $R_a(i,a)=0$, $D_a(i,j)=X$, $D_a(i,a)=0$. (Assign *X* as a special value, corresponding to the undetermined case) At each step, elements of R_a and D_a are recalculated, using matrices R_1...Rd, D_1... D_d of neighbors*1*...*d* of node a. After *t* steps all paths of length up to *t* hops will be determined. The rule for step *t* (initially, t=1) is the following:

If $R_a(i, j) = X$ *then for all m, such that P (i, m)=1 (the turn from i to m is permitted, node m is a neighbor of node a)*

If $D_m(y,i)=t-1$, *then* { $R_a(i,j):= m$; $D_a(i,j):=t$ } *(Here y is the input port for node m, which corresponds to the neighboring node a).*

For the hardware realization, at each step *t* every node should transmit to its neighbors, messages with numbers i, such that $D_m(y, i) = t-1$. During the whole pre-routing procedure, up to *N* such numbers can be sent by every link. The algorithm is terminated after *L* steps, where *L* is the maximal possible length of a minimal permitted

path between two nodes, or if at some step *t* no changes have been made in any of the matrices.

It is noted that the proposed algorithm can be used to construct a set of shortest paths for any given set of prohibited turns, which will increase the efficiency of the network. The objective is to find the shortest path from any node to any node in the network. To calculate this the following assumptions are made.

1) It is assumed that the links between nodes are full duplex, which means that there exists a two-way communication between the two nodes.

2) Data can be back routed; i.e., if data is received on port, it can be transmitted back on the same port.

3) A P(i,j) vector will be formed after the execution of the second block, if P(i,j)= '0' then the turn is prohibited.

4) Each node will be labeled and each node as an entity knows its' own label which is unique in the network.

For examples, each port will be labeled with a number and each Node with a letter. Port 0 is reserved for the local machine. Now let us consider the steps for finding the shortest path taking the above assumptions into consideration.

1) Initialize arrays 'D (5x16)' and 'R (5x16)'and zero counter. Through reset signal

   a) A counter is initialized to a count of "0".

   b) Values in 'D (i, a)' 'R (i, a) are set to '0', where 'a' is the node ID. This is the local machine. All other elements for Array 'D' and 'R' are initialized to 255. This assumes 8-bit value, for implementations sake.

Notes: For example, node B will set 'D (i, B)' and 'R (i , a) = '0'. Array 'D' is the distance vector. Array 'R' is the routing table vector. The wait period is determined by network latency.

2) Advance time counter by 1, i.e., ' t = t+1'

3) Update 'R (i, k)' using information from 'D (i, k)'. If 'D (i, k)' = t-1, 'P (i, j)' = '1' and 'R (i, k)' = '255' then write 'i' into 'R (i, k)'.

4) Make only ONE write to any location in array 'R'. This is accomplished by checking for the default state. The first write made to 'R' is the shortest possible route to destination node k though port i.

a) Read T (k) from port Pr and update D (i, k). Receive message from neighbor and Place in local 'T (k)' array. Update Array 'D (Pr, k)' with 't' if 'T (k)' = 't-1'; Where, Pr is the local receive port ID.

Notes: Each message received on a port may or may not update element(s) in Array 'D'. Suppose, Node 'A' received Array 'T (k)' from Node 'B' on Port '4'. Port 4 will cause an update elements 'D (4, k)' if 'T (k)' = t-1.

b) Build message from 'D (i, k)' array and transmit to neighbor.

Notes: To build the message for transmit check down the columns of the 'D' array. The 'D' array specifies the "hops" to a destination node. For example, if a column 'k' has the value of 't-1' then it can be said that destination node k is 't-1' hops away, which also means that all ports of this node see node k as 't-1' hops away, if the turn is allowed. If the turn is not allowed, write '255'. All this information must be put in the Tx message. The Tx message will be an array T (1X16).

CHAPTER 5

THE VHDL MODEL FOR TURN PROHIBITION

5.1 Description of the Model

The high level simulation module for the architecture discussed in the previous chapter is simulated and tested in VHDL. The basic architectural module underlying the behavioral process consists of 3 sequential units namely token unit, prohibition check and routing table construction. Each of the unit's output is fed as an input to the following unit, which determines its function based the input fed from the previous unit. The system level module consists of various tasks performed at different points of time. They are mentioned below along with the inputs and the outputs. The chip is checked for its sequential execution, which assumes that the Enable signal to be 'high', and also the token is assigned a certain value which can be the identity of the node. Also all the possible interrupts are made 'low', so as to confirm whether the prohibition unit takes the control of the chip after the token unit has completed its computation of the assigning the nodes with an identity. Here the architecture is verified keeping in mind that the nodes already have an identity assigned to them, which means to say that the prohibition unit is mainly targeted for the efficient performance to verify the architecture.

The block diagram for the model designed in VHDL is as shown in Figure. 5.1. This has three blocks basically which are the three main blocks discussed in the architecture namely the Token Unit, Prohibition Unit and the Routing Table Unit. As can be seen each block within the three main blocks has been divided into many sequential

sub-blocks. Each sub-block does a unique function, which is described in the architecture.

The Token Unit block has7 sub-blocks. The whole process starts when the Node id and the port status are known. The first sub-block namely the state machine calculates the minimum value of the $(d_a^2-2)/\sum (d_i-1)$ based on the information from all the neighboring nodes. After this is done then the transmit process is invoked where in the result to be sent out to the neighboring nodes is ready. Then the block waits and checks if there are any interrupts from the nodes nearing it. This is taken care of by the interrupt arbitration block. Based on the interrupt then either a transmit or a receive process is invoked. After the interrupts are checked, data bus arbitration takes place so as to make to the bus available for the data. Finally the chip select is generated which will basically set the enable signal high and also restores the token, which will enable the Turn Prohibition Unit.

In the Turn Prohibition Unit, the enable and the token is checked by the prohibit and the token processes. Then interrupt control checks for any of the external interrupts and the I/O control checks for the interrupts due to data or the address coming from the other nodes in the network. The next sub-block, which is the bus arbitrator, does the similar function as the one in the Token Unit. In fact the same block is invoked at different points of time depending on the necessity. The packet broadcast block takes care of formulating the packet and also has the information as to which turn is prohibited and which turn is permitted so that the neighboring nodes will have an idea on the turns in the network. The time out process is used as a counter within which the process has to be completed.

The Routing Table block gets the input from the Turn Prohibition block, which has mainly the information regarding the ports, which are permitted and the ports, which are not permitted. This information will be in the form of a 16 bit data. Then the table is formulated; this formulation is done initially for the first time. Only updating and reading from the records are done all the time. The build table block is done after updating each record, so that only final table will be stored and that will be used as a reference.

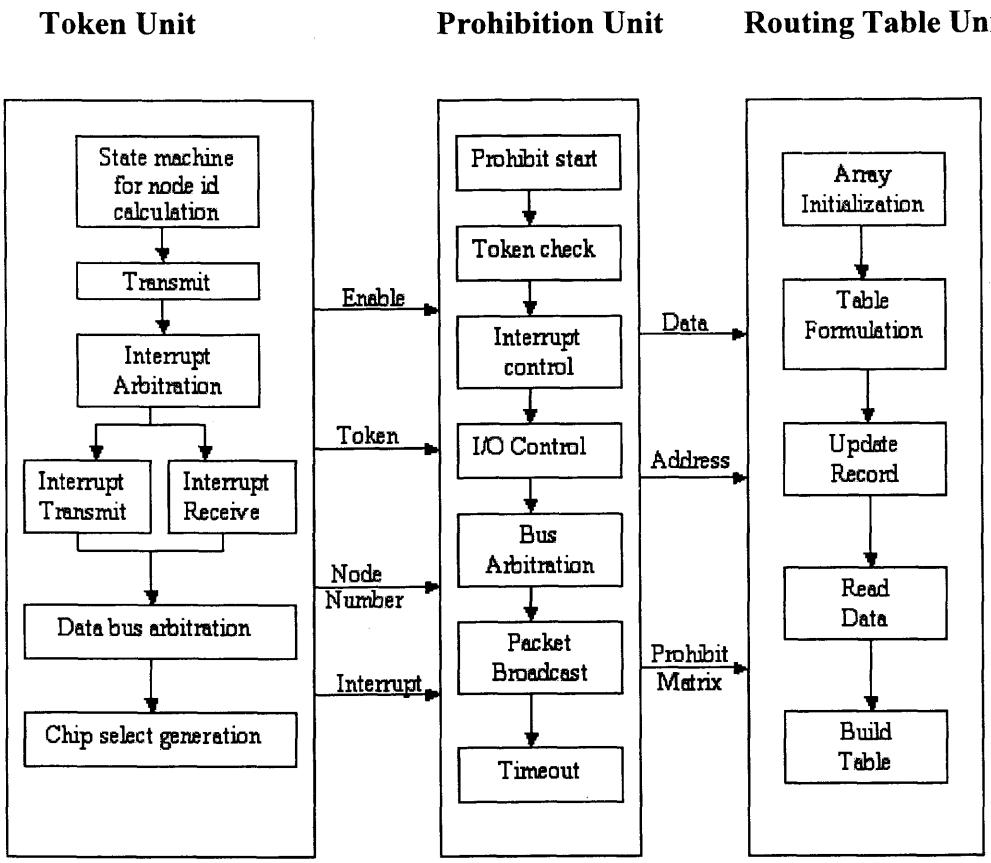

Figure 5.1 Block Diagram for the model designed in VHDL.

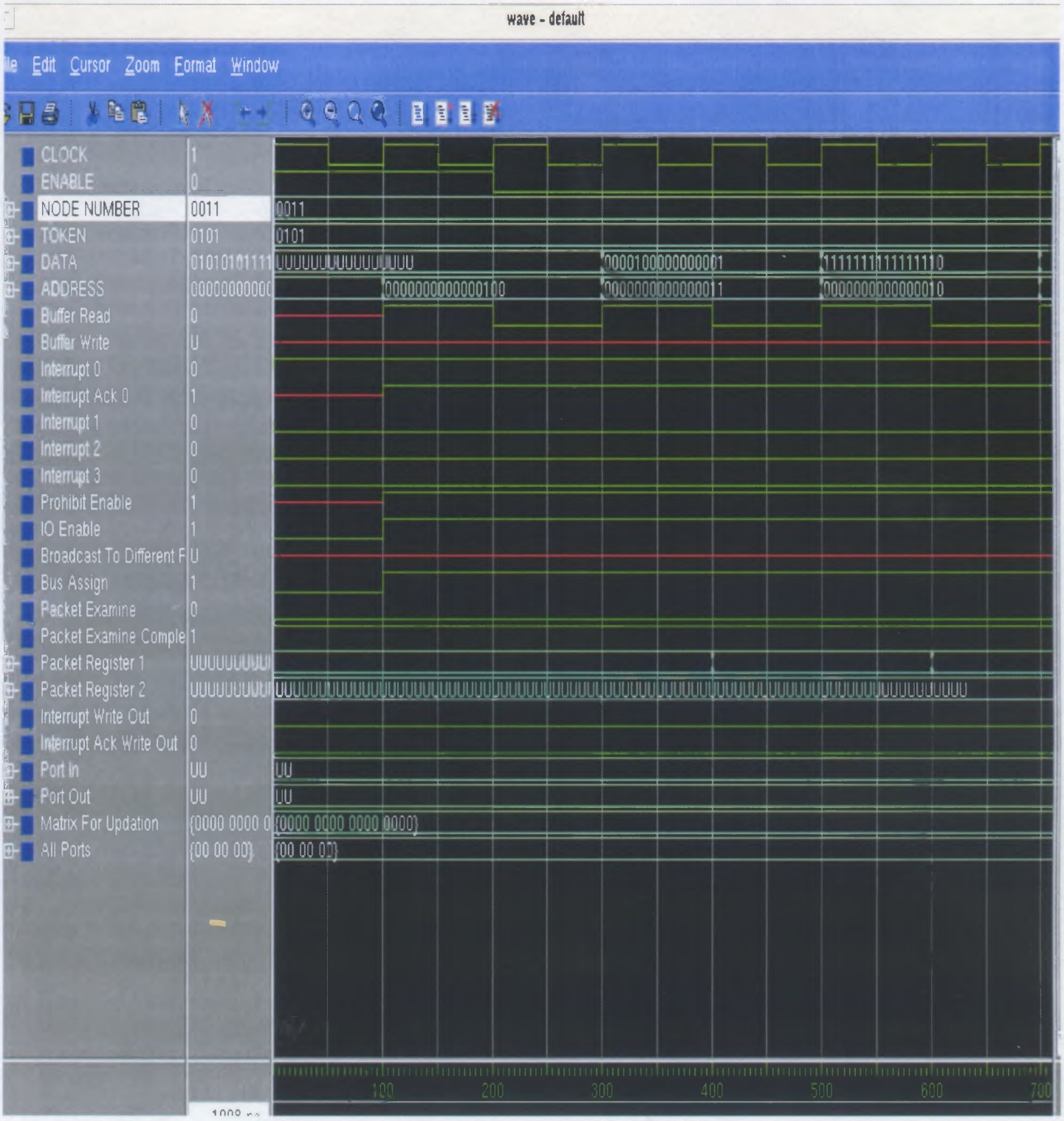

Figure 5.2 High-level timing simulations for the mentioned sequential tasks.

5.2 Simulation of the Model

As can be seen from Figure. 5.2, enable is '1' at 0ns. Therefore after 1 clock cycle that is 100ns prohibit enable is '1' and the whole block is turned on by this signal. Here the node number is 3, which means the node which is acted upon is 3 and the token is 5, which means it's the fifth node in the network. With this the buf_read signal also becomes high but since there is no interrupt at that time IO process doesn't start. After 100ns it senses the 'interrupt 0' to be 1, and with 'io_enable' being 1 at 200ns the IO process is started. 'Interrupt Ack 0' is made 1 at 100ns. Also 'bus assign' is high at 100ns since the 'Interrupt 0' is high at 0ns. Since only 'Interrupt 0' is high, all the other interrupts namely Interrupt 1, Interrupt 2, Interrupt 3 are all 0. This is to make sure that only interrupt is being handled at this time. After 1 more clock cycle that is at 300ns 'buf_read' is high and at 400ns data is read into 'packet_reg1', which can be clearly noticed from the change at 400ns in the 'packet register1' signal. Signals like 'buffer write', 'broadcast to diff ports' are undefined because only the read operation is taking place at this time. 'Packet register 2' is also undefined because the data that is being read to the 'packet register 1' and also the prohibition process is not yet completed. Signals like 'Port in', 'Port out', are undefined as far as Figure. 5.2 is concerned because the process is not yet completed whereas signals like 'Matrix for updation', 'all ports' have still 0's because the table is not formed yet.

From Figure. 5.3 notice that, there is a change in 'data', 'addr', 'buf_read' and 'packet_reg1' till 2100ns. This shows that the packet in the buffer is read 16 bits at a time into a register for further examination. This packet can be token release packet or a packet having the information about the other node in the network. Also notice that the

signals 'Port in', 'Port out', 'Matrix for updation' are all initially set to the '0', but the 'port in' signal gets a value '00' at 1100 ns indicating that the data which is going to be processed is coming from "Port 0". Therefore it is noticed that at 1000 ns, the 'Interrupt 0' has become low and after 1 clock cycle that particular interrupt is acknowledged which can be seen at 1100 ns. Now also notice that the signal 'bus assign' has become low at 1100 ns indicating that the data to be processed is fed into the 'packet register 1', so at the same time the 'packet examine' goes 'high' indicating that the prohibition unit has taken control of the incoming data. At 1200 ns it can be seen that the 'packet examine complete' signal is high indicating that the process is over and so at the next clock cycle which is at 1300 ns it can be seen that all the ports other than port 00 are intimated regarding the interrupts, which means that the module is now in the process of Table formulation and will not process any interrupts from the other ports. Signals 'interrupt 1', 'interrupt 2', 'interrupt 3' all remain low because only one interrupt that is 'interrupt 0' is taken into consideration for ease of understanding.

