2004

DESIGN ENHANCEMENT AND INTEGRATION OF A PROCESSOR-MEMORY INTERCONNECT NETWORK INTO A SINGLE-CHIP MULTIPROCESSOR ARCHITECTURE

Kanchan P. Bhide
University of Kentucky, kpbhid2@uky.edu

ABSTRACT OF THESIS

DESIGN ENHANCEMENT AND INTEGRATION OF A PROCESSOR-MEMORY INTERCONNECT NETWORK INTO A SINGLE-CHIP MULTIPROCESSOR ARCHITECTURE

This thesis involves modeling, design, Hardware Description Language (HDL) design capture, synthesis, implementation and HDL virtual prototype simulation validation of an interconnect network for a Hybrid Data/Command Driven Computer Architecture (HDCA) system. The HDCA is a single-chip shared memory multiprocessor architecture system. Various candidate processor-memory interconnect topologies that may meet the requirements of the HDCA system are studied and evaluated related to utilization within the HDCA system. It is determined that the Crossbar network topology best meets the HDCA system requirements and it is therefore used as the processor-memory interconnect network of the HDCA system. The design capture, synthesis, implementation and HDL simulation is done in VHDL using XILINX ISE 6.2.3i and ModelSim 5.7g CAD softwares. The design is validated by individually testing against some possible test cases and then integrated into the HDCA system and validated against two different applications. The inclusion of crossbar switch in the HDCA architecture involved major modifications to the HDCA system and some minor changes in the design of the switch. Virtual Prototype testing of the HDCA executing applications when utilizing crossbar interconnect revealed proper functioning of the interconnect and HDCA. Inclusion of the interconnect into the HDCA now allows it to implement "dynamic node level reconfigurability" and "multiple forking" functionality.

KEYWORDS: Interconnection Networks, Hybrid Architectures, Multiprocessor System, System on Chip, Crossbar Network

Kanchan P. Bhide

12/17/2004

DESIGN ENHANCEMENT AND INTEGRATION OF A PROCESS – MEMORY INTERCONNECT NETWORK INTO A SINGLE – CHIP MULTIPROCESSOR ARCHITECTURE

By

Kanchan P.Bhide

Dr. J. Robert Heath
Director of Thesis

Dr.Yu Ming Zhang
Director of Graduate Studies

12/17/2004

RULES FOR THE USE OF THESES

Unpublished theses submitted for the Master's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the theses in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this thesis for use by its patrons is expected to secure the signature or each user.

Name	Date

THESIS

Kanchan P.Bhide

The Graduate School
University of Kentucky
2004

DESIGN ENHANCEMENT AND INTEGRATION OF A PROCESSOR-MEMORY INTERCONNECT NETWORK INTO A SINGLE-CHIP MULTIPROCESSOR ARCHITECTURE

THESIS

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering in the College of Engineering at the University of Kentucky

By

Kanchan P.Bhide

Lexington, Kentucky

Director: Dr. J. Robert Heath, Associate Professor of Electrical and Computer Engineering

Lexington, Kentucky,

2004

MASTER'S THESIS RELEASE

I authorize the University of Kentucky Libraries to reproduce this thesis in whole or in part for the purposes of research.

Signed: Kanchan P. Bhide

Date: 12/17/2004

ACKNOWLEDGEMENTS

I would like to thank my advisor and guide Dr. Heath for his constructive advice and support which helped me complete my thesis in a timely manner.

I would also like to thank my committee members Dr. Dietz and Dr. Dieter for their valuable support. Each individual provided insights that guided and challenged my thinking, substantially improving the finished product.

In addition to the technical and instrumental assistance above, I received equally important assistance from my family and friends.

TABLE OF CONTENTS

Acknowledgements iii
List of Tables vi
List of Figures vii
Chapter One: Introduction
1.1 Background and Positioning of Research 1
1.2 Goals and Objectives 2

Chapter Two: Types of Interconnect Systems
2.1 Static Interconnection Networks 7
2.1.1 Completely Connected Networks (CCNs) 7
2.1.2 Limited Connection Networks (LCNs) 8
2.1.2.1 Linear Arrays 9
2.2 Dynamic Interconnection Networks 14
2.2.1 Bus-based Dynamic Interconnection Networks 14
2.2.2 Switch – based Interconnect Networks 16
2.2.2.3 Multi – stage Networks (MINs 18

Chapter Three: Multistage Interconnect Network Complexity
3.1 Crossbar Topology 22
3.2 Benes Network 23
3.3 Clos Network 23
3.4 Complexity Comparison 26

Chapter Four: Design of the Crossbar Interconnect Network
4.1 Organization of Shared Memory 33
4.2 Basic Design of Crossbar Switch 33

Chapter Five: Implementation of Variable Priority Interconnection and Virtual Prototype Validation of Correct Independent Operation and Operation as the Processor-Memory Interconnect of the HDCA
5.1 VHDL Design Capture 41
5.1.1 Modifications to Behavioral Approach 41
5.1.2 Implementation of 4 x 4 Crossbar Interconnect 42
5.1.3 Functional Testing of a 4 x 4 Crossbar Interconnect Network 43
5.1.4 Component Level Description and Testing 51
5.1.5 Validation of crossbar switch via HDCA system 59
5.1.6 Application 1 Described with Acyclic Process Flow Graph 66
5.1.7 Application 2 Described with Cyclic Flow Graph 91
5.1.8: Latency and Starvation Issues 115

Chapter Six: Dynamic Node Level Reconfigurability and Multiple Forking Capability
6.1 Concept of Node Level Reconfigurability and Changes to HDCA 117
6.2 Multiple Forking Capability of the HDCA System: 125
6.3 Application Describing Multiple Forking in HDCA System: 126

Chapter Seven: FPGA Resources Utilized in HDCA Virtual Prototype Development and Testing Environment

Chapter Eight: Conclusion

Appendices
Appendix A1:Post Place and Route VHDL Code For Functional Model of the Interconnect Network 138
Appendix A2:Post Place and Route VHDL Code For Acyclic Applications 142
Appendix B:Application 1: Acyclic Process Flow Graph Model 286

References 296

Vita 298

LIST OF TABLES

TABLE 3.1 – Table for comparison of complexity ……27
TABLE 5.1 – Shared Memory Address Space and Contents ……43
TABLE 5.2 – Parameters Depiction in Cases Described ……...44
TABLE 5.3 – Shared Memory Address Space and Contents……..51

LIST OF FIGURES

Figure 1.1, High Level Schematic of the HDCA system 4
Figure 1.2, CE Controller Block Diagram 5
Figure 2.1, A Topology Based Taxonomy for Interconnection Networks 6
Figure 2.2, Static and Dynamic Interconnect Networks [10] 7
Figure 2.3, Completely Connected Network 8
Figure 2.4, Linear Array Network 9
Figure 2.5, Ring Network 9
Figure 2.6, 2-D Mesh Network 10
Figure 2.7, 2-D Torus Network 10
Figure 2.8, 3x3x2 Mesh Network 11
Figure 2.10, Cube-Interconnect Networks with Different Dimensions. 12
Figure 2.11, Cube Connected Cycles. 13
Figure 2.12, Star Connected Network. 14
Figure 2.13, Single Bus System 15
Figure 2.14, Crossbar Network System. 16
Figure 2.15, Different Settings of the 2 x 2 SE 17
Figure 2.16, 8 x 8 Omega Network 19
Figure: 2.18, Re- arrangement of Connection 110 → 100 20
Figure 2.19, Clos Three-Stage Network in Block Form 21
Figure 3.1, 8 x 8 Crossbar Network 22
Figure 3.2, Benes Network[6] 24
Figure 3.3, Clos Network[6] 25
Figure 3.4, Complexity Chart for N<= 16 [6] 28
Figure 3.5, Complexity Chart for N>=16 [6] 28
Figure 3.6, Multiprocessor Shared Memory Organization 30
Figure 3.7, Shared Memory Organization 31
Figure 3.8, Organization of Each Memory Block 31
Figure 4.3, Decode Logic D[i] 35
Figure 4.4, Priority Logic Block 35
Figure 4.5, Flow Chart for the Priority Logic Block in Figure.4.4 36
Figure 4.1a, PI[i] and D[i][j] Bus Structure 37
Figure 4.1b, IM[j] Bus Structure 38
Figure 4.6, Plot Showing Gate Count vs Size of Crossbar Interconnect Network 38
Figure 4.2a, Detailed Block Diagram Interconnection Network for N x M Network 39
Figure 4.2b, Detailed Block Diagram of the Interconnect Network 40
Figure 5.1, Simulation Tracer 1 48
Figure 5.2, Simulation Tracer 2 49
Figure 5.3, Simulation Tracer 3 50
Figure 5.4, Simulation Tracer 4 55
Figure 5.5, Simulation Tracer 5 56
Figure 5.6, Simulation Tracer 6 57
Figure 5.7, Brief Overview of Priority Logic for the Interconnection Switch. 58
Figure 5.9, Memory/Register Architecture with Added Features 61
Figure 5.8, Changes Made to PE Controller- Additional Mux M5 62