In Figure. 5.4, after the Data is read, the 'Bus assign' signal becomes low indicating that all the Data is read from the bus. At the same instance 'Packet examine' signal goes high, which marks the starting of the actual process of turn prohibition. It can also be seen that the 'Port in' signal has got a value indicating that the packet is got from the Port '00'. 2 clock cycles after the 'Packet examine ' is high, a change in the signal 'Packet register 2' is seen, indicating that the data present in the 'packet register 1' is being acted upon and is finally put in 'Packet register 2'. At the same time the 'interrupt write out' is high indicating that the data is ready to be sent out, also it can be noticed that the 'Port out' signal has changed its value to '01' at 1400 ns, indicating that the data is to

be sent out through the Port 1. Now the data in the 'packet register 2' is sent out through Port 1. In this figure 'prohibit enable' and ' IO enable' remain high as the process is still in progress.

The continuation of Figure. 5.4 is shown in Figure. 5.5, wherein it can be seen that the packet register 2 has been updated with the data present in the 'packet register 1', indicating that the data is ready to be fed out to the main bus which will be sent through the port 1. Also it can be seen that the data signal has again gone back to the undefined state indicating that it is ready for the next IO process. After the last data packet is fed into the 'packet register 2', the 'bus assign' goes low indicating that the bus is free for data transfer which can be seen at 2200 ns. After the data has been moved to the 'packet register 2', the 'buffer write' signal which was undefined so far becomes high indicating that the data is ready to be fed out. In this process the interrupt write out and its acknowledge shows the transition. Now from the whole process it can be noticed that the 'port in' is port 0 and the 'port out' is port 1, and since the algorithm for turn prohibition has recognized that this particular turn doesn't pose any problems as far as deadlock is concerned, the matrix is now updated which will mainly have the permitted turns at 1's in the matrix and prohibited ones as 0's. This can clearly be seen from the signal 'Matrix for updation' wherein there is a 1 in the first set of 4 bit s and also a 1 in the next set of 4 bits, indicating that the turn from port 00 to port 01 is permitted.

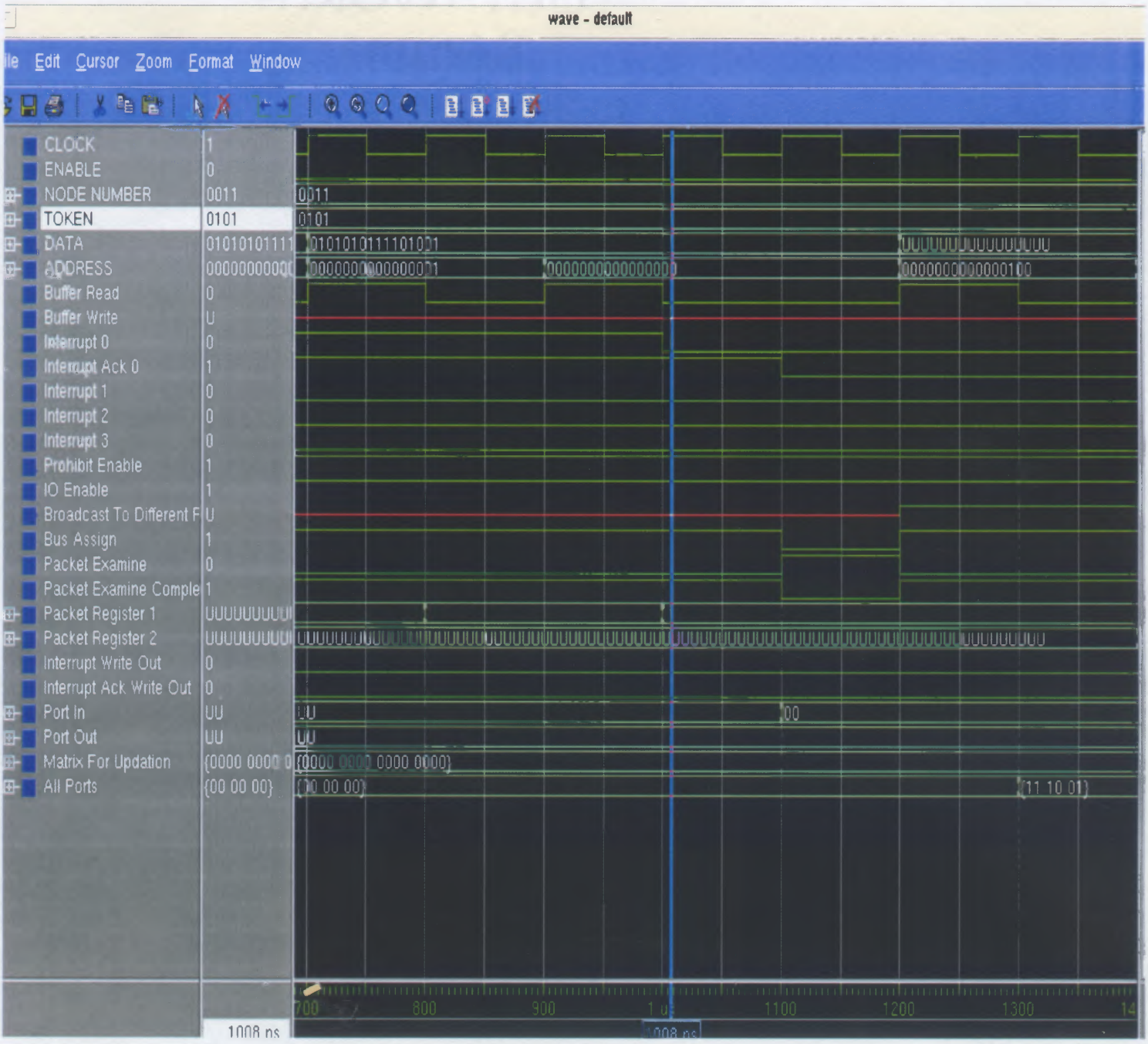

Figure 5.3 Timing simulation continued after Figure. 5.2.

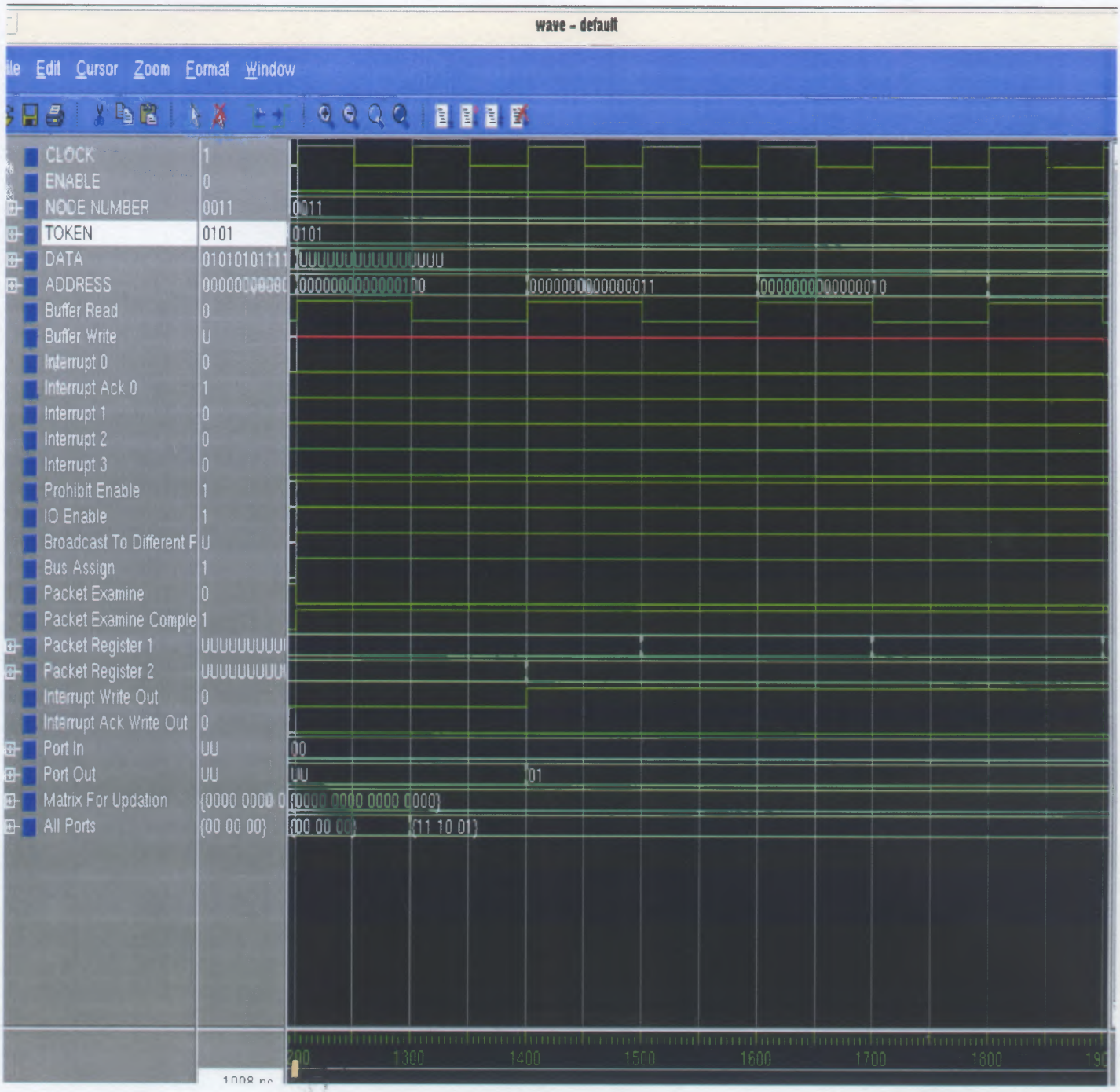

Figure 5.4 Timing simulation continued after Figure. 5.3.

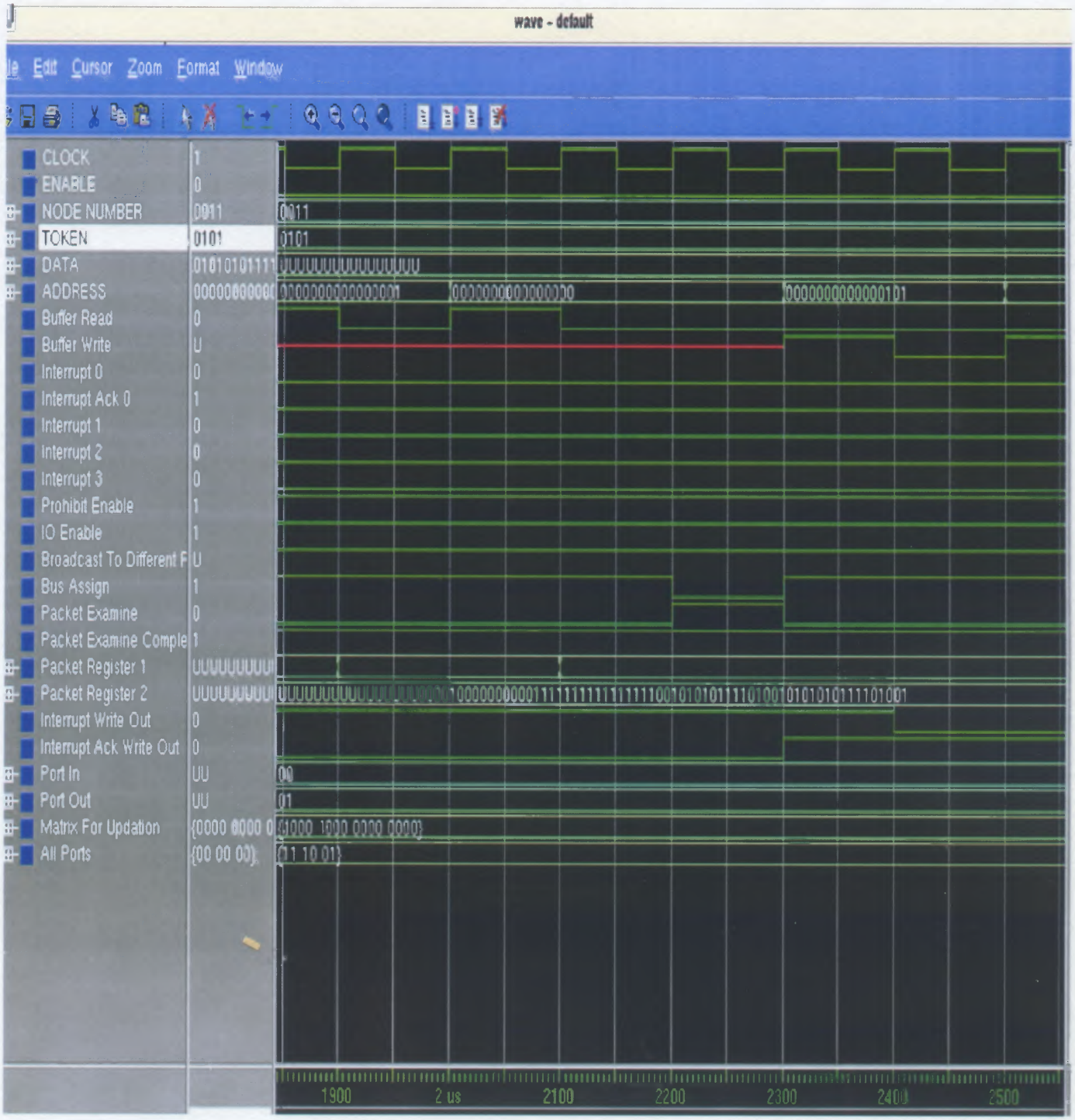

Figure 5.5 Timing simulation continued after Figure. 5.4.

CHAPTER 6

SYNTHESIS OF THE MODEL

6.1 Synthesis Problems

The VHDL model is synthesized with Cadence Ambit Bulidgates. However, it was not synthesizable first up as the build gates didn't support multiple wait statements in a process and also the event scheduling on signals. Therefore the model had to be modified keeping these in mind.

Only clock'event was synthesizable and this event could be used once in a process. Therefore checking the condition for every rising or falling edge of a clock was a difficult task. The model was changed accordingly so that at the beginning of every process the rising edge of the clock is checked and whenever there is an event desired either the rising edge or the falling edge, a flag is made high and this flag was being checked periodically when there was a necessity for a conditon to be checked at the rising or falling edge of the clock. Also mathematical functions like square root was not supported by the buil gates, so whenever square root was necessary, the squares of the signal had to be taken and calculated for results. There was also problem due to insufficient memory which resulted in disintegrating the blocks into smaller sizes. Power optimization is not supported by the Build gates. Only the blocks could be optimized for Area and Timing. In Build Gates we first build the generic and then optimize the generic for Area and Time. Then the generic is built, the verilog netlist is created, which will have the cell structures of ATL and XATL formats. These formats are further disintegrated to the

Nand, Nor And, Or, Mux, Latch, Flip Flop and other basic components during Optimization. The problem was that during disintegration, some of the cells in the generic like ATL_TRI ,ATL_DC couldn't be disintegrated further down into one of the basic components. This created a problem in Silicon Ensemble as these ATL cells were not recognized. Therefore back tracking had to be done to see which process had created the ATL cells and the process had to be modified keeping the functionality in picture.

In the synthesis, the design was mapped to TSMC 0.35μ technology standard cells generated by CMC. A total of 1 million gates resulted after optimizing it with strict area and time constraints. The design was optimized with area as priority. The optimization was done with Physically Knowledgeable Synthesis (PKS) option where the cells are physically placed for calculating the parasitic capacitances.

6.2 Synthesis of the model with Cadence Ambit BuildGates

In this section, synthesis of the VHDL model with Cadence Ambit Buildgates is described. Following are the steps that were followed for synthesis.

1. Ambit BuildGates is started by entering a command called "cadence" at the console and choosing the 6^{th} option.
2. Buildgates is started with *ac_shell –gui –pks* at the ac_shell prompt in the command window of the application .
3. The 'File -> Open' menu is brought up and the timing library option is selected to read the 'timing.ctlf' file provided by standard cell library vendor. This file has capacitance, timing and functionality of the cells and wire load models for calculating the delay due to routing parasitics.