Figure 5.11, An Example of a Process Flow Graph 64
Figure 5.12, Process Flow Graph for Application 1 66
Figure 5.13, Command Tokens for Both Copies of P1 to CE0 Issued by PRT Mapper 69
Figure 5.14, First Instruction and Input of First Five Values. 70
Figure 5.15, Input of Last 5 Values for Process P1 of copy 1 71
Figure 5.16, Two Command Tokens Issued to PRT Mapper of Copy 1 72
Figure 5.17, Command Tokens Issued to CE0 and CE1 by PRT Mapper of Copy 1 73
Figure 5.18, Instructions for Process P1 of Copy 2 and for Process P3 of Copy 1 74
Figure 5.19, Two Command Tokens Issued to PRT Mapper for Copy 2 75
Figure 5.20, Two Command Tokens Issued to CEs by PRT Mapper for Copy 2 76
Figure 5.21, Instruction Issued by CE0 of Process P2 for Copy 1 77
Figure 5.22, Instruction by CE1 of P3 Copy 2 and Commend Token from PRT to CE2 ... 78
Figure 5.23, Division Op. for P5 with Results, Command Token to PRT Mapper Copy 1 80
Figure 5.24, Command Tokens for P4 to PRT Mapper, from PRT to CE4 for Copy 1 81
Figure 5.25, Multiplication Operation by CE4, Token Issued to PRT Mapper Copy 1 82
Figure 5.26, Command Token for P5 Issued to PRT Mapper, from PRT to CE2 Copy 2.. 83
Figure: 5.27 Div Op.for P5 with Results, Command Token to PRT Mapper for Copy 2... 84
Figure 5.28, Join Instruction for Process P6 for Copy 1 86
Figure 5.29, Process P7 Instruction and Final Output of the Result of P6 for Copy 1 87
Figure 5.30, Multiplication Process Result, Command Token to PRT Mapper Copy 2 88
Figure 5.31, Join Process P6 Instructions for Copy 1 89
Figure 5.32, Process P7 with Final Value of the Result of Process P6 for Copy 1 90
Figure 5.33, Application for Swapping of two Values 91
Figure 5.34, Instruction for Process P1 and Input of First two Values 94
Figure 5.35, Process P1 Inputs of Last 3 Values 95
Figure 5.36, Instructions for Processes P2 and P4 with Results 96
Figure 5.37, Process P3: First Comparison 97
Figure 5.38, Process P5: First Execution 98
Figure 5.39, Process P2: Second Execution 100
Figure 5.40, Process P4: Second Execution 101
Figure 5.41, Process P3: Second Execution 102
Figure 5.42, Process P5: Second Execution 103
Figure 5.43, Process P2: Third Execution 104
Figure 5.44, Process P4: Third Execution 105
Figure 5.45, Process P3: Third Execution 107
Figure 5.46, Process P5: Third Execution 108
Figure 5.47, Process P2: Fourth Execution 109
Figure 5.48, Process P4: Fourth Execution 110
Figure 5.49, Process P3: Fourth Execution, Exit of Loop, Token Issued to PRT Mapper 111
Figure 5.50, Process P5: Fourth Execution, Exit of Loop, Token Issued to PRT Mapper 113
Figure 5.51, Process P6 Join Operation, Final Results, Values Swapped 114
Figure 6.1, Load Threshold Token Fed Into HDCA Setting the Value of Threshold Flag117
Figure 6.2, Input of two Load Threshold and Eight Command Tokens in the System 119
Figure 6.3, Process P1 for First Four Command Tokens 120
Figure 6.4, Forking of Tokens and Queue Depth of CE1 Reaching Threshold Value 121
Figure 6.5, Threshold Flag Set for CE0 and CE1, Stand by CE Triggered 123

Figure 6.6, Threshold Flag Set 2nd Time for CE0 and CE1, Stand by CE Triggered 124
Figure 6.6, Multiple Forking Concept Used in the HDCA System 125
Figure 6.7, Application Describing Multiple Forking in HDCA 126
Figure 6.8, Command Token Input into the System 128
Figure 6.9, Instruction for Process P1 129
Figure 6.10, Instruction for P2 with Result and Command Token for P3 130
Figure 6.11, Process P3 and Process P7 Execution and Results 131
Figure 6.12, Instruction for Process P4 and P5 with Results 132
Figure 6.13, Join Instructions for Process P8 with Result 133
Figure 6.14, Final Process P6 Join Operation and Result 134

Chapter One

Introduction

1.1 Background and Positioning of Research

Most parallel/distributed architectures require some type of interconnection network to link the various elements of the computer system. Anyone designing a parallel or distributed architecture computer system encounters a major obstacle; the Interconnection Network problem. A multiple processor system consists of two or more processors connected to each other in a manner that allows them to share the simultaneous execution of a given computational task. Interconnection networks should be capable of providing rapid data transfer among processors; memories etc.

There are a number of styles of communication for multiprocessor networks. These can be broadly classified according to the communication model as multiple processors (single address space/shared memory) versus multiple computers (multiple address/distributed memory). They can be classified according to the physical connection used as bus based versus network based multiple processors. The organization and performance of a multiple processor system is greatly influenced by the interconnection network used to connect them. A single shared bus network or a crossbar switch can be used as an interconnection network. While the first technique represents a simple, easy to expand topology; it is however limited in performance since it does not allow more than one processor/memory transfer at any given time. The crossbar provides full processor/memory distinct connections however it can be expensive. Multistage interconnection networks sometimes strike a balance between the limitation of a single shared bus network system and expense of crossbar switch system. The cost of multistage interconnection network can be considerably less as compared to the crossbar, especially for large numbers of processor/memories. Crossbar interconnect,though are cost and complexity competitive with other types of interconnects when within a certain size range, as will be later addressed. Another way to connect the multiple processors to multiple memories is to use multiple bus system. It can also be suggested as a compromise between shared bus system and crossbar switch.

This thesis is primarily a study of various kinds of interconnect networks which may meet the requirements of the Hybrid Data/Command Driven Architecture (HDCA) multiprocessor system [16,17,7] shown in Figure 1.1. It then involves enhancement of a design developed in [6]. The design is described in Vhsic Hardware Description Language (VHDL) for an interconnect network suitable for integration into the HDCA system.

The HDCA is a multiprocessor shared memory architecture. It is a hybrid data flow machine since it uses data flow concepts to migrate data from one process to another yet uses a program counter in the actual execution of processes on processors. It utilizes advantages of both Von-Neumann and data flow type architectures [16,17]. The shared memory is organized as a single memory divided into blocks as shown in the Figure 1.1 and is explained in detail in Chapter 3. The interconnect network to be designed should be able to connect requesting processors on one side of the interconnect network to the memory blocks on the other side of the interconnect network as shown in Figure 3.7. The efficiency of the interconnect network increases with number of parallel connections between the processors and memory blocks.

1.2 Goals and Objectives

The goal of this work is to analyze different types of interconnect networks applicable to the HDCA system. Their connection, routing mechanism, complexity are described and evaluated. This work also includes design, VHDL description synthesis and implementation of interconnect network based on a crossbar topology. Analysis will show this topology to be the one that best meets the HDCA system requirements.

The crossbar switch can provide simultaneous connections among all its inputs and all its outputs. The crossbar contains a switching element (SE) at the intersection of any two lines extended horizontally or vertically inside the switch. Circuit switching networks can be classified by the connection of the network and the relation of the network inputs and outputs [1]. The most basic distinction concerns whether inputs and outputs are differentiated or whether all ports are treated equally. Single sided networks have only one set of connections or ports, all of them are treated equally. They can be designed to meet the requirements of inter-processor communication and are unique to distributed memory systems. In distributed memory systems processors do not have to share common memory,

each processor communicates with its own local memory, hence using single sided crossbar network communication between any two processors is possible. Double sided networks connect two distinct set of ports, usually referred to as inputs and outputs. Ports in one set generally may be connected only to the ports of the other set.

The HDCA system under consideration is shared memory architecture. It will be seen the double sided crossbar network is the most suitable to be used as the interconnect network. The design will need some kind of priority logic, which prioritizes conflicting requests for memory accesses by the processors. In the HDCA system the memory is divided into blocks, each block containing memory locations with different address ranges. The interconnect design will be a combination of a double sided crossbar interconnect network, priority logic and a shared memory organization. A single bus system network is being used as a means to communicate between the CEs and data memory in a "first phase prototype" of the HDCA system [5]. The interconnect design to be developed and described in this thesis was developed specific to the HDCA system but it may possibly be used in other multiprocessor systems.

A survey of different types of interconnect networks is described in Chapter 2. Complexity of dynamic interconnection networks is discussed in chapter 3. The detailed crossbar interconnect network design is described in Chapter 4. Chapter 5 includes individual simulation testing set up and results, validation of the crossbar is done using the HDCA system, modifications to the HDCA system for proper functioning of the entire system are described. Additional functionality incorporated into the HDCA system, Dynamic Node Level Reconfigurability and Multiple Forking capability of the architecture is explained in detail and exhibited with the help of application in Chapter 6. Virtual prototype development and the testing environment is discussed in Chapter 7. Conclusions and recommendations are being included in Chapter 8.