4. The 'cmosp35_4m.lef' file is read by typing 'read_lef ~/proj/cmosp35_4m.lef' in the command window of the application. This file contains the technology information used to develop the standard cells and also their abstract views. This file is used to run the PKS.
5. The VHDL file is read by selecting the VHDL option in the File -> open window.
6. The VHDL model is then mapped to generic gates with 'Commands -> Build Generic ...' and selecting the first 3 options in the build generic window.
7. The constraints are set by typing the following commands in the command window. The clock is necessary for timing optimization. The second command tells the tool that the input arrival time is 0; third one tells the tools that data required time is 10 ns. These two commands are the constraints to optimizer. The fourth command tells the tools to use the wire load model enclosed.

```
set_clock clock -period 2.0 -waveform {0.0 1.0}
set_input_delay 0.0 -clock clock [find -input *]
set_data_required_time 10 -clock clock [find -output *]
set_wire_load_mode enclosed
```

8. Optimization window is brought up by 'Command -> Optimize' menu. The 'Effort level' is set to high, 'Flatten mode' is set to off, 'Priority' is set to Area/Time and in 'Options' minimize area/Timing budget is selected accordingly. The optimization is shown in Figure. 6.1, Figure. 6.2.
9. Now a gate level design is produced in which optimization is done with the wire load models in the library. Wire lengths calculated are approximate and the timing analysis done by the tools to added buffers is not accurate. For accurate results PKS is done by selecting the PKS option in the optimize window. In this, the cells

are actually placed and the wire lengths are calculated. The placement is shown in Figure. 6.3, Figure. 6.4.

10. The synthesis is complete and the design is saved as gate level verilog netlist and DEF format which has the placement information generated in PKS using 'save' window. If timing driven placement and routing is to be done a GCF file should also be produced which has the timing constraints and the path of 'ctlf' file.

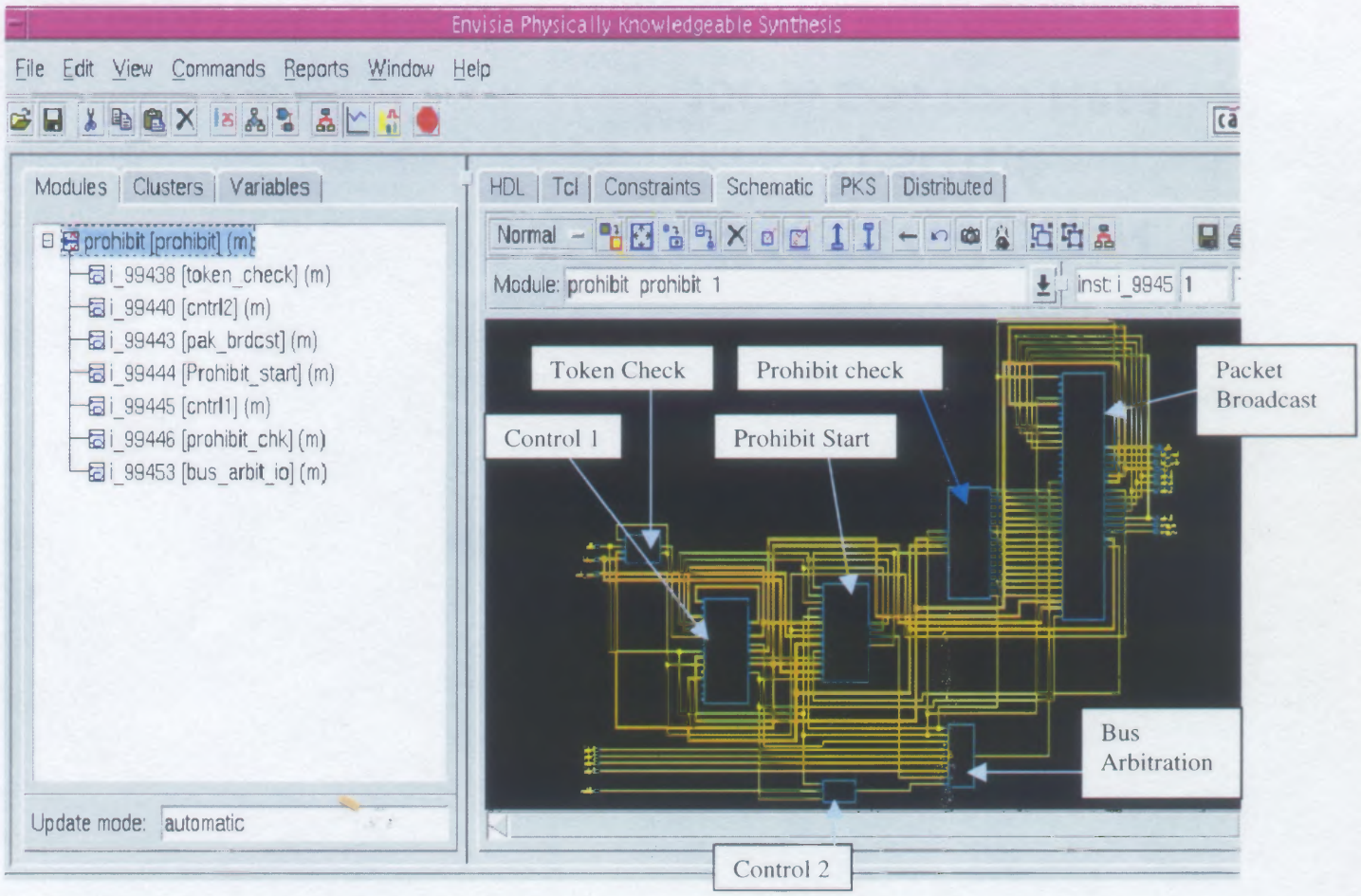

Figure 6.1 Optimized synthesis for the Turn Prohibition Unit and Token Unit.

In the Figure 6.1 it can be seen that optimized synthesis for the Turn Prohibition Unit and the Token Unit has 7 blocks namely the token check, control 2, packet broad cast, prohibit start, control 1, prohibit check and bus arbitration unit. All these blocks are labeled in the figure. The functionality remains the same as described in the architecture.

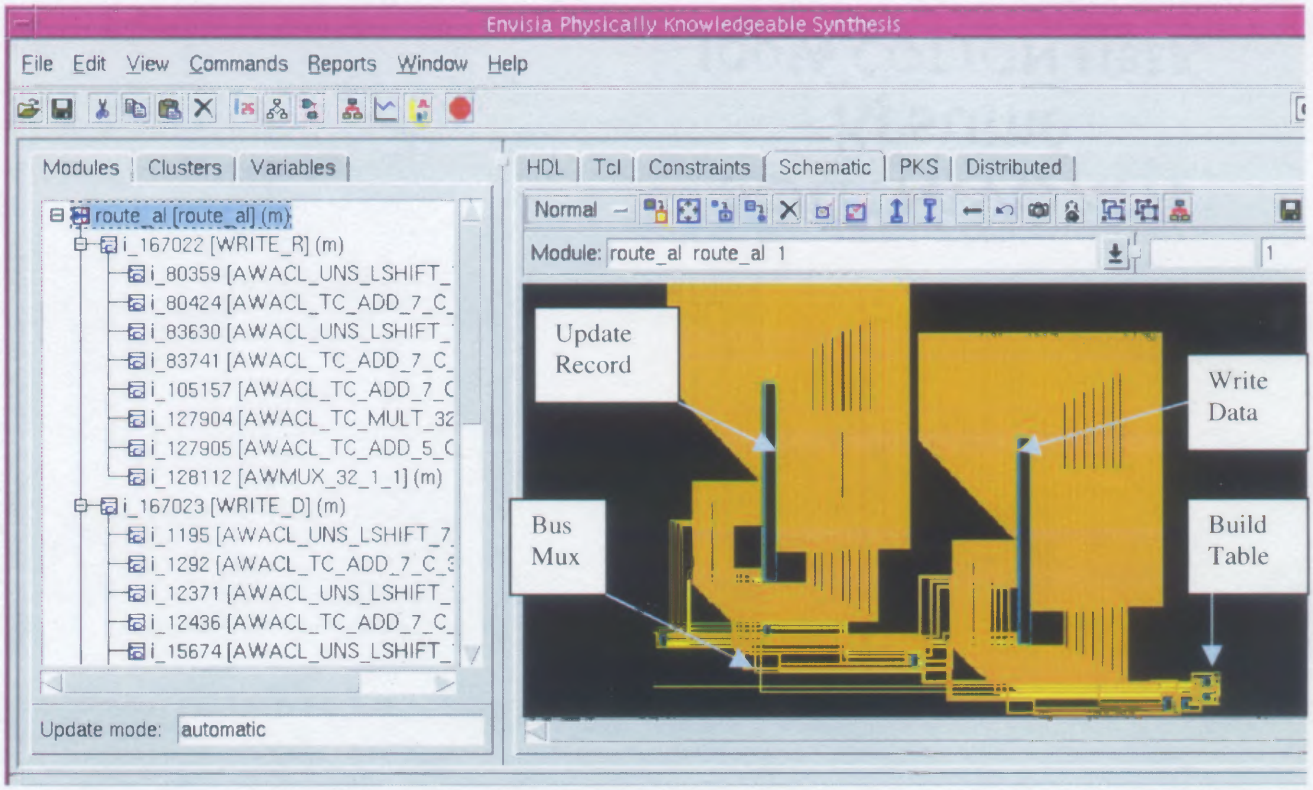

Figure 6.2 Optimized Synthesis for the Routing Table Unit.

In the Figure 6.2, the optimized synthesis for the Routing Table unit is presented. Though there are 7 blocks in this module, only 4 of them are highlighted, as the other three blocks are very small and cannot be seen unless it is zoomed further. The blocks which are visible are the update record blocks, multiplexer block, write data block and the table formulation block. These blocks perform the same function as described in the architecture. The darker lines show the connectivity between the blocks.

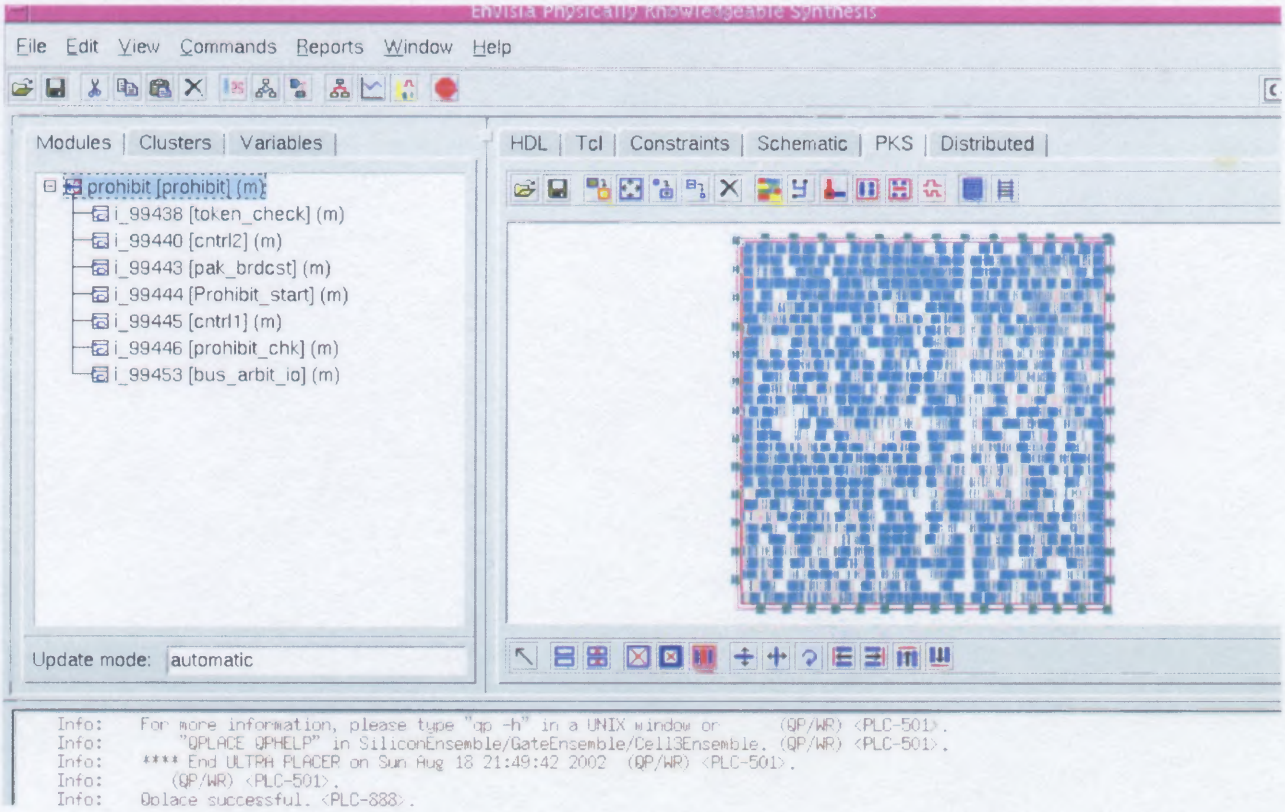

Figure 6.3 PKS showing the placement of all the cells.

The PKS option also generates the random placement of the IP blocks depending on the approximate size of the chip. A total of 54 IO pins are created randomly around the die area of the chip as shown in Figure. 6.4. In fact all these pins are not used by the process, only 40 pins are used by the whole process of which 16 pins are for the data and 16 pins are for the address and the rest 8 pins are for the clock, chip select, interrupt, interrupt acknowledge, node ID, port status, read/write and the Tx/Rx. In addition to this a reset pin can also be taken into consideration. Since the use of this pin is not demonstrated, this pin is not taken into account. Since this PKS synthesis is done automatically, the pins placement and the pin count cannot be controlled. However the pin count obtained by the synthesis is always greater than the desired pin count

6.3 Post Synthesis Simulation

In the synthesis process, gate level models of the design were produced and were simulated with MTI Model Sim for functionality as well as timing as described in chapter 5. For simulation purposes, a top-level verilog module is obtained from the Cadence Ambit Buildgates in which all the modules are instantiated. Since all the modules are sequential, all the blocks were simulated individually in a pipelined way, which means that the output of one block is input to the next one with the produced delay. The final results are shown in the following figures.

Figures 6.4 through Figure 6.6 show the solutions for the model developed. The results were almost the same as those obtained before synthesis. All the signals behave in the same way as obtained before the synthesis, but only with delays due to the gates created by the synthesis. It is observed that the worst delay is around 1200ns which is observed at the 'Packet register 2' which happens to be the final process. The difference can be seen from Figure 5.4 and Figure 6.6.

The delays observed here are just due to the gate delays. So in the worst case this delay will not be more than 2μ second. However, in the actual layout there will be a lot of parasitic capacitances due to routing metal layers. Due to size of the design, the actual delay may drastically go up because of these parasitic capacitances.

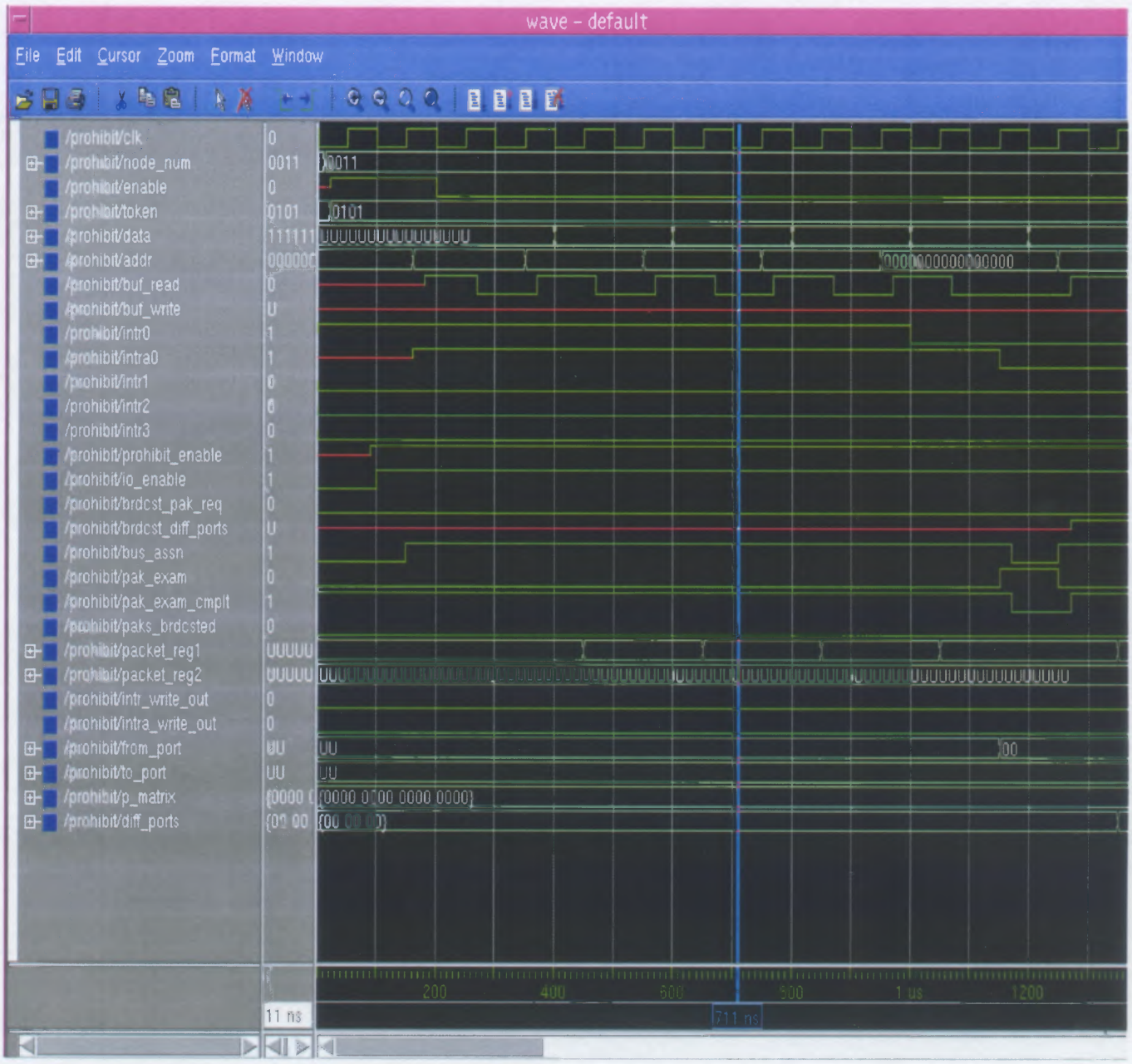

Figure 6.4 Post Synthesis Simulation for the model designed.

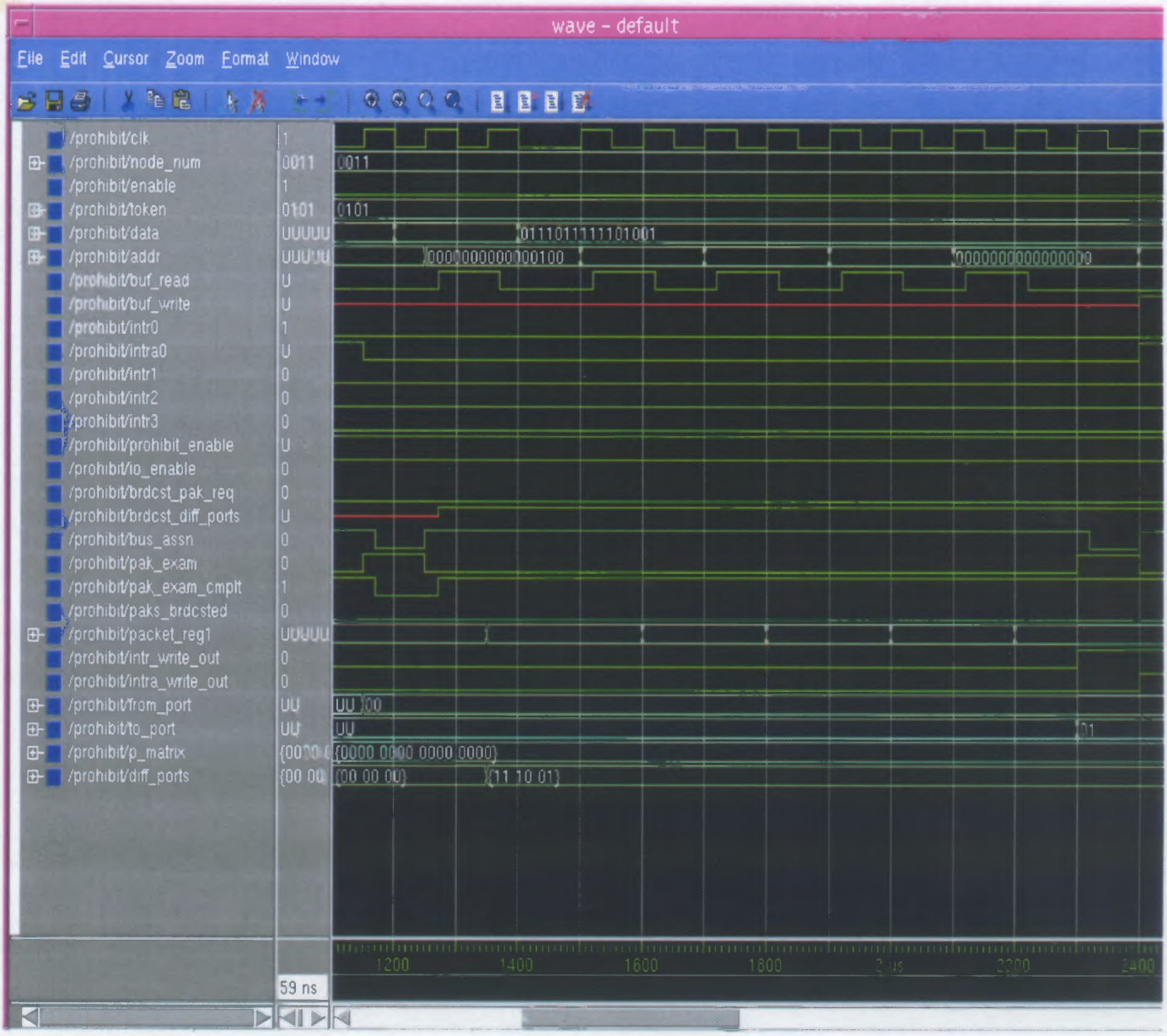

Figure 6.5 Post Synthesis Simulation continued from the Figure 6.4.

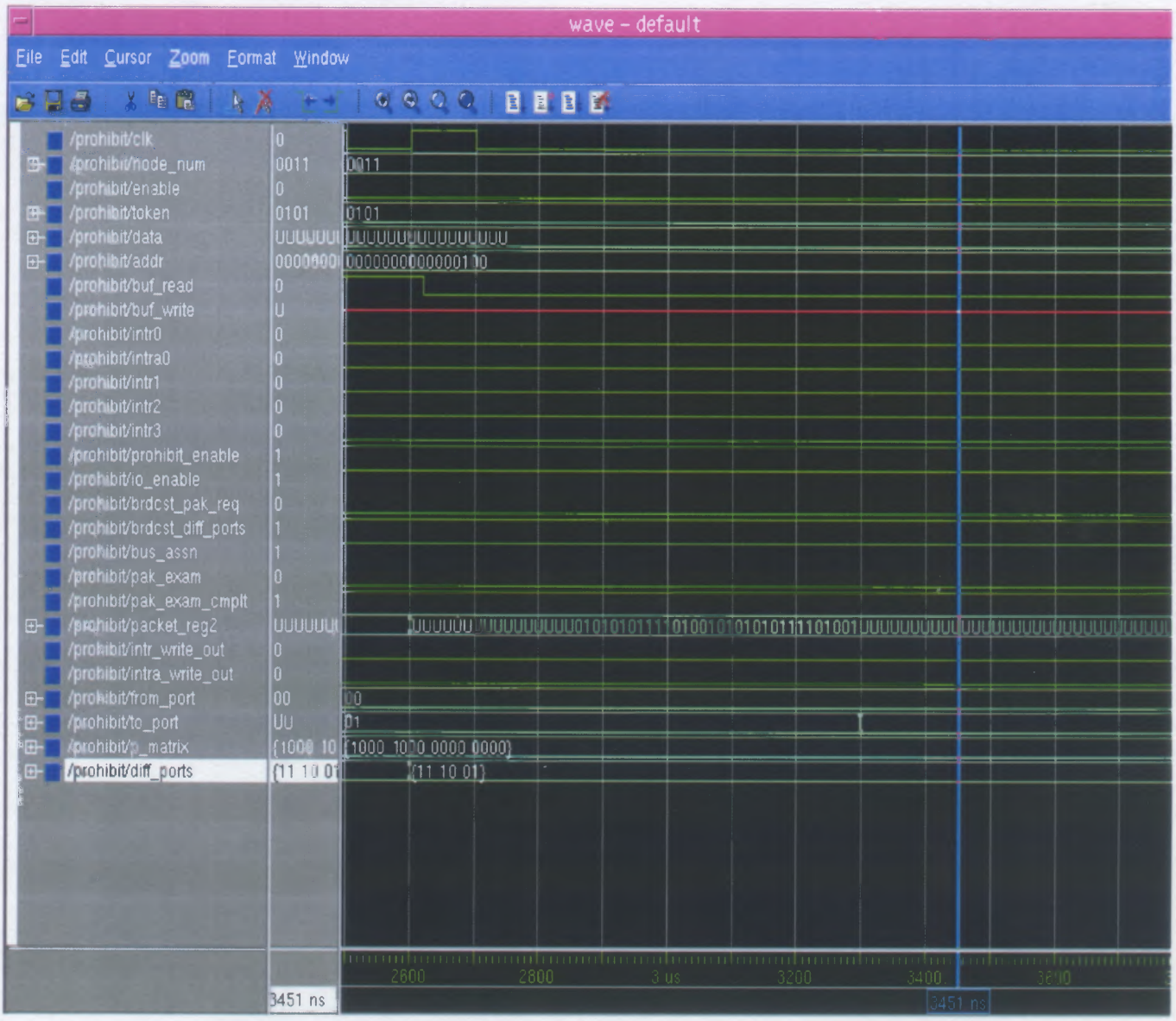

Figure 6.6 Post Synthesis Simulation continued from the Figure 6.5.

CHAPTER 7

LAYOUTS

7.1 Place & Route with Silicon Ensemble

The placement and routing using Cadence Silicon Ensemble is discussed in this section This performs the timing and power driven placement. The layouts of the individual blocks are put together using the top-level verilog module with the help of the same tool. Only regular placement and routing is done to save time and computer memory. The total die size without pads came upto 6 mm x 6 mm. This could not be imported into the Cadence IC tools for DRC and Extraction due to insufficient computer memory. In this tool the synthesized design is imported in block level verilog format .The standard cell library is imported into tool in LEF format which is the "cmosp35_4m.lef". The output to this tool is the layout which is stored in the form of LEF block and DEF form which is then imported to the Cadence Virtuso Layouts for Design Rule check and the Extraction. Following steps are followed to get the layouts using the Silicon Ensemble.

1. Silicon Ensemble is started by entering a command called "cadence" at the console and choosing the 4^{th} option.
2. The "cmosp35_4m.lef" file is imported by using the 'File -> Import -> LEF' menu. After this step a database is created for storing the created design and viewing it when needed.
3. The verilog netlist which has a '.v ' extension ,which was generated in the Ambit Build Gates is imported by using 'File -> Import -> Verilog' menu.

The top module should be identified correctly. Also the power (VDD!), and ground (VSS!) nets should be entered as show .Make sure the VDD and VSS are in upper case. The reference libraries and the compiled verilog output libraries are "cds_vbin" by default . Also a verilog module called "cells.v" is also imported along with the main verilog module, which has the information about the basic gates and their specifications, which are helpful for generation fo the layouts.

4. After the compiling is done successfully the floorplan has to be initialized by using 'Floor plan → Initialize' menu. The IO to core distances in the dialog are initialized to 40 microns. Space utilizations is kept for about 70-75 %. This creates the rows for standard cell placement and 40 micron empty space around them. This space is used for VDD and VSS rings and IO connectivity. Figure. 7.1 shows the layout without the power and ground stripes.
5. IOs are placed with 'Place → IO' menu. In the displayed dialog, IO constraint file option is selected and the name of the file is entered. This is a DEF file, which has IO pads' placement information. This is developed manually as per our requirements.
6. Power plan dialog is brought up with 'Route → Plan Power' menu. VDD and VSS rings and stripes are added. Rings are the power paths that surround the core area and stripes are the power paths that pass over the core area.
7. Standard cells are placed with 'Place → Cells' menu with 'Generate Congestion Map' option. The layout after placement of IO and cells and the power rails is shown in Figure. 7.2. As can be seen in the Figure. 7.2 two

stripes are added which run vertically in between the layout for the power and ground lines, so as to minimize the drop.

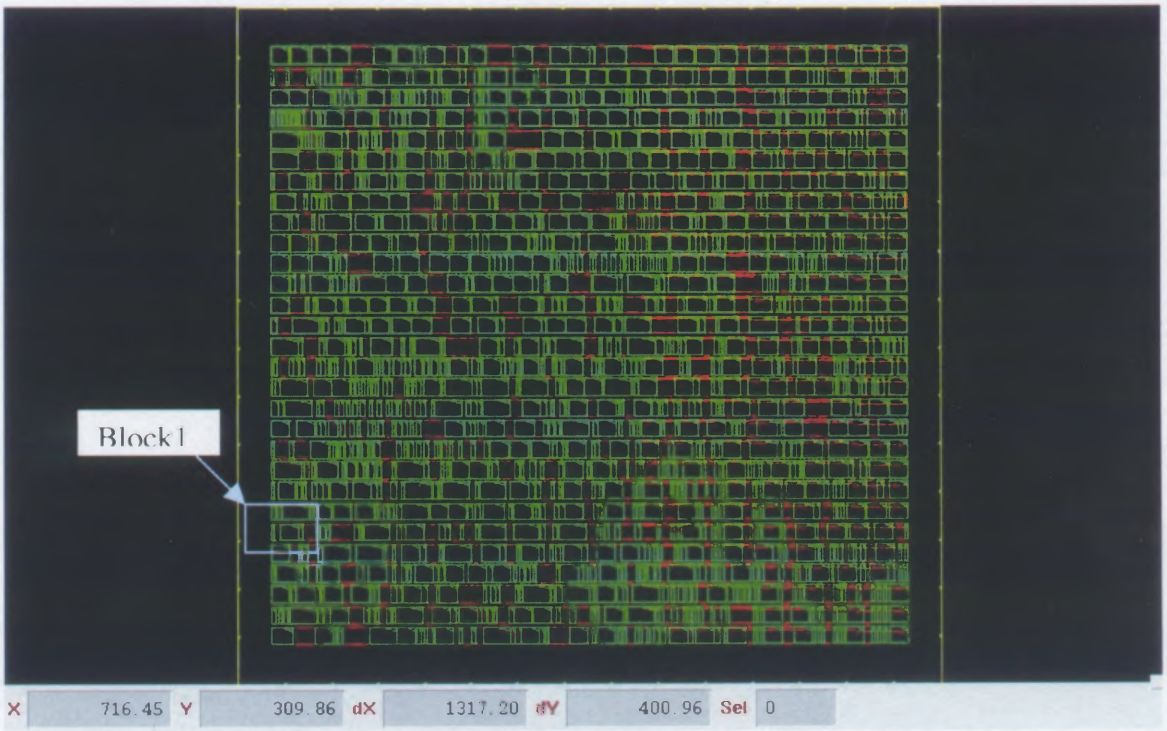

Figure 7.1 Layout of a block without the power stripes.