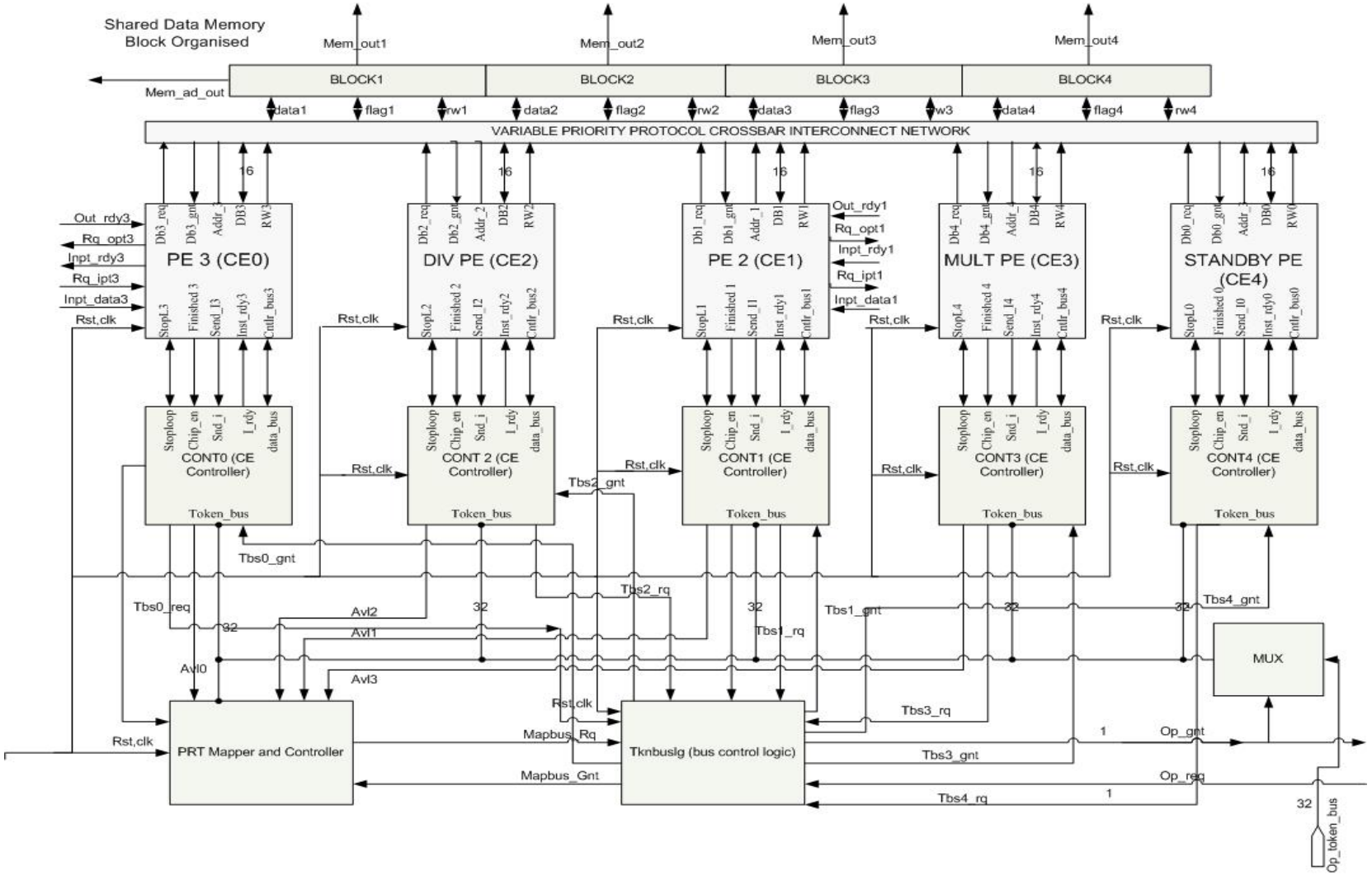

Figure 1.1, High Level Schematic of the HDCA system

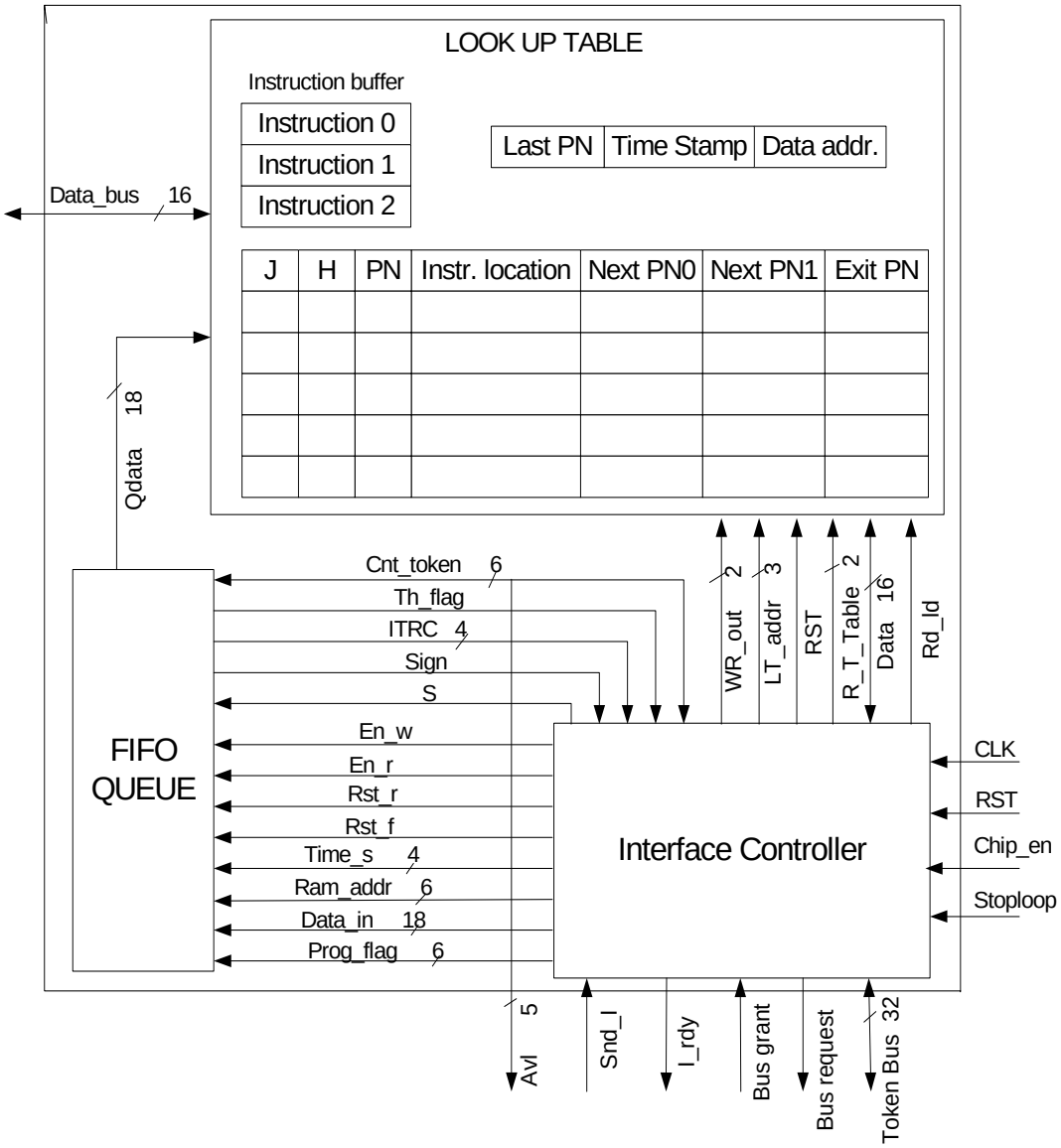

Figure 1.2, CE Controller Block Diagram

Chapter Two

Types of Interconnect Systems

Interconnect networks can be classified as either static or dynamic [4]. Static networks can be further classified according to their interconnection pattern as One-dimension (1D), Two Dimensional (2D) and Hypercube (HC's).Dynamic networks can be further classified according to the scheme of interconnection as bus-based versus switch-based. Bus-based networks are classified as single bus or multiple bus. On the other hand switch-based dynamic networks are classified according to the structure of the interconnect network as Single-Stage (SS), Multistage (MS), or crossbar networks.

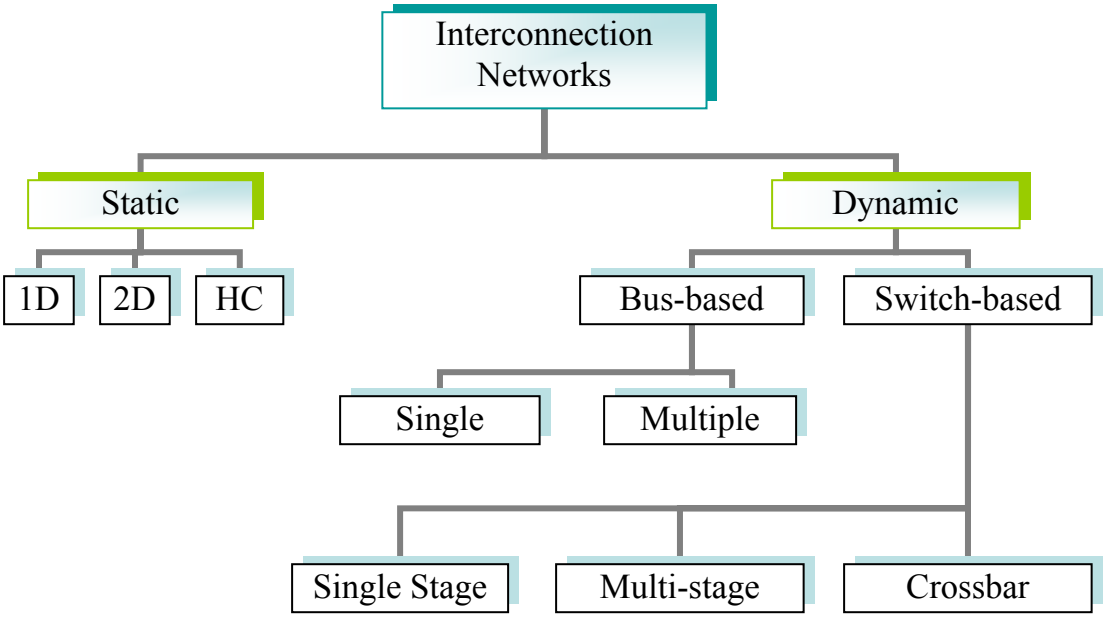

Figure 2.1, A Topology Based Taxonomy for Interconnection Networks

Static networks form all connections when the machine is designed rather than when the connection is needed. In a static network, messages must be routed along established link. This means a single message must 'hop' through intermediate processors on its way to its destination. The nodes are the computers and the routing is done by the computers. Dynamic networks establish a connection between two or more nodes 'on the fly' as messages are routed along the links. The nodes are switching elements and the routing is done by the network [8] [9].