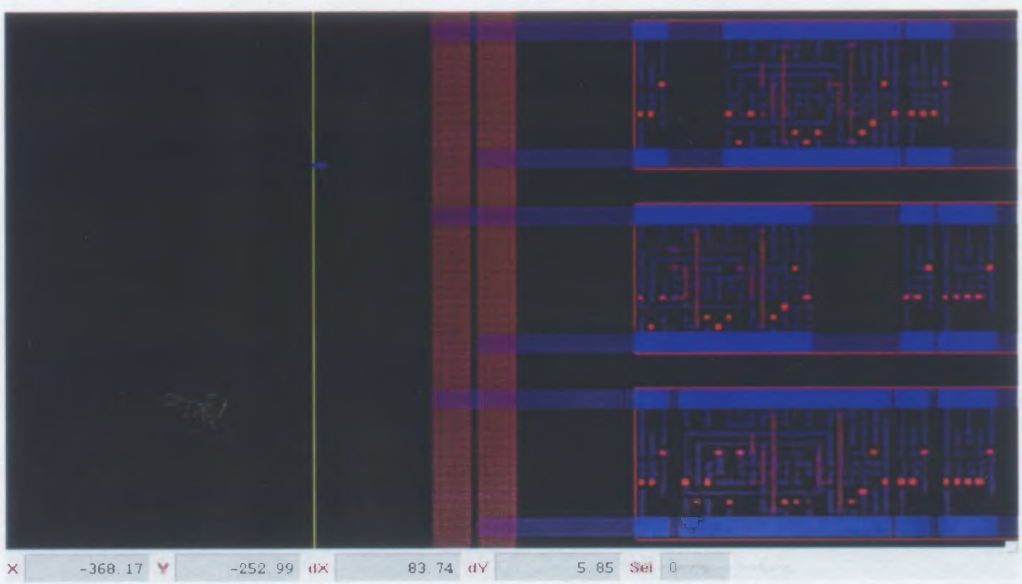

Figure 7.2 Expansion of 'Block 1' shown in Figure 7.1 showing the power connections.

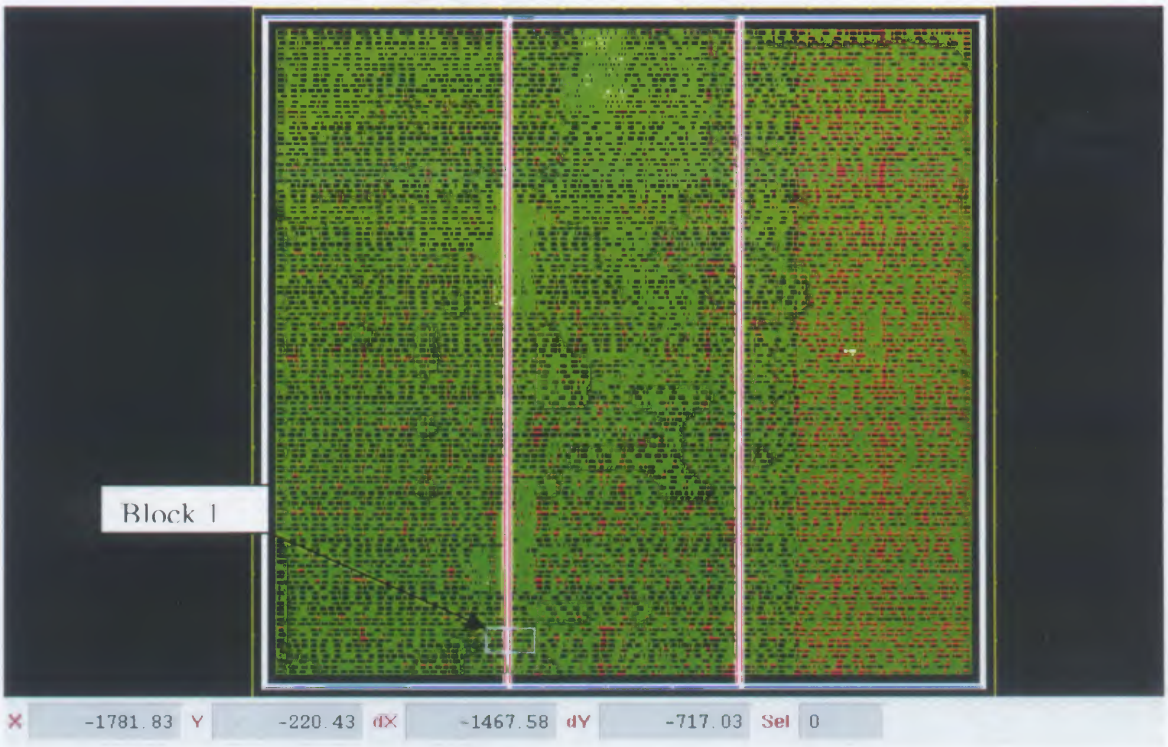

Figure 7.3 Layout of the chip with power and ground rails with stripes in between.

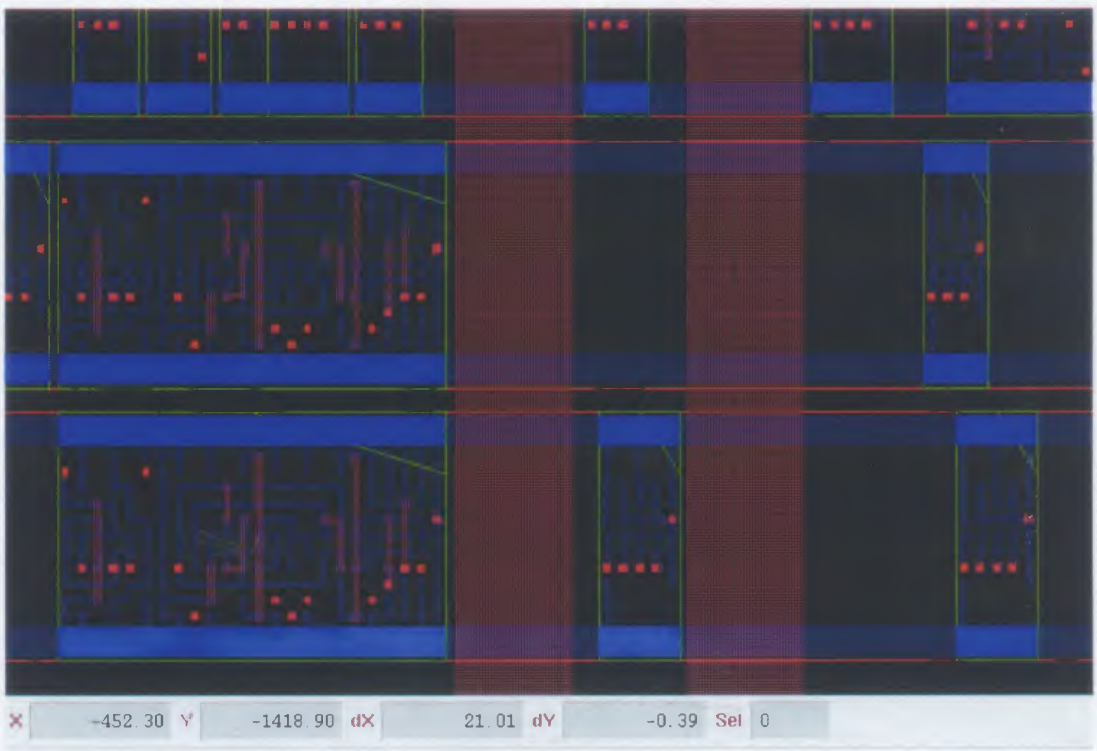

Figure 7.4 Expansion of Block 1 shown in Figure 7.3. VDD and VSS stripes can be seen in between the layouts for drop control.

8. The connection to the rails are done by using 'Route → Connect Ring' menu. Figure. 7.3 shows the power connections between the cells and the VDD and VSS rails.These rails are present at the outer ring of the Die.
9. Routing is done with 'Route → Wroute' menu as shown in Figure. 7.2. Here global and the final routing is done together.
10. The design is saved as DEF ,LEF BLOCK formats with 'File → Export' menu. Also the database is saved with 'File → Save' menu. This is useful for viewing or modifying the layout at any time.

This concludes the layout process through Silicon Ensemble tool. The DEF and the LEF block format is then exported to the Cadence IC tools for DRC and Extraction.

7.2 DRC and PE with Cadence IC

Cadence IC tools are used for DRC and Extraction of parasitics. Also the GDSII for tape out can be produced only from this tool set. A step-by-step procedure for working with the imported design from Silicon Ensemble is described here. The flow diagram is as below.

1. ICFB is invoked with '*icfb &*' command.
2. In the IC environment, a library is created for the design with 'File → New → Library' menu in the CIW window.
3. The layout developed in the SE is imported into IC in DEF format with 'File→ Import → DEF' menu. In the dialog, the library name is the name of the library created and view name is 'auto Layout'

4. Imported design will use the 'abstract' view for all cells. All 'abstract' views are to be changed to 'layout' views. First, 'auto Layout' view is opened, then with 'Tools →Layout' menu layout tool is invoked.

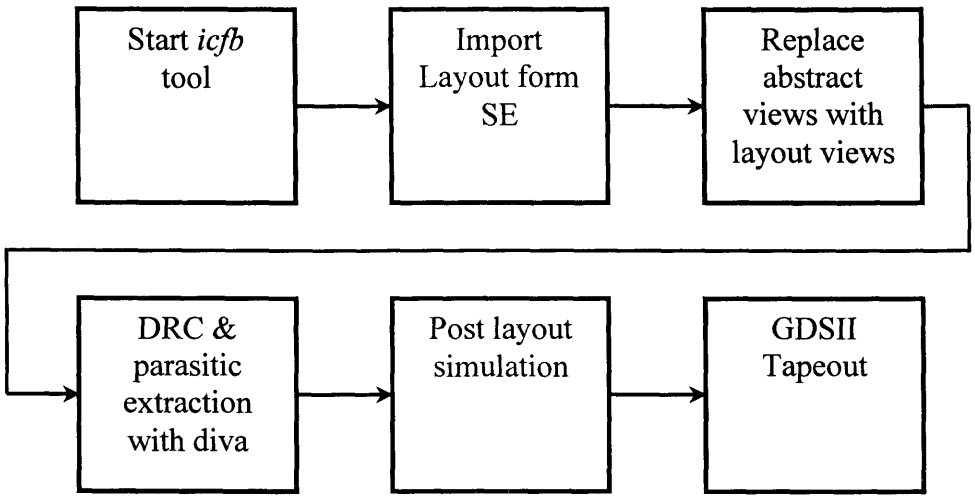

Figure 7.5 Cadence IC Design Flow.

5. Using 'Edit →. Search' command search menu is brought up. All instances with view name = 'abstract' are searched and replaced with view name = 'layout'. The modified design is saved as 'layout' view.
6. With 'Verify → DRC...' menu DRC dialog is brought up and in the library box, name is changed to the name of the technology library and "OK" button is clicked. This will check for DRC and report the errors with flashing rectangles. These are to be corrected manually. Usually if the LEF file imported into SE is perfect in technology point of view then there will be no DRC errors.

CHAPTER 8

CONCLUSIONS

This thesis gives the architectural details and the on–chip implementation procedure for the TPBR algorithm, which will avoid deadlocks in wormhole networks like Network Of Workstations. It can be seen that the fractions of the turns, which will create the cycles in the channel dependency graph, are eliminated so that the whole of the network would become deadlock free, which will lead to efficient communication within the network. It is also considered that an efficient way of creating routing tables based on the set on prohibited turns thus by minimizing the average delivery time and achieving low communication latency, low memory requirements and achieve high bandwidth in data communications. The implementation in VHDL was a straightforward approach in which as the block were sequential. The VHDL model could be synthesized and implemented on silicon with some exceptions. The total number of gates produced was 1 million. Most of the gates were due to the use of 'for' loops and also due to the lack of complex gates in the standard library, which just has basic gates. This resulted in large area and hence the gates. The gate level verilog simulation of the design could not be carried out and so was the DRC and extraction because of large gate count and insufficient computing power. The gate count could be considerably decreased by doing one or more of the following.

1. Use of Karnaugh–maps for breaking down the modules and getting it to a basic Boolean expressions and then performing the manual synthesis.

2. Use of shift operations for addition and multiplication of Std_logic_vectors, as there is no direct syntesis for operations on Std_logic_vector
3. Reducing the type conversion, like integer to std_logic_vector and back, can reduce the gate counts drastically.
4. Use of tools which support numric_std and unsigned libraries for optimization during the synthesis.
5. Using standard cell libraries which have complex gates, for optimization in area and timing. The library used here was very well optimized for only for timing and area but there are no libraries for power optimization.
6. Use of merging operators during the synthesis. For this the whole VHDL model should be synthesized with out breaking it into blocks, which requires a large memory and time.
7. The device level simulations can be carried out by using Star-Sim and Start-time which are high capacity simulators.

Use of multiple *'event* in a process was not possible on clock signal and also use of *'event* on any other signal was not possible in a process, which resulted in lots of 'if ' and 'for' statements for condition checking. Use of flags for the *clock'event* at every instance during the process resulted in expanding the code. This also increases the pin count as the data and the address packets were fed parallely instead of using a shift registers for serial operations.