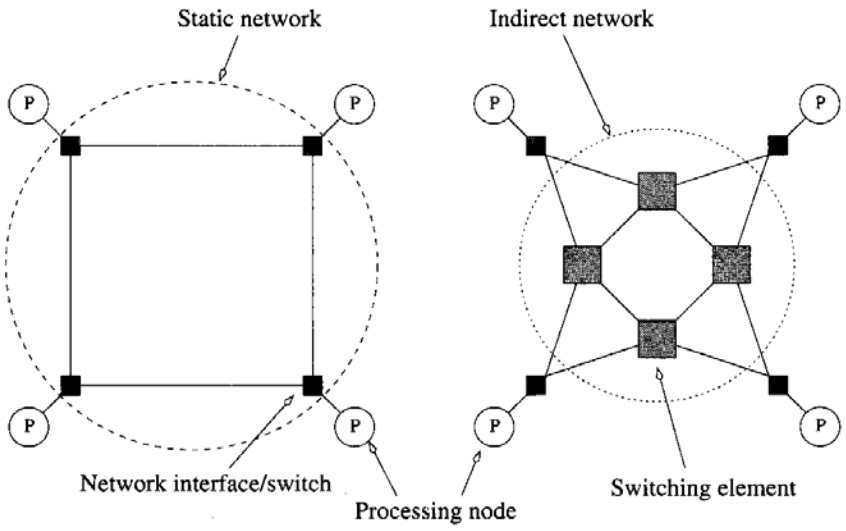

Figure 2.2, Static and Dynamic Interconnect Networks [10]

2.1 Static Interconnection Networks

The features are having fixed paths, unidirectional or bi-directional, between processors. Two types of static networks can be identified. They are 'Completely Connected Networks' (CCNs) and 'Limited Connection Networks' (LCNs) [4].

2.1.1 Completely Connected Networks (CCNs)

In a completely connected network each node is connected to all other nodes in the network. An example having nodes N = 6 is shown in Figure. 2.3. A total of 15 links are required in order to satisfy the complete interconnectivity of the network.

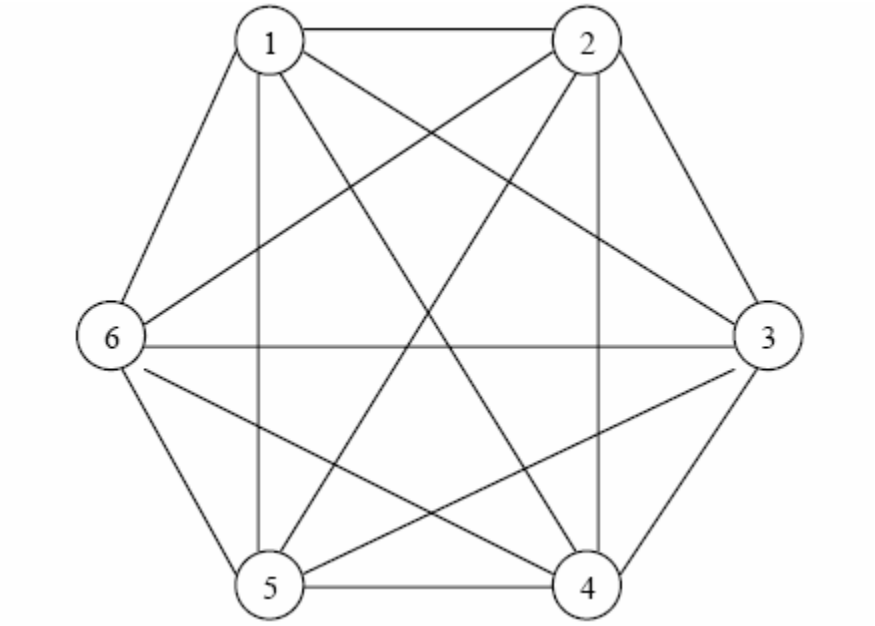

Figure 2.3, Completely Connected Network

These networks guarantee fast delivery of messages from any source node to any destination node, only one path has to be traversed. Completely connected networks are however expensive in terms of number of links needed for construction. This disadvantage is more severe for higher values of N.

The number of links in a completely connected network is given by N (N-1)/2, i.e. O (N^2). The delay complexity of CCNs, measured in terms of the number of links traversed as messages are routed from any source to any destination is constant, O(1).

2.1.2 Limited Connection Networks (LCNs)

As the name suggests these networks do not provide a direct link from every node to every other node in the network. The length of the path for the message to traverse from source node to destination is expected to be longer compared to the paths in CCNs. Two conditions have to be taken into consideration for the LCNs, the need for a pattern of interconnection among nodes and the need for a mechanism for routing messages around the network until they reach their destinations.

2.1.2.1 Linear Arrays and Ring (Loop) Networks

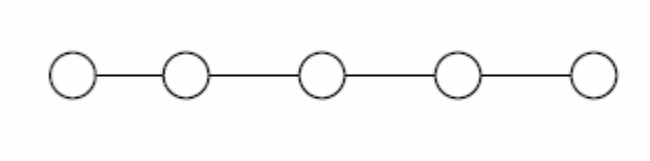

Figure 2.4, Linear Array Network

Each node is connected to its two immediate neighboring nodes, except the ones at each end. If node *a* needs to communicate with node *b*, *b*>*a*, the message traverses through nodes *a+1, a+2… b-a*. Similarly if *b*<*a*, the message has to traverse nodes *a-1, a-2… a-b*. The worst possible case is when node 1 has to send a message to node 'N'. The message has to traverse a total of N-1 nodes before it reaches the destination node. Hence though these systems are easy to design and have a simple architecture. They tend to be slow. This is particularly shows when 'N' is large.

The network complexity of a linear array is O (N) and time complexity is O (N). If the two nodes at the extreme ends of a linear array network are connected, then the resultant network has a ring (loop) architecture.

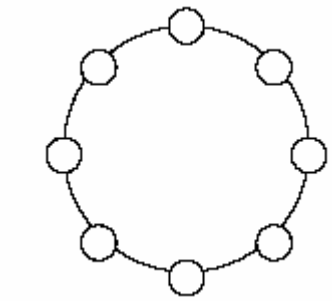

Figure 2.5, Ring Network

2.1.2.2 Mesh-connected Networks

An *n*-dimensional mesh can be defined as an interconnection structures that has $K_0 \times K_1 \times \ldots \times K_{n-1}$ nodes where n is the number of dimensions of the network and K_i is the radix of dimension *i*.

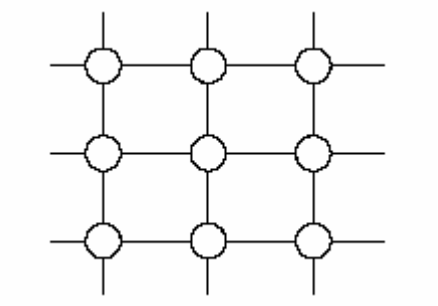

Figure 2.6, 2-D Mesh Network

The linear array as described above is a 1-D mesh network. The mesh architecture with wraparound connections forms a 'torus'.

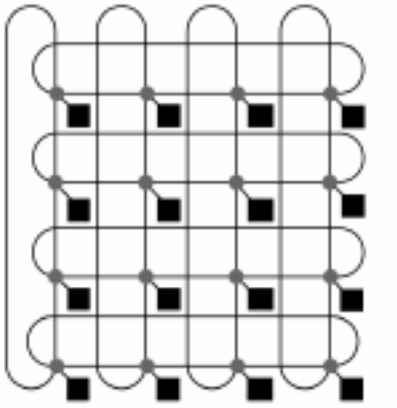

Figure 2.7, 2-D Torus Network

A number of routing mechanisms have been used to route messages around meshes. One such routing mechanism is known as the 'dimension-ordering' routing. A message is routed in one given dimension at a time, arriving at the proper coordinate in each dimension before proceeding to the next dimension. Consider a 3-D mesh as shown in the Figure below. It is a 3x3x2 mesh network. Each node is represented by its position *(i, j, k)*, messages are first sent along the *'i'* dimension then along the *'j'* dimension and finally along the *'k'* dimension. A maximum two turns are allowed and these turns are from *'i'* to *'j'* and then from *'j'* to *'k'*. Figure 2.8 shows the route of a message sent from

node S at (0, 0, 0) to node D at position (2, 1, 2). For a mesh interconnection network with 'N' nodes, the longest distance traveled between any two arbitrary nodes is O ($\sqrt{N}$)

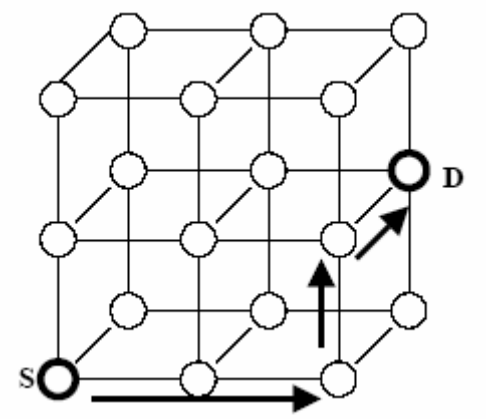

Figure 2.8, 3x3x2 Mesh Network

Multiprocessors with mesh interconnection networks are able to support many scientific computations very efficiently. Another advantage of mesh interconnection networks is that they are scalable.

2.1.2.3 Tree Network

A binary tree network shown in the Figure 2.9 is an example of a tree network.