APPENDIX A

VHDL MODEL FOR TURN PROHIBITION

This Appendix gives the listing of the VHDL model for the TPBR Algorithm.

```
-- TOKEN UNIT
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
library work;
entity token_unit is
port (
      clock : in std_logic;
      node_id : in std_logic_vector(3 downto 0):="0000";
      vector_8: in std_logic_vector(7 downto 0);
      enable: out bit:='0';
      order: out std_logic_vector(7 downto 0):="00000000";
      intr0,intr1,intr2,intr3 : in std_logic;
      dbus : inout std_logic_vector(15 downto 0);
      pstat : in std_logic_vector(3 downto 0);
      addbus: out std_logic_vector(15 downto 0);
      rw : out std_logic; --0 read 1 write
      reset: in std_logic;
      cs0,cs1,cs2,cs3: out std_logic);

architecture arch_behavioral of token_unit is
type other_array is array( integer range 0 to 3) of std_logic_vector(10
downto 0);
type array_sort is array (integer range 0 to 1) of array_new;
type state is (start, ini_tx, waiting ,spec,transmit,receive,ending );
type array_new is array( integer range 0 to 15) of integer;
signal node_id: integer;
signal BITPM: bit_vector(3 downto 0):="0000";
signal route_table: other_array;
signal degree:std_logic_vector(1 downto 0):="01";
signal order: integer;
type port_array is array (integer range 0 to portsize-1) of
std_logic_vector(127 downto 0);
signal in_port,port_to :port_array;
signal act_tx : std_logic:='0';
signal act_all: std_logic:='0';
signal intr_out: std_logic:='0';-- no intr
signal intr_out_ck : std_logic:='1'; -- no response
signal port_num: std_logic_vector(1 downto 0);
signal d_rx: std_logic :='0';-- no data
signal rxd_port: integer:=0;
signal rxd_ck: std_logic :='1'; -- no ck;
signal dbus_busy : std_logic:='0';
signal txd_data: std_logic:='0'; -- 0 no data, 1 have data to transmit
signal txd_port: integer:=0;
signal txd_ck: std_logic:='1';
```

```
signal rec_begin: bit :='0';
signal present_state, next_state : state;
signal t_count :std_logic_vector(7 downto 0) ;
signal sreset: bit:='1';
signal txd_begin: bit :='0';
signal trans_ing : bit:='0';

begin

state_machine : process(present_state,d_rx,reset,trans_ing)
            variable all_dirty: bit:='1';
function get_order(signal pac: std_logic_vector(127 downto 0);
signal id:integer) return integer is
            variable sort:array_sort;
            variable flag,flag1,i,j,order: integer;
            begin
            ini: for i in 0 to 15
            loop
            sort(0)(i):=i;
            sort(1)(i):=sv2i(pac(71-i*4-2 downto 71-i*4-3));
            end loop ini;
            flag:=-1;
            flag1:=-1;
            sorting: for i in 1 to 15
            loop
            flag:=sort(0)(i);
            flag1:=sort(1)(i);
            j:=i-1;
            loop2: while ((j>=0) and flag1<sort(1)(j))
            loop
            sort(0)(j+1):=sort(0)(j);
            sort(1)(j+1):=sort(1)(j);
            j:=j-1;
            end loop loop2;
            sort(0)(j+1):=flag;
            sort(1)(j+1):=flag1;
            end loop sorting;
            finding: for i in 0 to 15
            loop
            if (id=sort(0)(i)) then
            order:=i;
            end if;
            end loop ;
            return order;
            end get_order;

      variable shared_reg, tran_reg: std_logic_vector(127 downto 0);
variable src, s_port, dest, d_port, tp_vec, sub_vec, time_left,crc
:std_logic_vector(7 downto 0);
      variable data_field : std_logic_vector(63 downto 0);

begin
if(rxd_ck='0') then rxd_ck<='1';
d_rx='0') then end if;
      case present_state is
      WHEN  start => id :=sv2i(node_id);
      mynode_id <= id;
```

```
for i in 0 to 3 loop
route_table(i)(10)<='0';
end loop;
for i in 0 to 3 loop
if(pstat(i)='1') then
route_table(i)(10)<='1';
bit_port(i)(10):='1';
BITPM(i) <='1';
else
BITPM(i)<='0';
route_table(i)(10)<='0';
bit_port(i)(10):='0';
end if;
end loop;

case pstat is
when "0000" => dg :=0;
when "0001" |"0010"|"0100"|"1000" => dg :=1;
when "0011" |"0110"|"1100"|"1001"|"1010"|"0101" => dg :=2;
when "0111" |"1110"|"1011"|"1101" => dg :=3;
when "1111" => dg :=4;
when others => null;
end case;
      if trans_ing = '0' then
      for i in 0 to portsize-1 loop
      port_to(i) <= (others => '0');
      end loop;
end if;
if (reset ='0') then
next_state <= start;
elsif (id =0) then
next_state <=ini_tx;
else
next_state<=waiting;
end if;
when ini_tx => src :="0000"&node_id(3 downto 0);
                  s_port :="0000"&pstat(3 downto 0);
dest :="11111111";

for i in 63 downto 0 loop
data_field(i) := '0';
end loop;

data_field(4*id+2) :='1';
data_field(4*id+1 downto 4*id+0):=int_vector(degree)(1 downto 0);
create_pac:for i in 0 to portsize-1
port_to(i)(127 downto 120 ) <= src(7 downto 0);
port_to(i)(119 downto 112 ) <= s_port;
port_to(i)(111 downto 104 ) <= dest;
port_to(i)(103 downto 96 ) <= d_port;
port_to(i)( 95 downto 88 ) <= tp_vec;
port_to(i)( 87 downto 80 ) <= sub_vec;
port_to(i)( 79 downto 16 ) <= data_field;
port_to(i)( 15 downto 8 ) <= time_left;
port_to(i)( 7 downto 0 ) <= crc;
end loop create_pac;
```

```
      if(d_rx'event=false and trans_ing'event=false
      and reset'event=false) then
      pro_act_ini(act_tx);
      end if;

      if (reset ='0') then
      next_state <= start;
      else
      next_state <= waiting;
      end if;
      when waiting =>if (reset='0') then
      next_state<=start;
      else
      if (d_rx'event and d_rx='1') then
      aport:=rxd_port;
            TIME_LEFT:=int_vector(sv2i(shared_reg(7 downto 0))-1);
                  if ((shared_reg(95 downto 88)="00000000") and
      (sv2i(TIME_LEFT)>0)) then
            next_state<=spec;
            tran_reg(127 downto 120):="0000"&node_id;
            tran_reg(119 downto 8):=shared_reg(119 downto 8);
            tran_reg(7 downto 0):=TIME_LEFT;
            bit_port(aport)(10):='1';

            finding: for i in 0 to 3
                  loop
if((bit_port(i)(10)='0') =bit_port(i)(9 downto 2))) then BITPM(i)<='0';
elsif ((shared_reg(127 downto 120)/=bit_port(i)(9 downto 2))and
(bit_port(i)(10)='1')) then
                  BITPM(i)<='1';
                  port_to(i)<=tran_reg;
                  end if;
            end loop finding;

      elsif ((shared_reg(95 downto 88)="00000001") and
(sv2i(TIME_LEFT)>0)) then
            route_table(aport)(10)<='1';
            route_table(aport)(9 downto 2)<=shared_reg(127 downto 120);
            next_state<=receive;
            else rxd_ck<='0';
            end if;
            else
            next_state<=waiting;
            end if ;
            end if;
WHEN spec => rxd_ck<='0';
if(d_rx'event=false and trans_ing'event=false and reset'event=false)
then
            pro_act_ini(act_tx);
      end if;

process:state machine
      if (trans_ing ='0' and trans_ing'event) then    next_state <=
start;
      else
      next_state<= spec;
      end if;
```

```
      rxd_ck<='0';
      when receive =>if (reset='0') then
      next_state<=start;
      else
      tran_reg(127 downto 120):="0000"&node_id;
      tran_reg(119 downto 8):=shared_reg(119 downto 8);
      tran_reg(7 downto 0):=TIME_LEFT;
      if (shared_reg(71-mynode_id*4-1)='0') then
      tran_reg(71-mynode_id*4-1 downto (71-mynode_id*4-3)):='1'&degree;
      tran_reg(79 downto 72):=int_vector(sv2i(shared_reg(79 downto
72))+1);
            finding2: for i in 0 to 3
            loop
            if(route_table(i)(10)='1') then
            BITPM(i)<='1';
            port_to(i)<=tran_reg;
            else
            BITPM(i)<='0';
            end if;
            end loop finding2;
      next_state<=transmit;
      finding5: for i in 0 to 3
      loop
if((route_table(i)(10)='0') or (shared_reg(127 downto
120)=route_table(i)(9 downto 2))) then
      BITPM(i)<='0';
      elsif ((shared_reg(127 downto 120)/=route_table(i)(9 downto 2))
and (route_table(i)(10)='1'))then
      BITPM(i)<='1';
      port_to(i)<=tran_reg;
      end if;
      end loop finding5;
      next_state<=transmit;
      end if;
      end if;
      when ending =>if (reset='0') then
      next_state<=start;
      else
      enable<='1';
      order<=int_vector(order);
      end if;
      end case;
           end process state_machine;
      state_clked: process(clock)
      begin
      if(clock'event and clock='1') then
      present_state <= next_state;
      end if;
      end process state_clked;

transmit: process (act_tx,txd_ck,txd_data,clock)
      variable j :integer :=0;
      variable control :integer :=0;
      begin
      if(clock'event and clock='1') then
```

```
            if txd_ck = '1' and txd_data = '0' and control=1 then
      if BITPM (j)='1' then   txd_port <= j;
      txd_data <='1';
            trans_ing<='1';
      j:=j+1;
      else j:=j+1;
      end if;
      if(j=4) then
      j:=0;
      control:=0;
      trans_ing<='0';
      end if;
      elsif txd_ck='0' then -- if txd_ck =0,means i/o device is still
transmitting,at present,the txd_data,should be 0
      txd_data <='0';
      end if;
      end if;
      end process transit;

intr_arb:process(reset,clock,intr_out_ck)
      begin
      if(clock'event and clock='1' and reset='0') then
      ck0<='1';
      ck1<='1';
      ck2<='1';
      ck3<='1';
      intr_out<='0';
      elsif(clock'event and clock ='1' and reset='1') then
      if(intr_out='0') then
            if(intr0='1') then
            intr_out<='1';
            port_num<="00";
            ck0<='0';
            elsif (intr1='1') then
            intr_out<='1';
            port_num<="01";
            ck1<='0';
            elsif (intr2='1') then
            intr_out<='1';
            port_num<="10";
            ck2<='0';
            elsif (intr3='1') then
            intr_out<='1';
            port_num<="11";
            ck3<='0';
            endif;
      end if;
      elsif(intr_out='1') then
                  case
                  port_num is
                  when "00" => ck0<='1';
                  when "01" => ck1<='1';
                  when "10" => ck2<='1';
                  when "11" => ck3<='1';
                  when others => null;
                  end case;
            end if;
```

```
      end if;
end process intr_arb;

intrrx:process(reset,clock,intr_out,rxd_ck)

      variable bitm: integer:=0;
      variable count : integer :=0;
      variable clock_count: integer :=0;
      begin
      if(clock'event and clock='1' and reset='0') then
      dbus<="ZZZZZZZZZZZZZZZZ";
      addbus<="ZZZZZZZZZZZZZZZZ";
      rw<='Z';
      elsif (clock'event and clock='1' and reset='1' and
      intr_out_ck='0') then
      intr_out_ck<='1';
      elsif (rxd_ck='0' and d_rx='1' and
      clock'event and clock='0') then
      d_rx<='0';
      elsif (clock'event and clock='1' and reset='1') then
      if( intr_out='1' and d_rx='0' )     then if(dbus_busy='0' or
rec_begin='1') then
      rec_begin<='1';
      if(count<8) then
      if(clock_count=0) then

            case count is
            when 0 => addbus<="0000000000000000";
            when 1 => addbus<="0000000000000001";
            when 2 => addbus<="0000000000000010";
            when 3 => addbus<="0000000000000011";
            when 4 => addbus<="0000000000000100";
            when 5 => addbus<="0000000000000101";
            when 6 => addbus<="0000000000000110";
            when 7 => addbus<="0000000000000111";
            when others => addbus<="ZZZZZZZZZZZZZZZZ";
            end case;
      rw<='0';
      clock_count:=clock_count+1;
            case port_num is
            when "00" => bitm:=0;
            when "01" => bitm:=1;
            when "10" => bitm:=2;
            when "11" => bitm:=3;
            when others => null;
            end case;
      elsif(clock_count=2) then
      in_port(bitm)((count+1)*16-1 downto count*16)<=dbus;
      clock_count:=clock_count+1;
      elsif(clock_count=5) then
      addbus<="ZZZZZZZZZZZZZZZZ";
      rw<='Z';
      clock_count:=0;
      count:=count+1;
      else
      clock_count:=clock_count+1;
      end if;
```

```
      else  d_rx<='1';
      rxd_port<=bitm;
      count:=0;
      intr_out_ck<='0';
      rec_begin<='0';
      end if;
      end if;
      end if;
      end if;
end process intrrx;

dbus_arb: process(rec_begin,txd_begin,trans_ing)
      begin
      if(rec_begin='1' or txd_begin='1') then
      dbus_busy<='1';
      else if(trans_ing='0') then
            dbus_busy<='0';
            end if;
      end if;
end process dbus_arb;

chip_select: process(reset,rec_begin,txd_begin)
      variable count: integer :=0;
      begin
      if(clock'event and clock='1' and reset='0') then
      cs0<='1';
      cs1<='1';
      cs2<='1';
      cs3<='1';
      elsif (reset='1') then
      if(rec_begin='1') then
            case port_num is
            when "00"=>cs0<='0';
            when "01"=>cs1<='0';
            when "10"=>cs2<='0';
            when "11"=>cs3<='0';
            when others=> null;
            end case;
      elsif(txd_begin='1') then
            case txd_port is
            when 0=>cs0<='0';
            when 1=>cs1<='0';
            when 2=>cs2<='0';
            when 3=>cs3<='0';
            when others=> null;
      end case;

      elsif(rec_begin='0' and txd_begin='0') then
      s0<='1';
      s1<='1';
      s2<='1';
      s3<='1';
      end if;
      end if;
end process chip_select;
end arch_behavioral;
```

```
-- TURN PROHIBITION

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY prohibit IS
PORT(clock        : IN std_logic;
      node_num    : IN std_logic_vector(3 DOWNTO 0);
      enable            : IN std_logic;
      token             : IN std_logic_vector(3 DOWNTO 0);
      data        : INOUT std_logic_vector(15 DOWNTO 0);
      addr        : INOUT std_logic_vector(15 DOWNTO 0);
      buf_read    : OUT std_logic;
      buf_write   : OUT std_logic;
      intr0             : IN std_logic;
      intra0            : OUT std_logic;
      intr1             : IN std_logic;
      intra1            : OUT std_logic;
      intr2             : IN std_logic;
      intra2            : OUT std_logic;
      intr3             : IN std_logic;
      intra3            : OUT std_logic;
      mem_cs      : OUT std_logic;
      mem_rw      : OUT std_logic);
END prohibit;

ARCHITECTURE behaviorale OF prohibit IS

SIGNAL prohibit_enable, prohi_chk, brdcst_diff_ports  : std_logic;
SIGNAL io_enable, brdcst_pak_req, bus_assn      : std_logic := '0';
SIGNAL intr_write_out, write_out, timeout, token_relsd      : std_logic
:='0';
SIGNAL pak_exam_cmplt         : std_logic :='1';
SIGNAL paks_brdcsted, pak_exam            : std_logic := '0';
SIGNAL pac_reg1,  packet_register_2             : std_logic_vector(79
DOWNTO 0);
      SIGNAL from_port, to_port           : std_logic_vector(1 DOWNTO
0);
      TYPE matrix4x4 IS ARRAY(3 DOWNTO 0, 3 DOWNTO 0) OF std_logic;
SIGNAL matp:
matrix4x4:=(('0','0','0','0'),('0','0','0','0'),('0','0','0','0'),('0',
'0','0','0'));
TYPE matrix3x3 IS ARRAY(2 DOWNTO 0) OF std_logic_vector(1 DOWNTO 0);
SIGNAL diff_ports : matrix3x3 := ("00","00","00");
CONSTANT one : natural := 1;
      BEGIN
            --FIRST PROCESS
            Prohibit_start : PROCESS(clock,enable, token_relsd)

            BEGIN
                  if ( clock='1' AND enable='1') then
                        prohibit_enable <= '1';
                  end if ;
                  if (token_relsd = '1') then
                        prohibit_enable <= '0';
                  end if ;
            END PROCESS Prohibit_start;
```

```
            --SECOND PROCESS
            token_check : PROCESS(clock, token, to_port, write_out)
            VARIABLE state : INTEGER := 0;
            BEGIN
                        IF rising_edge(clock) THEN
                        IF token = "0001" THEN
                              brdcst_pak_req01 <= '1';
                              CASE state IS
                              WHEN 0 => state := 1;
                              WHEN 1 => state := 2;
                     WHEN 2 => IF (to_port = "11" AND write_out = '1')
THEN
                                          state := 3;
                                    END IF;
                              WHEN 3 => IF(write_out = '0') THEN
                                          write_out <= '0';
                                          brdcst_pak_req01 <= '0';
                                          state := 0;
                                    END IF;
                              WHEN OTHERS => NULL;
                              END CASE;
                        END IF;
                  END IF;
            END PROCESS token_check;

            --THIRD PROCESS
cntrl1 : PROCESS(clock,intr0,intr1,intr2,intr3, intr_write_out,
pak_exam_cbitpmlt, prohibit_enable, bus_assn)
            BEGIN
IF (rising_edge(clock) AND rising_edge(pak_exam_cbitpmlt) AND
(rising_edge(intr0) OR rising_edge(intr1) OR rising_edge(intr2) OR
rising_edge(intr3) OR rising_edge(intr_write_out))) THEN
                        IF prohibit_enable ='1' AND bus_assn='0' THEN
                              io_enable <= '1';
                        ELSE
                              io_enable <= '0';
                        END IF;
                  END IF;
            END PROCESS cntrl1;

            --FOURTH PROCESS
            cntrl2 : PROCESS
            VARIABLE state, inner_state1, inner_state2 : INTEGER := 0;
            BEGIN
                  CASE state IS
      WHEN 0 => IF rising_edge(pak_exam) AND falling_edge(bus_assn)
THEN
                                    pak_exam_cbitpmlt <= '0';
                                    state := 1;
                                    END IF;
      WHEN 1 => IF rising_edge(clock) THEN
                                    pak_exam_cbitpmlt <='1';
                                    state := 2;
                              END IF;
      WHEN 2 => IF rising_edge(clock) THEN
      IF brdcst_pak_req = '1' THEN
```

```
      CASE inner_state1 IS
                WHEN 0 => IF to_port = "11" AND rising_edge(write_out)
THEN inner_state1 := 1;
                END IF;
      WHEN 1 => IF falling_edge(write_out) THEN brdcst_pak_req <= '0';
                                    inner_state1 := 0;
                                    state := 0;
                END IF;
      WHEN OTHERS => NULL;
                END CASE;
      ELSIF prohi_chk = '1' THEN
      WAIT ON matp, token_relsd;
      prohi_chk <= '0';
      ELSIF brdcst_diff_ports = '1' THEN
      CASE inner_state2 IS
WHEN 0 => IF to_port=diff_ports(2) AND rising_edge(write_out) THEN
      inner_state2 := 1;
            END IF;
      WHEN 1 => IF falling_edge(write_out) THEN

      brdcst_diff_ports <= '0';
      inner_state2 := 0;
      state := 0;
      END IF;
            WHEN OTHERS => NULL;
      END CASE;
            END IF;
      END IF;
      WHEN OTHERS => NULL;
      END CASE;
END PROCESS cntrl2;

            --FIFTH PROCESS
            bus_arbit_io : PROCESS(clock, io_enable, token_relsd,
bus_assn, matp, intr_write_out, intr3, intr2, intr1, intr0)
            VARIABLE intr : std_logic_vector(4 DOWNTO 0) := "00000";
            VARIABLE read_state, write_state, state : INTEGER := 0;
            variable intra : std_logic_vector(2 downto 0);
            PROCEDURE io_read_opr(state : INOUT INTEGER) IS
                  VARIABLE j : integer RANGE 0 TO 127;
                  BEGIN
                        FOR i IN 4 DOWNTO 0 LOOP
                              addr <= std_logic_vector(unsign(i, 16));
                              j := ((i+1)*16)-1;
                              buf_read <= '1';