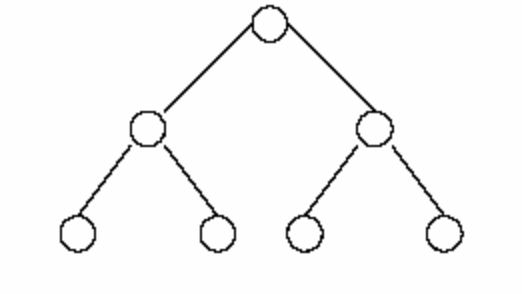

Figure 2.9, A Tree network

If a node at level *'i'*, assuming that the root node is at level 0; needs to communicate with a node at level *'j', (i>j)* and the destination node belongs to the same root's child sub tree, then it sends its message up the tree traveling node at levels *i-1, i-2,…j+1* until it reaches the destination node.

2.1.2.4 Cube- connected Networks

Cube connected networks have their interconnection network patterned after the n-cube structure. An n-cube, also called the Hypercube of order n, is defined as a undirected graph having 2^n vertices (0 to $2^n - 1$) such that there is an edge between a given pair of vertices if and only if, the binary representation of their addresses differs by one and only one bit.

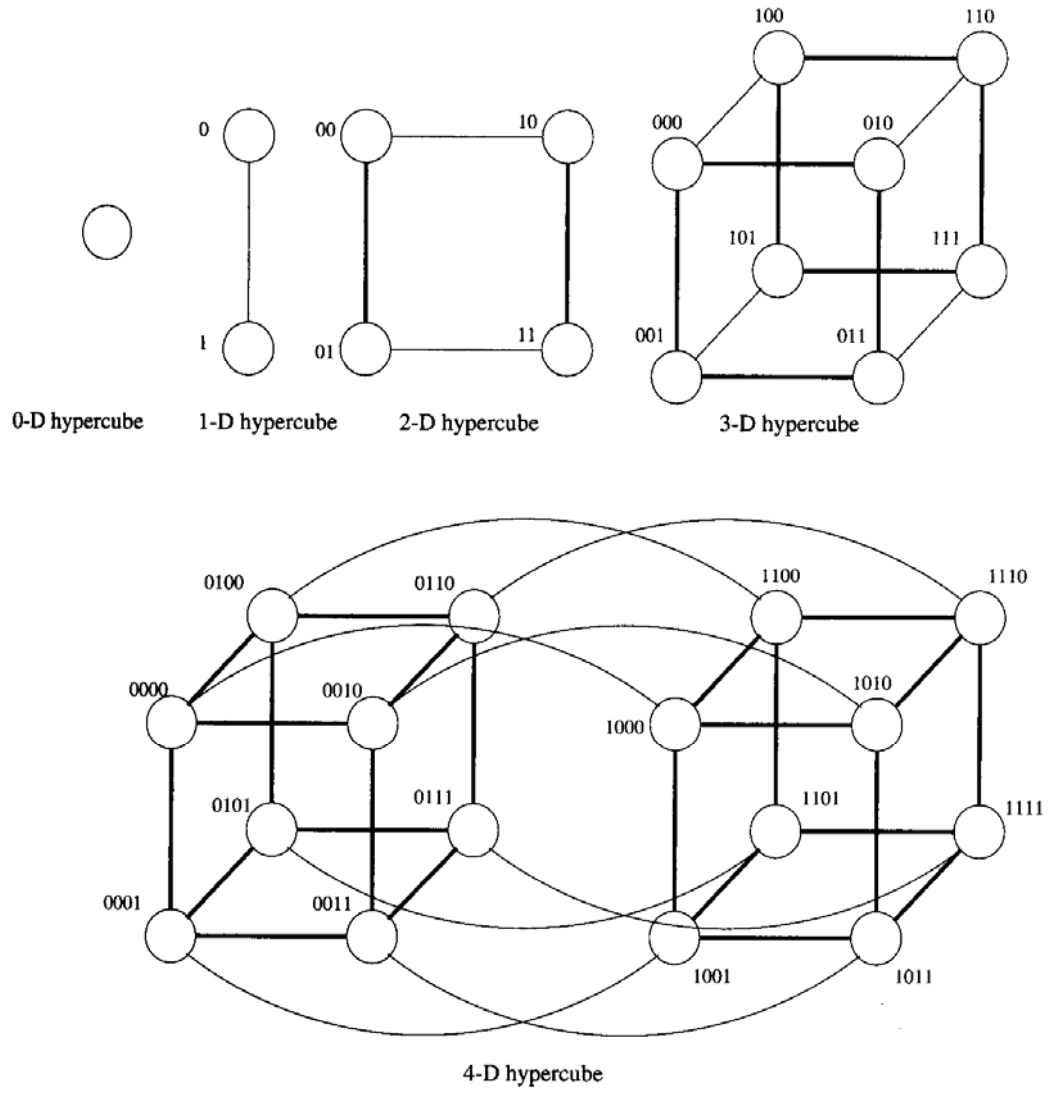

Figure 2.10, Cube-Interconnect Networks with Different Dimensions.

In a cube-based system processing elements are positioned at the vertices of the graph. Edges of the graph represent point-to-point communication links between

processors. Each processor in a 4-cube network is connected to 4 other processors. If the source of message is at *'i'* and the destination is at *'j'* then route of message can be found by xoring the binary address representation of *'i'* and *'j'*. For instance, if a message is sent from source 'S' node 0101 to destination 'D' node 1011, then the xoring operation results in 1110. This means that the message has to traverse only along 2, 3 and 4 (from left to right) in order to arrive at the destination. The order of message travel is not important. Once a message travels in any order along the three dimensions it reaches the destination node.

The hypercube is also known as logarithmic architecture. This is due to the fact that the maximum number of links a message has to traverse in order to reach its destination in an *n*-cube containing $N= 2^n$ nodes in $\log_2 N = n$ links. Another feature of the hypercube networks is the recursive nature for their construction. An n-cube can be constructed from two sub cubes each having an *(n-1)* degree by connecting nodes of similar addresses in both sub cubes. As shown in Figure 2.9 the 4-cube network is constructed from two 3- cube networks. However it can be seen that the hypercube networks are not easily scalable.

One of the variations from the basic hypercube interconnection is the cube-connected cycle architecture. In this architecture, 2n+r nodes are connected in an n-cube fashion such that groups of r nodes each form cycles at the vertices of the cube.

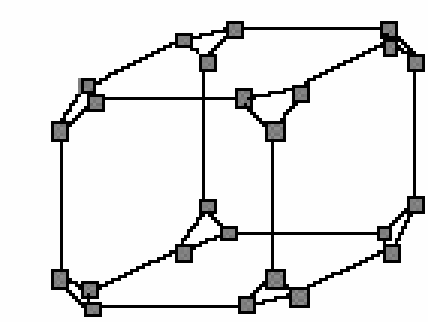

Figure 2.11, Cube Connected Cycles.

Figure 2.11 shows a 3-cube connected cycle network with r=3. It has three nodes forming a loop at each vertex of the 3-cube.

2.1.2.5 Star Connected Network

In a star topology [6](see Figure 2.12) there is one central node processor to which all other nodes are connected, each node has one connection whereas center node has N-1 connections. The routing mechanism is trivial. If the message is routed from one of the nodes to center node the path is just the edge connecting them. If a message is routed from a source node to a destination node other than the destination node, then the message is routed from a source node to a center node and from that node to a destination node. Star networks are not feasible for large networks since the central node becomes a bottleneck.

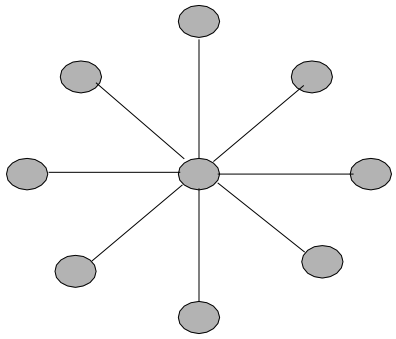

Figure 2.12, Star Connected Network.

2.2 Dynamic Interconnection Networks

These networks include bus-based systems (single and multiple) and switch – based systems (single-stage, multi-stage and the crossbar) [4].

2.2.1 Bus-based Dynamic Interconnection Networks

2.2.1.1 Single bus Systems

They are the simplest and an efficient solution when the cost and a moderate number of processors are involved. [6]

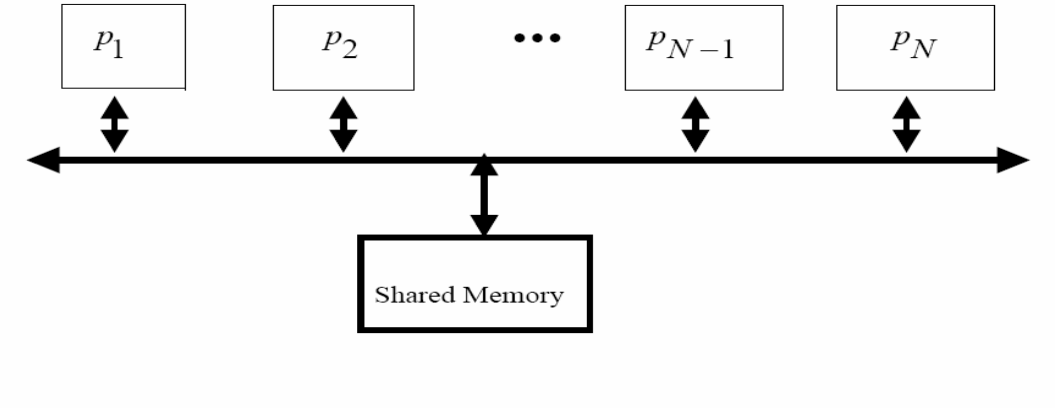

Figure 2.13, Single Bus System

All processors communicate with a single shared memory. Typical size of such a system is between 2 to 32 processors. However single bus systems are inherently limited by the bandwidth of the bus and also that only one single bus communication can take place at a time. The main drawback is a bottleneck to the shared memory when the number of processors becomes large and also a single point of failure hangs the entire system.

2.2.1.2 Multiple Bus Systems

The use of multiple buses to connect multiple processors is a natural extension to the single shared bus system. This system uses several parallel buses to interconnect multiple processors and multiple memory modules. A number of connection schemes is possible with this system [4].

- Multiple - bus with full bus-memory connection: All the memory modules connected to all buses.
- Multiple – bus with single bus-memory connection: Each memory module connected to a specific bus.
- Multiple – bus with partial bus-memory connection: Each memory module connected to a subset of buses.
- Multiple – bus with class- based memory connection: Memory modules are grouped into classes and each class is connected to a specific subset of buses.

Buses can be classified as ‘synchronous’ or ‘asynchronous’. For any event on the synchronous bus, the transaction time is taken into account. The transaction time is known apriori. In the case of asynchronous buses,the occurrence of an event is triggered by availability of data and the readiness of devices to initiate bus transactions.

2.2.2 Switch – based Interconnect Networks

Connections among processors and memory modules are made with the help of simple switches. Three basic topologies exist, they are Crossbar, Single –stage, and Multi-stage.

2.2.2.1 Crossbar Networks

All the processors in a crossbar network(Figure 2.14) have dedicated buses directly connected to all memory blocks. It represents the other extreme to the limited single bus network; it can provide simultaneous connections among all its inputs and all its outputs [1,4,10].

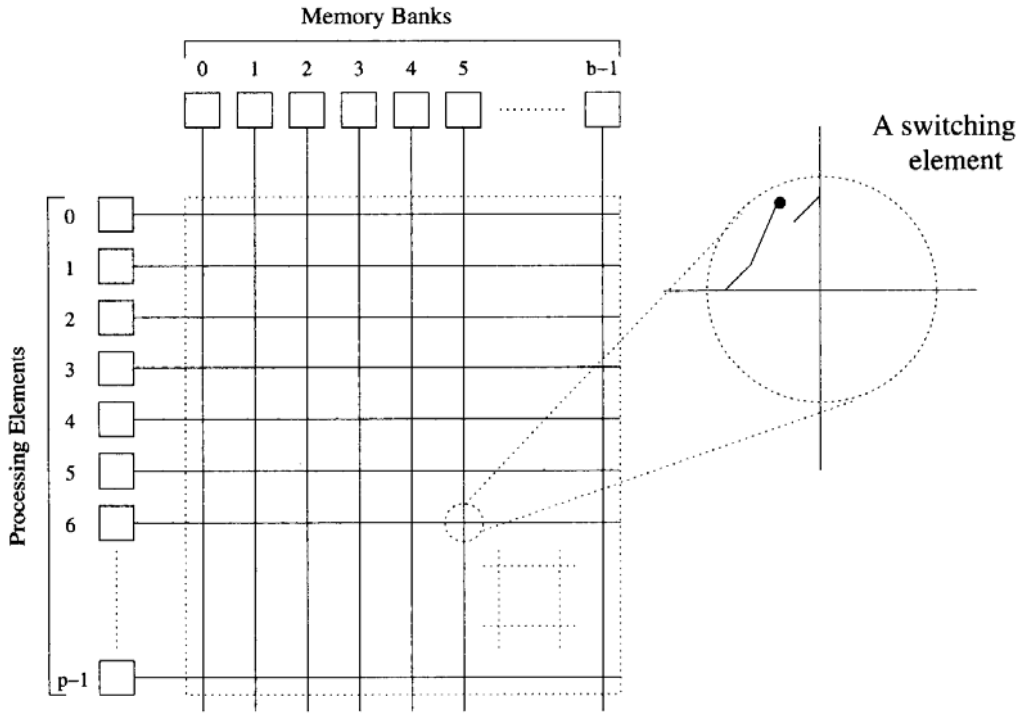

Figure 2.14, Crossbar Network System.

A crossbar contains a Switching Element (SE) at the intersection of any two lines extended horizontally or vertically inside the switch. It is a 'nonblocking' network that allows any input – output connection pattern to be executed.

The network complexity for an N x N crossbar measured in terms of number of switching points is O (N^2). The time complexity measured in terms of the amount of input to output delay is O (1). In spite of high speed, their use is normally limited to those systems containing 32 or fewer processors due to increase in complexity and hence the cost.

2.2.2.2 Single Stage Interconnection Networks

In these networks a single stage of switching elements (SEs) exists between the inputs and the outputs of the network. The simplest that can be used is the 2 x 2 switching element. There could be four possible settings that such switching element can assume.

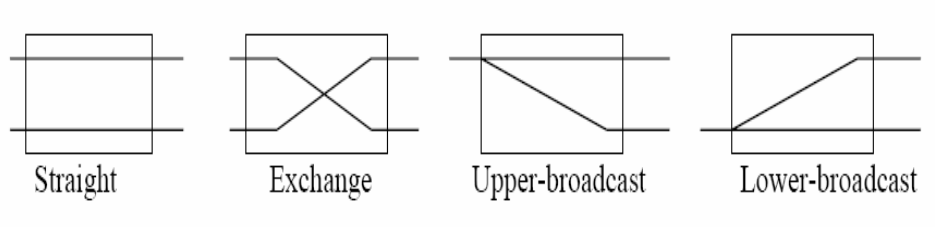

Figure 2.15, Different Settings of the 2 x 2 SE

- Straight: Upper input is transferred to the upper output and the lower input to lower output.
- Exchange: Upper input is transferred to the lower output and the lower input to upper output.
- Upper – broadcast: Upper input is broadcast to both the upper and lower outputs.
- Lower – broadcast: Lower input is broadcast to both the upper and lower outputs.

A well known connection pattern for interconnecting the inputs and the outputs of a single-stage network is the'Shuffle-Exchange'. Two operations are used; they can be defined using an 'm' bit- wise address pattern of inputs, $p_{m-1}\ p_{m-2}\ldots p_1 p_0$, as given:

$$S\ (P_{m-1}\,P_{m-2}\ldots P_1\,P_0) \;= P_{m-2}\,P_{m-3}\ldots P_1\,P_0\,P_{m-1}$$

$$E\ (P_{m-1}P_{m-2}\ldots P_1P_0) = P_{m-1}P_{m-2}\ldots P_1\,\overline{Po}$$

Perfect shuffle operation: cyclic shift 1 place left as in 101 → 011

Exchange operation: invert least significant bit as in 101 → 100 [8]

With these operations data is circulated from input to output until it reaches its destination. The network complexity of the single stage interconnection network is O (N) and the time complexity is O (N).

2.2.2.3 Multi – stage Networks (MINs)

Multi-stage Interconnection Networks (MINS) are introduced as a means to improve some limitations of the single–stage networks while keeping the cost within limits. MINs can provide a number of simultaneous paths between processors and memory modules. The routing of a message from a given source to a given destination is based on the destination address, also called 'Self–routing'. Each bit in the destination address is used to route the message through one stage in several of these networks[1,4,8]. The first MSB of the destination address is used to control the routing in the first stage, the next bit for stage two and so on [4][8].

MINs can be classified in a number of different ways; one of the criterions is 'blockage' [1,4]. Based on this criterion the MINs are classified as

- Blocking Networks: These networks possess the property that in the presence of a currently established interconnection between a pair of input to output the arrival of a new interconnection between two arbitrarily unused input and output may or may not be possible.
- Re–arrangeable Networks: These networks have the property that it is always possible to re- arrange already established connections in order to make allowance for other connections to be established simultaneously.
- Non–blocking Networks: These are networks that are characterized by the property that in the presence of a currently established connection between any pair of input/output, it is always possible to establish a

connection between any arbitrary unused pair of input/output without rearrangement of any existing connection.

Omega Network

A multi-stage interconnect network using 2 x 2 switch boxes and a perfect shuffle interconnection pattern between the stages is called an Omega Network [1,4](see Figure 2.16).

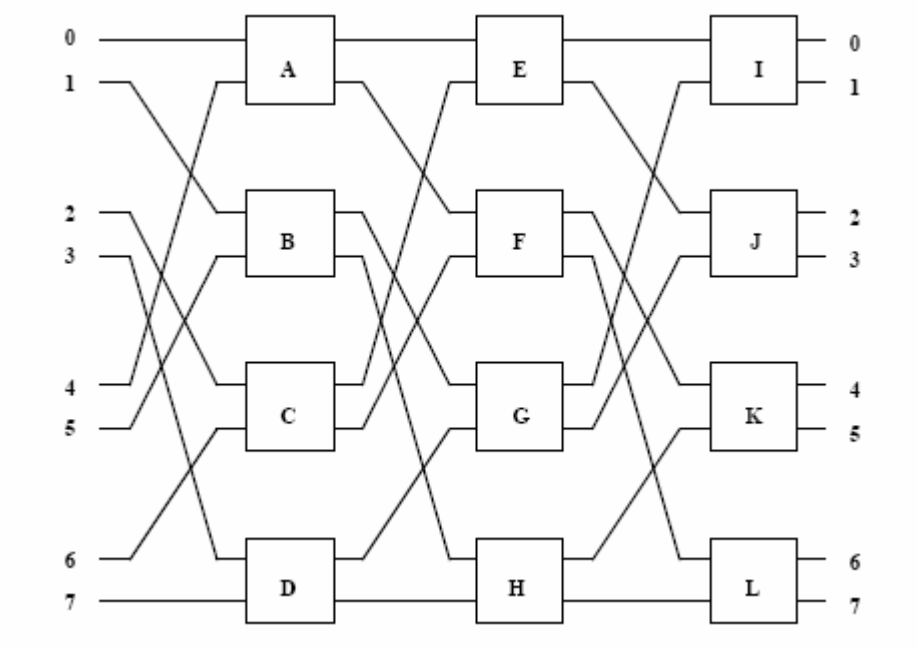

Figure 2.16, 8 x 8 Omega Network

In general an N x N Omega Network connects N inputs to N outputs where N = 2 n. There are 'n' stages of N/2 switches of size 2 x 2 and the input is a shuffle connection. In an Omega Network there is one unique path from each input to each output. This is an example of a 'blocking network'.