                              CASE state IS
            WHEN 0 =>   IF rising_edge(clock) THEN
                              pac_reg1(j DOWNTO (j-15)) <= data;
                                                buf_read <= '0';
                                                state := 1;
                        END IF;
            WHEN 1 =>  IF rising_edge(clock) THEN
                                          state:= 0;
                        END IF;
            WHEN OTHERS => NULL;
                        END CASE;
                        END LOOP;
```

```
                        CASE intra IS
                              WHEN "001" => from_port <= "00";
                              WHEN "010" => from_port <= "01";
                              WHEN "011" => from_port <= "10";
                              WHEN "100" => from_port <= "11";
                              WHEN OTHERS => NULL;
                        END CASE;
                        pak_exam <= '1';
                  END io_read_opr;

            PROCEDURE opr_write(state : INOUT INTEGER) IS
                  VARIABLE j : integer RANGE 0 TO 127;
                  BEGIN
                        CASE to_port IS
                              WHEN "00" => intra0 <= '1';
                              WHEN "01" => intra1 <= '1';
                              WHEN "10" => intra2 <= '1';
                              WHEN "11" => intra3 <= '1';
                              WHEN OTHERS => NULL;
                        END CASE;
                        FOR i IN 4 DOWNTO 0 LOOP
                              addr <= std_logic_vector(unsign(i+1,
16));
                              j := ((i+1)*16)-1;
                              buf_write <= '1';
                              CASE state IS
                  WHEN 0 => IF rising_edge(clock) THEN
                        data <= packet_register_2(j DOWNTO (j-15));
                              buf_write <= '0';
                              state := 1;
                                          END IF;
                  WHEN 1 => IF rising_edge(clock) THEN
                              state := 0;
                              END IF;
                  WHEN OTHERS => NULL;
                              END CASE;
                                    END LOOP;
                  END opr_write;


            BEGIN
IF rising_edge(clock) AND ((io_enable='1'OR token_relsd ='1') AND
bus_assn='0') THEN

                  IF token_relsd = '1' THEN
                        bus_assn <='1';
                        mem_cs <= '1';
                        CASE state IS
                        WHEN 0=> IF rising_edge(clock) THEN
                                          mem_rw <= '1';
                                          addr <= "0000000000000001";
            data(15 DOWNTO 0) <=

      matp(0,0)&matp(0,1)&matp(0,2)&matp(0,3)&

      matp(1,0)&matp(1,1)&matp(1,2)&matp(1,3)&
```

```
      matp(2,0)&matp(2,1)&matp(2,2)&matp(2,3)&

      matp(3,0)&matp(3,1)&matp(3,2)&matp(3,3);
                                          state := 1;
                                    END IF;
                              WHEN 1 => IF rising_edge(clock) THEN
                                    mem_cs <= '0';
                                    bus_assn <= '0';
                                    state := 0;
                                 END IF;
                              WHEN OTHERS => NULL;
                              END CASE;
                        ELSE
                              bus_assn <= '1';
                              pak_exam <= '0';
                              intr := intr_write_out & intr3 & intr2 & intr1
& intr0;
                              IF std_match(intr, "1----") THEN
                                    write_out <= '1';
                                    intra := "000";
                              ELSIF std_match(intr, "00001") THEN
                                    intra0 <= '1';
                                    intra := "001";
                              ELSIF std_match(intr, "0001-") THEN
                                    intra1 <= '1';
                                    intra := "010";
                              ELSIF std_match(intr, "001--") THEN
                                    intra2 <= '1';
                                    intra := "011";
                              ELSIF std_match(intr, "01---") THEN
                                    intra3 <= '1';
                                    intra := "100";
                              END IF;
                              CASE intra IS
                                    WHEN "000" => opr_write(write_state);
                                          write_out <= '0';
                                    WHEN "001" => io_read_opr(read_state);
                                          intra0 <= '0';
                                    WHEN "010" => io_read_opr(read_state);
                                          intra1 <= '0';
                                    WHEN "011" => io_read_opr(read_state);
                                          intra2 <= '0';
                                    WHEN "100" => io_read_opr(read_state);
                                          intra3 <= '0';
                                    WHEN OTHERS => NULL;
                              END CASE;
                              bus_assn <= '0';
                        END IF;
                  END IF;
                  END PROCESS bus_arbit_io;

                  -- \SIXTH PROCESS
pak_brdcst : PROCESS (clock, from_port, pac_reg1, brdcst_diff_ports,
      timeout, brdcst_pak_req, write_out, node_num, diff_ports, token,
                  prohibit_enable, bus_assn)
```

```
            VARIABLE inner_state1, inner_state2, inner_state3 : INTEGER
:= 0;
            VARIABLE inter_state1, inter_state2, inter_state3 : INTEGER
:= 0;
            BEGIN
                  IF rising_edge(clock) AND (brdcst_pak_req='1' OR
            brdcst_pak_req01='1'OR brdcst_diff_ports='1' OR timeout
='1') THEN
                        IF brdcst_pak_req='1' OR brdcst_pak_req01='1'
THEN
                              CASE inner_state1 IS
                                    WHEN 0 =>FOR i IN 0 TO 3 LOOP
                                          CASE inter_state1 IS
                        WHEN 0 => IF rising_edge(clock) AND write_out = '0'
THEN
            packet_register_2(79 DOWNTO 72) <= "0000" & node_num
      packet_register_2(71 DOWNTO 64) <= std_logic_vector(unsign(i,8));
            packet_register_2(63 DOWNTO 56) <= "0000" & node_num;
            packet_register_2(55 DOWNTO 48) <= "11111111";
            packet_register_2(47 DOWNTO 40) <= "00000010";
      packet_register_2(39 DOWNTO 32) <= "00000000";
      packet_register_2(31 DOWNTO 16) <= "0000000000000000";
                  packet_register_2(15 DOWNTO 8) <= "00100000";
      packet_register_2(7 DOWNTO 0) <= "00000000";
      to_port <= std_logic_vector(unsign(i,2));
      intr_write_out <= '1';
      inter_state1 := 1;
END IF;
WHEN 1 => IF rising_edge(clock) AND write_out = '1' THEN
            END IF;
WHEN OTHERS => NULL;
END CASE;
END LOOP;
      inner_state1 := 1;
WHEN 1 => If rising_edge(clock) AND write_out = '0' THEN
                                                paks_brdcsted <= '1';
                                                inner_state1 := 0;
                                          END IF;
                                    WHEN OTHERS => NULL;
                              END CASE;
                        ELSIF brdcst_diff_ports='1' THEN
                              CASE from_port IS
                                    WHEN "00" =>

      diff_ports(0)<="01";diff_ports(1)<="10";diff_ports(2)<="11";
                                    WHEN "01" =>

      diff_ports(0)<="00";diff_ports(1)<="10";diff_ports(2)<="11";
                                    WHEN "10" =>

      diff_ports(0)<="00";diff_ports(1)<="01";diff_ports(2)<="11";
                                    WHEN "11" =>

      diff_ports(0)<="00";diff_ports(1)<="01";diff_ports(2)<="10";
                                    WHEN OTHERS => NULL;
                              END CASE;
                              CASE inner_state2 IS
```

```
                                        WHEN 0 =>FOR i IN 0 TO 2 LOOP
                                                CASE inter_state2 IS
                                                      WHEN 0 => IF
rising_edge(clock) AND write_out = '0' THEN

      packet_register_2(79 DOWNTO 72) <= pac_reg1(79 DOWNTO 72);
      packet_register_2(71 DOWNTO 64) <= pac_reg1(71 DOWNTO 64);
      packet_register_2(63 DOWNTO 56) <= pac_reg1(63 DOWNTO 56);
      packet_register_2(55 DOWNTO 48) <= pac_reg1(55 DOWNTO 48);
      packet_register_2(47 DOWNTO 40) <= pac_reg1(47 DOWNTO 40);
      packet_register_2(39 DOWNTO 32) <= pac_reg1(39 DOWNTO 32);
      packet_register_2(31 DOWNTO 16) <= pac_reg1(31 DOWNTO 16);
      packet_register_2(15 DOWNTO 8) <= pac_reg1(15 DOWNTO 8);
      packet_register_2(7 DOWNTO 0) <= pac_reg1(7 DOWNTO 0);
      to_port <= diff_ports(i);
      intr_write_out <= '1';
      inter_state2 := 1;
      END IF;
WHEN 1 => IF rising_edge(clock) AND write_out = '1' THEN
      intr_write_out <= '0';
inter_state2 := 0;
END IF;
WHEN OTHERS => NULL;
END CASE;
END LOOP;
inner_state2 := 1;
WHEN 1 => IF rising_edge(clock) AND write_out = '0' THEN
inner_state2 := 0;
END IF;
WHEN OTHERS => NULL;
END CASE;
packet_register_2(79 DOWNTO 72) <= "0000" & node_num;
packet_register_2(71 DOWNTO 64) <= std_logic_vector(unsign(i,8))
packet_register_2(63 DOWNTO 56) <= "00000000";
packet_register_2(55 DOWNTO 48) <= "11111111";
packet_register_2(47 DOWNTO 40) <= "00000010";
packet_register_2(39 DOWNTO 32) <= "1111" & token;
packet_register_2(31 DOWNTO 16) <= "0000000000000000";
packet_register_2(15 DOWNTO 8) <= "00100000"; -- TIME_LEFT( 1 BYTE ) =>
20H

      packet_register_2(7 DOWNTO 0) <= "00000000"; -- CRC( 1 BYTE ) =>
00
                        to_port <= std_logic_vector(unsign(i,2));

      intr_write_out <= '1';

      inter_state3 := 1;
      END IF;
      WHEN 1 => IF rising_edge(clock) AND rising_edge(write_out) THEN
                              END IF;
            WHEN OTHERS => NULL;
            END CASE;
                  END LOOP;
                  inner_state3 := 1;
            WHEN 1 => IF rising_edge(clock) AND falling_edge(write_out)
THEN                        token_relsd <= '1';
```

```
                              inner_state3 := 2;
                    END IF;
      WHEN 2 => IF rising_edge(clock) AND falling_edge(prohibit_enable)
AND rising_edge(bus_assn) THEN
                    inner_state3 := 3;
              END IF;
      WHEN 3 => IF rising_edge(bus_assn) THEN
                        token_relsd <= '0';
                        paks_brdcsted <= '0';
                        inner_state3 := 0;
      END IF;
           WHEN OTHERS => NULL;
            END CASE;
                        END IF;
                    END IF;
            END PROCESS pak_brdcst;

            --SEVENTH PROCESS
            prohibit_chk : PROCESS
            BEGIN
                  WAIT ON clock UNTIL clock='1' AND prohi_chk='1';
                  matp(to_integer(unsigned(pac_reg1(71 DOWNTO 64))) ,
                        to_integer(unsigned(from_port))) <= '1' ;
                  matp(to_integer(unsigned(from_port)),
                  to_integer(unsigned(pac_reg1(71 DOWNTO 64)))) <= '1';
            END PROCESS prohibit_chk;

      END behaviorale;



-- ROUTING TABLE UNIT
-- Intialize
LIBRARY ieee ;
USE ieee.std_logic_1164.ALL ;
ENTITY intialize IS
PORT ( enable : IN bit ;
node_id : IN integer ;
data_out : OUT integer ;
row, column : OUT integer ;
write_d, write_r, done : OUT bit ;
state : IN bit_vector(5 downto 0) ;
clock : IN std_logic);
END intialize ;
ARCHITECTURE behavioral OF intialize IS
SIGNAL ii, jj : integer := 0 ;
FUNCTION bvec2int --bit_vec2int
(SIGNAL bit_vec : IN bit_vector (15 downto 0))
RETURN integer IS
VARIABLE int_value, i : integer := 0;
BEGIN
int_value := 0 ;
i := 0 ;
IF bit_vec(i) = '1' THEN
int_value := int_value + 2**i ;
END IF ;
```

```
i := i + 1 ;
RETURN int_value ;
END bvec2int ;

FUNCTION integer_vec --int2bit_vec
(SIGNAL int_in : IN integer)
RETURN bit_vector IS
VARIABLE bit_vec_value : bit_vector(15 downto 0) ;
VARIABLE j, int_bitm: integer ;
BEGIN
bit_vec_value := "0000000000000000" ;
int_bitm := int_in ;
j := 0 ;
-- vec:LOOP
IF int_bitm >= 2**(15-j) THEN
bit_vec_value(15-j) := '1' ;
int_bitm := int_bitm - 2**(15-j) ;
ELSE
bit_vec_value(15-j) := '0' ;
END IF ;
j := j + 1 ;
RETURN bit_vec_value ;
END integer_vec ;
BEGIN

S0: PROCESS(clock,enable,state)
VARIABLE r,c, do : integer := 0 ;
VARIABLE i,ii,jj,j : integer := 0 ;
VARIABLE wd, wr : bit := '0' ;
BEGIN
if ( clock = '1' AND enable = '1' AND state = "000001") then
done <= '0' ;
write_d <= wd ;
write_r <= wr ;
row <= jj ;
column <= ii ;
data_out <= do ;
wd := '1' ;
wr := '1' ;
ii := i ;
jj := j ;
IF node_id = i THEN
do := 0 ;
ELSE
do := 255 ;
END IF ;
IF j = 4 THEN
j := 0 ;
END IF ;
j := 1 + j ;
end if ;
i := i + 1 ;
IF i = 16 THEN
done <= '1' ;
END IF ;
if(rising_edge(clock)) then
done <= '0' ;
```

```
write_d <= wd ;
write_r <= wr ;
row <= jj ;
column <= ii ;
data_out <= do ;
wd := '0' ;
wr := '0' ;
END if;
END PROCESS S0 ;
END behavioral ;

-- Table_buld
LIBRARY ieee ;
USE ieee.std_logic_1164.ALL ;
ENTITY table_buld IS
PORT ( r5, c5 : OUT integer ;
d_in : IN integer ;
tx_count : OUT bit ;
state : IN bit_vector( 5 downto 0);
count : IN integer ;
pr : OUT bit_vector (1 to 4) ;
p_in : IN bit ;
p_read : OUT bit ;
d_read : OUT bit ;
s5_done : OUT bit ;
clock : IN std_logic );
END table_buld ;
ARCHITECTURE behavioral OF table_buld IS
SIGNAL tx_array : bit_vector(15 downto 0) ;
BEGIN
TRANSMIT:PROCESS(clock)
VARIABLE i, k, pt, ta : integer ;
BEGIN
if (clock = '1' and state(3) = '1') then
pt := 1 ;
end if ;
tx_array <= "0000000000000000" ;
i := 0 ;k := 0 ;r5 <= i ;c5 <= k ;
d_read <= '1';
if (clock'event and clock = '1') then
d_read <= '0' ;
end if;
if (clock'event and clock = '1') then
IF d_in = count - 1 THEN
r5 <= pt ;c5 <= i ;p_read <= '1';

if(clock'event and clock = '1') then
p_read <= '0' ;
end if ;

if(clock'event and clock = '1') then
IF p_in = '0' THEN
tx_array(k) <= '1' ;
END IF ;
end if ;
END IF ;
END IF;
```

```
i := i + 1 ;
IF i = 5 THEN
i := 0 ;
END IF ;
k := k + 1 ;
IF k = 16 THEN
ta := 1 ; pr(pt) <= '1' ;

--TRANSMIT:LOOP -----
if (clock'event and clock = '1') then
tx_count <= tx_array(ta) ;
ta := ta + 1 ;
-- END LOOP ; ------
END IF ;
end if;
IF k = 16 THEN
k := 0 ;
END IF ;
-- END LOOP ;

if(clock'event and clock ='1') then
pr(pt) <= '0' ;
pt := pt + 1 ;
end if ;
IF pt = 5 THEN
pt := 1 ;
END IF ;
-- END LOOP ;

s5_done <= '1' ;
if (clock'event and clock = '1') then
s5_done <= '0' ;
end if ;
END PROCESS TRANSMIT ;
END behavioral ;

-- Array_read
LIBRARY ieee ;
USE ieee.std_logic_1164.ALL ;
ENTITY array_read IS
PORT( d_bus : IN std_logic_vector(15 downto 0) ;
state : IN bit_vector(5 downto 0) ;
s4_done : OUT bit ;
ck : OUT bit_vector ( 1 to 4 ) ;
interupt : IN bit_vector (1 to 4 ) ;
w_s4 : OUT bit ;
clock : IN std_logic ;
count : IN integer ;
r4 , c4 : OUT integer) ;
END array_read ;
ARCHITECTURE behavioral OF array_read IS
BEGIN
READ_N:PROCESS(clock)
VARIABLE pr, k : integer ;
BEGIN
if (clock'event and clock = '1' ) then
if (state(2) = '1')then
```

```
pr := 1 ;
      IF interupt(pr) = '1' THEN
      ck(pr) <= '1' ;
      k := 0 ;
            If(clock'event and clock = '1') then
            -- RREAD:LOOP
            IF bvec2int(d_bus) = count - 1 THEN
            r4 <= pr ;
            c4 <= k ;
            w_s4 <= '1' ;
            END IF;
                  If (clock'event and clock = '1') then
                  w_s4 <= '0' ;
                  k := k + 1 ;
                  -- EXIT WHEN k = 16 ;
                  END IF;
            END IF ;
      END IF ;
      ck(pr) <= '0' ;
      -- WAIT UNTIL (clock'EVENT AND clock = '1') ;
      IF (clock'event and clock = '1') then
      pr := pr + 1 ;
      -- EXIT WHEN pr = 5 ;
      END IF ;

      s4_done <= '1' ;
      if (clock'event and clock = '1') then
      s4_done <= '0' ;
      End IF;
END IF;
End if;
END PROCESS READ_N ;
END behavioral ;

-- Status
LIBRARY ieee ;
USE ieee.std_logic_1164.ALL;
ENTITY status IS
PORT(clock : INOUT std_logic ;
state : OUT bit_vector(5 downto 0);
state_done : IN bit_vector(5 downto 0);
enable : IN bit ;
done : OUT bit ;
count : IN integer );
END status ;
ARCHITECTURE behavioral OF status IS

BEGIN
S0:PROCESS (enable, clock )
VARIABLE st, sd : bit_vector(5 downto 0);
BEGIN

If (clock'event and clock = '1' ) then
if enable = '0' then
state <= "000000" ; -- default state
ELSE if ( clock = '1') then
state <= st ; --output latch
```

```
sd := state_done ; --input latch

    IF st = "000000" THEN
    st := "000001" ;
    END IF ;

CASE sd IS
WHEN "000001" => st(1) := '1' ;
st(0) := '0' ;
WHEN "000010" => st(2) := '1' ;
st(1) := '0' ;
WHEN "000100" => st(3) := '1' ;
st(2) := '0' ;
WHEN "001000" => st(1) := '1' ;
st(3) := '0' ;
IF count = 15 THEN
st(4) := '1' ;
st(2) := '0' ;
done <= '1' ;
END IF ;
WHEN "010000" => st(0) := '1' ;
st(4) := '0' ;
WHEN OTHERS => NULL ;
END CASE ;
End if ;
END IF ;
end if ;
END PROCESS S0 ;
END behavioral ;

-- Record_renew
LIBRARY ieee ;
USE ieee.std_logic_1164.ALL ;
ENTITY record_renew IS
PORT ( state : IN bit_vector(5 downto 0) ;
p_in : IN bit ;
clock : IN std_logic;
row, column : OUT integer ;
r_in, d_in : IN integer ;
s3_done, write_r : OUT bit ;
count : IN integer ;
read_p : OUT bit ;
r_data : OUT integer );
END record_renew ;
ARCHITECTURE behavioral OF record_renew IS
BEGIN
UPDATE_R : PROCESS
VARIABLE i ,j, k : integer ;
BEGIN
i := 0 ;
j := 0 ;
k := 0 ;
row <= 0 ;
column <= 0 ;
 IF (clock'event and clock = '1')  then
 if state(1) = '1' THEN
--    OUTER:LOOP
```

```
        k := 0 ;
                    row <= i ; ----address d array for read
                    column <= k ;
                    IF (Clock'event and clock = '1') THEN
                    IF d_in = count - 1 THEN
                    j := 0 ;
                    --      WRITE:LOOP
                            r_data <= i ;
                            row <= i ; ----addrss p array for read
                            column <= j ;
                              read_p <= '1';
                            IF (CLOCK'event and clock = '1') THEN
                              read_p <= '0' ;
                            END IF ;
                            IF p_in = '0' THEN
                            row <= j ; ----address r array for read
                            column <= k ;
                            if (clock'event and clock = '1') then
                                  IF r_in = 255 THEN
                                  row <= j ; ---address r array for write
                                  column <= k ;
                                  write_r <= '1' ;
                                  IF (CLOCK'event and clock = '1') THEN
                                  write_r <= '0' ;
                                  END IF ;
                                  END IF;
                            END IF;
                            END IF ;
                            --END LOOP WRITE;
                    END IF;
                    END IF ;

END IF ;
END IF;
s3_done <= '1' ;
WAIT UNTIL (clock'EVENT AND clock ='1');
s3_done <= '0' ;
-- WAIT UNTIL (clock'EVENT AND clock = '1') ;
END PROCESS UPDATE_R ;
END behavioral ;
```

APPENDIX B

TOP LEVEL VHDL MODEL

This Appendix gives the listing of top level VHDL model for simulation in VHDL.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.ALL ;
ENTITY all_route IS
PORT(
id : IN bit_vector(15 downto 0) ;
start : IN bit ;
ck : OUT bit_vector(1 to 4) ;
interupt : IN bit_vector(1 to 4) ;
r_w : OUT bit ;
d_bus : INOUT std_logic_vector(15 downto 0) ;
done : OUT bit) ;
END all_route ;

ARCHITECTURE all_route OF all_route IS
SIGNAL clock : std_logic;
SIGNAL tx_count, p_in, p_read, d_read : bit ;
SIGNAL s0write_d, s0write_r, w_s3, s0_done, s3_done, p_in, read_p,
w_s4, s4_done : bit :='0';
SIGNAL node_id, d_in, r_in, r5, c5 : integer ;
SIGNAL init_data, row, column, r3, c3, r4, c4, count, r_data, d_ins5 :
integer ;
TYPE my_array IS ARRAY (natural RANGE 0 to 4 , natural RANGE 0 to 15)
OF integer ;
SIGNAL array_r, array_d : my_array ;
SIGNAL state_done, state : bit_vector(5 downto 0);
SIGNAL d_buf : std_logic_vector(15 downto 0) ;
TYPE p_array IS ARRAY (natural RANGE 0 to 4 , natural RANGE 0 to 4) OF
bit ;
SIGNAL pr, ckn : bit_vector( 1 to 4) ;
SIGNAL parray : p_array :=
(('0','0','0','0','0'),('0','0','1','0','0'),('0','1','0','0','0'),
                    ('0','0','0','0','0'),('0','0','0','0','0'));

COMPONENT intialize
PORT (enable : IN bit;node_id : IN integer ;data_out : OUT integer ;row
: OUT integer ; column : OUT integer ;
      write_d : OUT bit ; write_r : OUT bit ;done : OUT bit;clock : IN
std_logic ;state : IN bit_vector(5 downto 0));
END COMPONENT ;

COMPONENT status
PORT ( clock : INOUT std_logic ;state : OUT bit_vector(5 downto 0)
;state_done : IN bit_vector(5 downto 0);enable : IN bit ;
      done : OUT bit ;count : IN integer );
END COMPONENT ;
```

```
COMPONENT record_renew
PORT (state : IN bit_vector(5 downto 0) ;clock : IN std_logic ;p_in :
IN bit ;row : OUT integer ;column : OUT integer ;
      r_in : IN integer ; d_in : IN integer ; s3_done : OUT bit
;write_r : OUT bit ;count : IN integer ;read_p : OUT bit ;
      r_data : OUT integer);
END COMPONENT ;

COMPONENT array_read
PORT (d_bus : IN std_logic_vector(15 downto 0) ;state : IN bit_vector(5
downto 0) ;s4_done : OUT bit ;
      ck : OUT bit_vector(1 to 4) ;interupt : IN bit_vector(1 to 4)
;w_s4 : OUT bit ;clock : IN std_logic ;count : IN integer ;
      r4 : OUT integer ; c4 : OUT integer) ;
END COMPONENT ;

COMPONENT table_buld
PORT (r5 : OUT integer ; c5 : OUT integer ;d_in : IN integer ;tx_count
: OUT bit ;state : IN bit_vector(5 downto 0);
     count : IN integer ;pr : OUT bit_vector(1 to 4) ;p_in : IN bit
;p_read : OUT bit ;d_read : OUT bit ;s5_done : OUT bit ;
     clock : IN std_logic );
END COMPONENT ;

FOR ALL :table_buld USE ENTITY work.table_buld(behavioral) ;
FOR ALL :array_read USE ENTITY work.array_read(behavioral) ;
FOR ALL :record_renew USE ENTITY work.record_renew(behavioral) ;
FOR ALL :intialize USE ENTITY work.intialize(behavioral) ;
FOR ALL :status USE ENTITY work.status(behavioral) ;


FUNCTION bv2st (in_data : IN bit_vector)
RETURN std_logic_vector IS
VARIABLE bitm : std_logic_vector(15 downto 0);
BEGIN
FOR i IN 0 to 15
LOOP
IF in_data(i) = '1' THEN
bitm(i) := '1' ;
ELSE
bitm(i) := '0' ;
END IF ;
END LOOP ;
RETURN bitm ;
END bv2st ;


BEGIN
node_id <= bvec2int(id) ;

COUNTER:PROCESS
BEGIN
WAIT ON clock UNTIL (clock'EVENT AND clock = '1') ;
-- WAIT until rising_edge(clock);
IF start = '0' THEN
count <= 0 ;
```

```
END IF ;
IF (state_done(0) OR state_done(3)) = '1' THEN
count <= count + 1 ;
END IF ;
END PROCESS COUNTER ;

U1:intialize
PORT MAP ( start, node_id, init_data, row, column, s0write_d,
s0write_r, s0_done, clock, state) ;
state_done(0) <= s0_done ;

U2:status
PORT MAP ( clock, state, state_done, start, done, count ) ;

U3:record_renew
PORT MAP (state, clock, p_in, r3, c3, r_in, d_in, s3_done, w_s3, count,
read_p, r_data) ;
state_done(1) <= s3_done ;

U4:array_read
PORT MAP (d_bus, state, state_done(2) , ckn, interupt, w_s4, clock,
count , r4, c4) ;
ck <= ckn ;

U5:table_buld
PORT MAP (r5, c5, d_ins5, tx_count, state, count, pr, p_ins5, p_read,
d_read, state_done(3), clock);

WRITE_D:PROCESS(clock)
VARIABLE s : bit_vector(5 downto 0) ;
BEGIN
-- WAIT ON clock UNTIL (clock'event AND clock = '1') ;
if (clock'event and clock = '1') then
s := state ;
CASE s IS
WHEN "000001" => IF s0write_d = '1' THEN
array_d(row,column) <= init_data ;
END IF ;
WHEN "000010" =>d_in <= array_d(r3,c3) ;
WHEN "000100" => IF w_s4 = '1' THEN
array_d(r4,c4) <= count ;
END IF ;
WHEN "001000" =>
IF d_read = '1' THEN
d_ins5 <= array_d(r5,c5) ;
ELSIF p_read = '1' THEN
p_ins5 <= parray(r5,c5) ;
END IF ;
WHEN OTHERS => NULL ;
END CASE ;
End if ;
END PROCESS WRITE_D ;

WRITE_R:PROCESS(clock)
VARIABLE s : bit_vector(5 downto 0) ;
BEGIN
if(clock'event and clock ='1') then
```

```
s:= state ;
CASE s IS
WHEN "000001" =>
IF s0write_r = '1' THEN
array_r(row,column) <= init_data ;
END IF ;
WHEN "000010" =>
IF w_s3 = '1' THEN
array_r(r3,c3) <= r_data ;
ELSIF read_p = '1' THEN
p_in <= parray(r3,c3);
ELSE
r_in <= array_r(r3,c3);
END IF ;
WHEN OTHERS => NULL ;
END CASE ;
End if;
END PROCESS WRITE_R ;

MUTIPLEX: PROCESS(d_bus, d_buf, state, tx_count, pr, clock)
BEGIN
CASE state IS
WHEN "000100" => d_bus <= d_buf ;
WHEN "001000" =>
IF tx_count = '1' AND pr /= "0000" THEN
r_w <= '1' ;
d_bus <= integer_vec(count) ;
ELSIF pr /= "0000" THEN
d_bus <= "0000000011111111" ;
r_w <= '1' ;
ELSE
d_bus <= "ZZZZZZZZZZZZZZZZ" ;
r_w <= '0' ;
END IF ;

WHEN OTHERS =>
d_bus <= "ZZZZZZZZZZZZZZZZ" ;
END CASE ;
END PROCESS MUTIPLEX ;

END all_route;
```

REFERENCES

[1] Sharad Jaiswal, Lev Zakrevski, Mehmet Mustafa, Mark Karpovsky. "Unicast Wormhole Message Routing in Irregular Computer Networks". Proc. of Int. Conf. on Parallel and Distributed Processing Techniques and Applications, 1998, pp. 2279-2285.

[2] R. Liebeskind-Hadas and et al "Tree-Base Multicast Routing in the Mesh with No Virtual Channels," Proc. of the First Merged Int. Parallel Processing Symp. and Symp. on Parallel and Distributed Processing, pp.244-249, 1998.

[3] L. Zakrevski, S. Jaiswal, L. Levitin and M. Karpovsky "A New Method for Deadlock Elimination in Computer Networks with Irregular Topologies," Proc. of the IASTED Conf. PDCS-99, 1999.

[4] J.Duato, and et al Interconnection Networks: An Engineering Approach, Los Alamitos, IEEE CS Press, 1997.

[5] L. Ni, M. and P. McKinley, K. "A Survey of Wormhole Routing Techniques in Directed Networks," Computer, vol. 26, pp. 62-76, 1993.

[6] M. Schroeder and et al. "Autonet: A High-Speed self-configuring Local Area Network Using Point-to-point Links," (Technical Report 59, DEC SRC, April 1990).

[7] W. Dally and C. Seitz, L. "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," IEEE Trans. on Comput. Vol. 36, pp. 547- 553, 1987.

[8] N. Boden and e. al. "Myrinet: A Gigabit per second Local Area Network," IEEE Micro, pp. 29-35, 1995.

[9] F. Harary "Graph Theory", Addison-Wesley, Reading, Mass, 1969.

[10] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, B.Smith, "The Tera Computer System," Technical report, Tera Computing Company, 1991.

[11] M. Bae and B. Bose "Resource Placement in Torus-Based Network," IEEE Trans. on Computers, vol.46, pp.1083-1092, October 1997.

[12] P.H. Bardell, W.H. McAnney, and J.Savir, Built-in Test for VLSI: Pseudorandom Techniques, John Wiley, New York, 1987.

[13] C. Cunningham and D. Avresky "Fault-Tolerant Adaptive Routing for Two-Dimensional Meshes," Proc. of First Int. Symp. on High Performance Computing Architecture, Raleigh, North Carolina, USA, January 1995.

[14] *Cadence Open Book Documentation*, Cadence Design Systems, 2000.