Benes Network

It is a well known example of a re–arrangeable network [1,4]. Figure 2.17 shows an example 8 x 8 Benes Network. Two simultaneous connections are shown established in the network.A message is to be routed from 110 → 100 and 010 → 110. The paths are shown, one as a dotted line and another as a bold dark line [4].

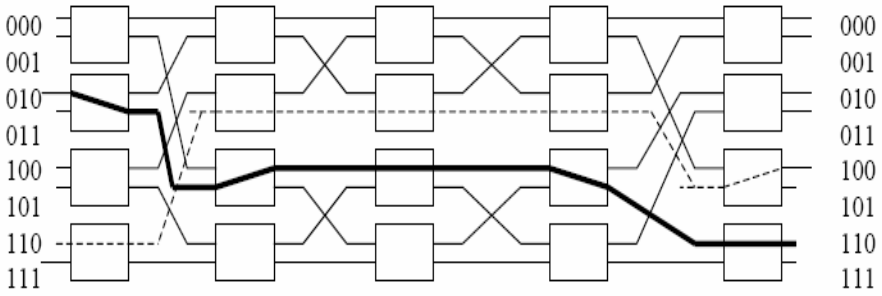

Figure: 2.17, Benes Network with Two Simultaneously Established Paths.

The re-arrangeable property of this network can be exhibited with the following example. If a message is to be routed from 101 → 001 with the previously two paths established, the path from 110→ 100 must be re routed. This re–arrangement is shown in Figure 2.18.

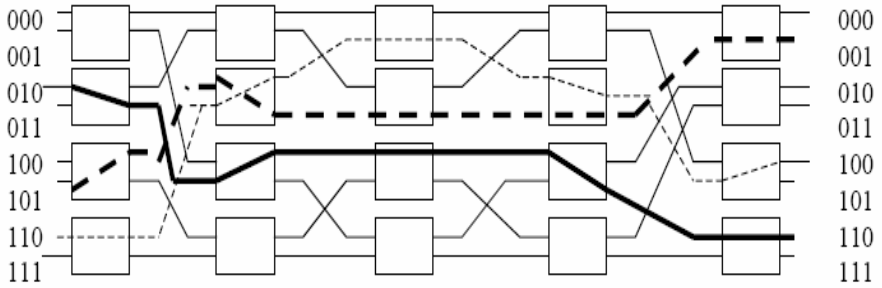

Figure: 2.18, Re- arrangement of Connection 110 → 100

Clos Network

It is a well known example of a non-blocking networks. [1,4] It comprises of $r_1(n_1 \times m)$ input crossbar switches, $m(r_1 \times r_2)$ middle crossbar switches and $r_2 m \times n_2$ output crossbar switches. Generally Clos's three stage nonblocking arrangement uses rectangular crossbars in all stages; however with an equal number of inputs and outputs, the middle switches are square. Each crossbar has one output connected to an input of each crossbar of the stage that follows, hence there always exists a possible path through each of the middle – stage switches between any input and output [1].

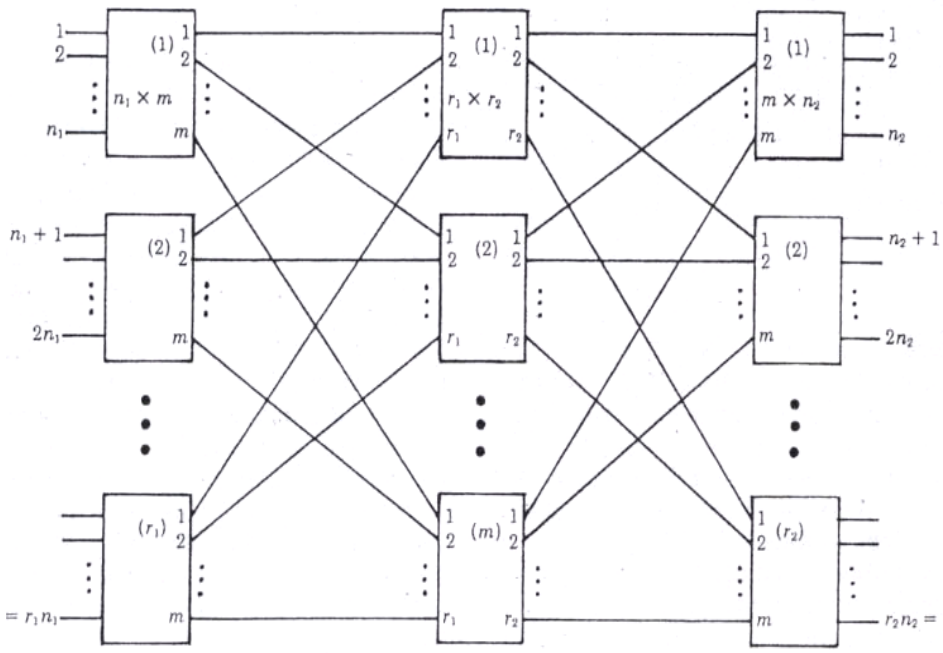

Figure 2.19, Clos Three-Stage Network in Block Form

Chapter Three

Multistage Interconnect Network Complexity

Chapter 2 overviewed the various types of interconnect network systems used in today's world. The main issue in this chapter is to choose a suitable interconnect network which can be used in the HDCA system[7]. For the HDCA system, the desired interconnect should be able to establish non-blocking, high speed connections from the requesting Computing Elements (CEs) to the shared memory. The interconnect should be able to resolve any conflicts of two or more processors wanting to access the same memory block and grant access to the memory to the CE which has the highest priority. Considering this brief overview of the requirements of an interconnect network suitable for the HDCA system, it can be concluded that the non-blocking, re-arrangeable multistage or crossbar networks (Benes, Clos and Crossbar) discussed in chapter two are suitable to use in the HDCA system.

3.1 Crossbar Topology

The Crossbar provides simultaneous connections among all its inputs and outputs. It is a non-blocking, very reliable and high–speed network. An (8x 8) example is shown in Figure 3.1.

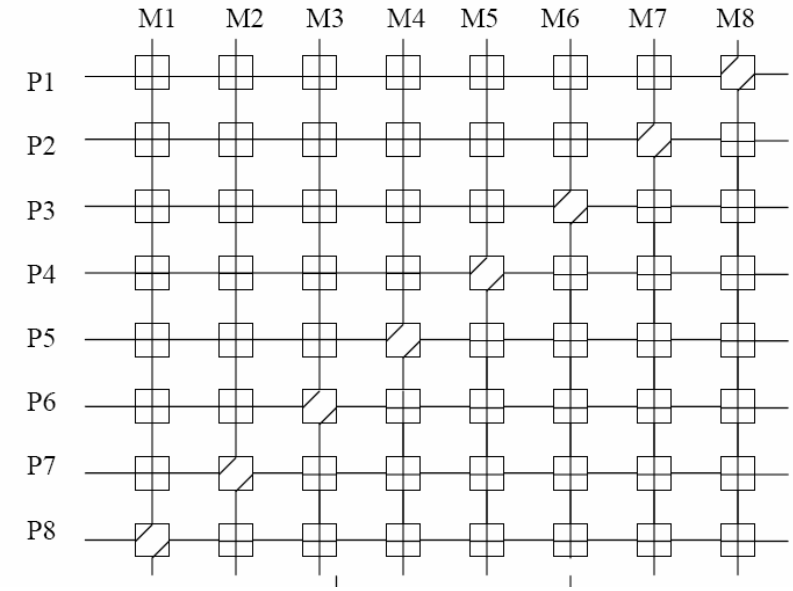

Figure 3.1, 8 x 8 Crossbar Network

A typical crossbar network with 'N' inputs(processors) and 'M' outputs(memory modules), is depicted by $X_{N, M}$. As discussed in Chapter two, the complexity of a crossbar is given by N x M. Complexity increases with the increase in number of inputs and outputs. This is the main limitation of this network. This reduces the scope of scalability of crossbars [6].

3.2 Benes Network

The Benes network(Figure 3.2) is a re-arrangeable multistage network. [1,6] For any value of N,d should be chosen so that $\log_d N$ is an integer. The number of stages for an N x N Benes network is given by (2logdN-1) and has (N/d) crossbar switches in each stage. Hence $B_{N,d}$ is implemented with [[(N/d)*(2$\log_d$N-1)] crossbar switches. The general architecture is as shown in Figure 3.2. As can be seen in the Figure,

N: Number of inputs or outputs

d: Dimension of each crossbar switch ($X_{d,d}$)

I: First stage switch = $X_{d, d}$

II: Middle stage switch = $B_{N/d,d}$

III: Last stage switch = $X_{d,d}$

The complexity of the network is given by [(N/d)* (2$\log_d$N-1)* d^2]. The network latency is a factor of (2$\log_d$N-1), since there are (2$\log_d$N-1) stages between input and output. There are different routing paths available between input and output. It is a limited scalability network; this and network latency are the main drawbacks of the Benes network. For very large networks Benes network implementation is cost effective.

3.3 Clos Network

Clos network is a non-blocking multistage network as stated in Chapter two. Figure 3.3 shows a typical N x M Clos network represented by $C_{N, M}$ [1,6]

The blocks I and III are always crossbar switches and II is a crossbar switch for a 3 stage Clos network. In implementations of higher order Clos networks, II is a lower order Clos network.

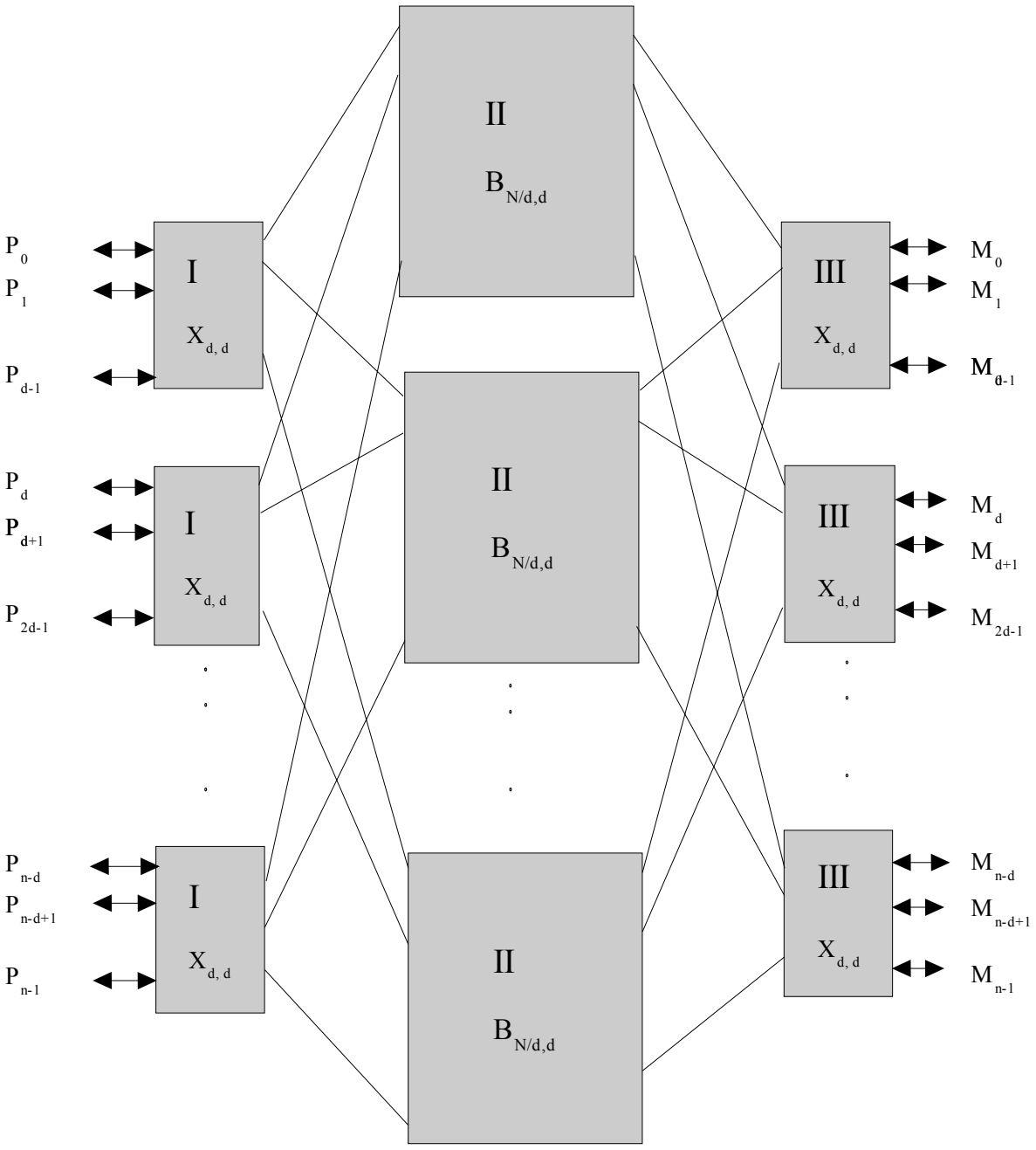

Figure 3.2, Benes Network[6]

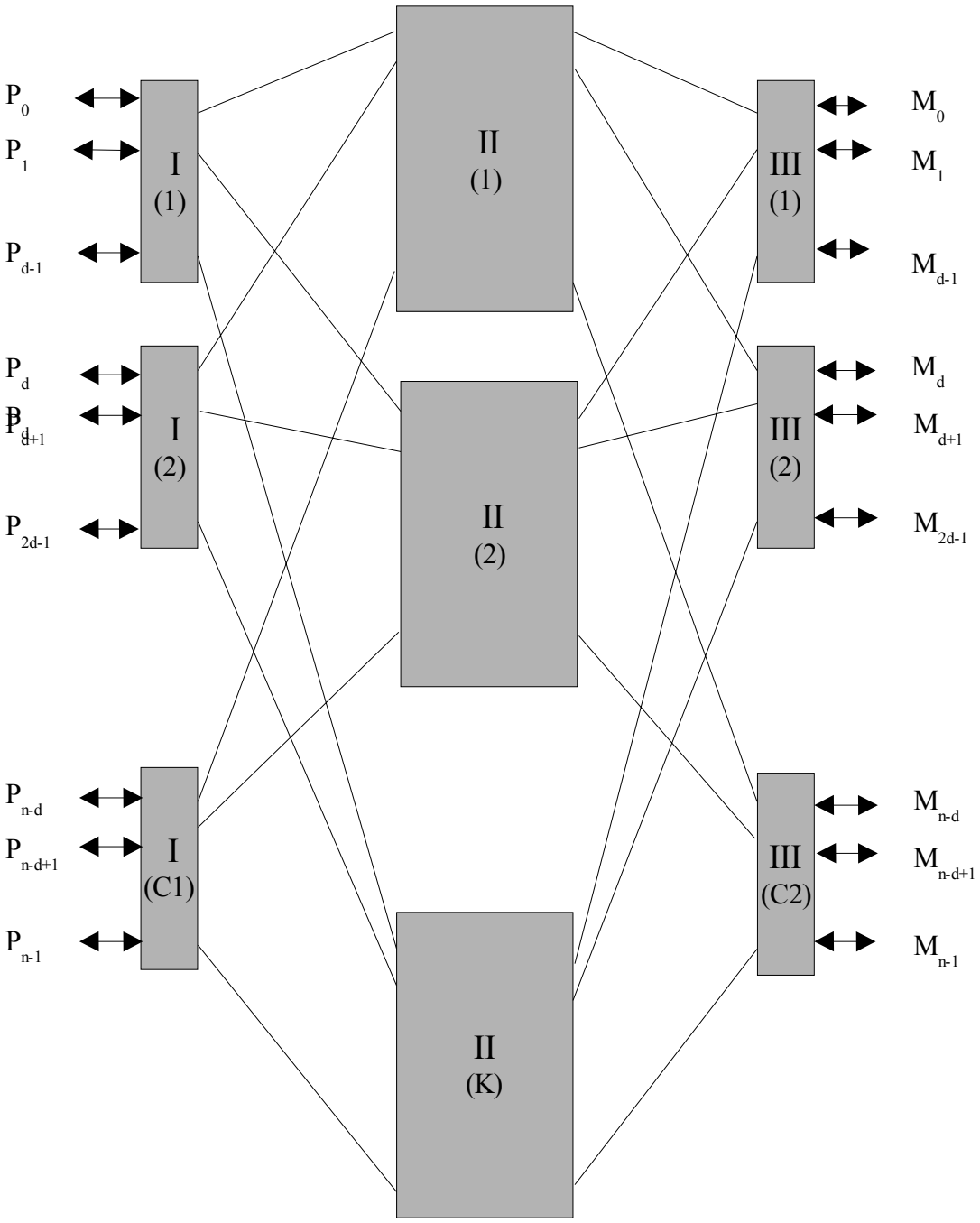

Figure 3.3, Clos Network[6]

N: Number of processors

M: Number of Memory blocks

K: Number of Second stage switches

C1: Number of First stage switches

C2: Number of Third stage switches

For a three stage Clos network, I = $X_{N/C1,K}$, II = $X_{C1,C2}$, III = $X_{K,M/C2}$ and the condition for a non-blocking Clos implementation is K = N/C1 + M/C2 - 1.

A three stage Clos implementation for N = 16, M = 32, C1 = 4, C2 = 8 has K = 16/4 + 32/8 - 1 = 7. Each 1st stage switch becomes a 4 x 7 crossbar switch and the 2nd stage switch becomes a 4 x 8 Crossbar switch and each third stage switch becomes a Crossbar switch of size 7 x 4. ($I = X_{4,7}$ $II = X_{4,8}$ $III = X_{7,4}$). The complexity of a Clos network is given by C_{clos} = [K (N +M) + K (C1*C2)]. Using the non-blocking condition, K = N/C1 + M/C2 - 1. For N = M & C1 = C2, K = 2N/C1 - 1 and hence

$C_{clos} = (2N/C1 - 1)\ \{2N + C1^2\}$.

For an optimum cross point count for non-blocking Clos networks, $N/C1 = (N/2)^{1/2}$

$=> C1^2 = 2N$

$=> C_{clos} = ((2N)^{1/2} - 1).\ 4N.$ (Approximately)

The Clos network can be implemented for any non-prime value of 'N'. The main advantage of clos network implementation is scalability; however disadvantages are network latency and implementation for smaller networks.

3.4 Complexity Comparison

Table 3.1 shows the complexity comparison of the three networks discussed above. In the table 'I' is the complexity and 'II' is the corresponding network implementation for different values on 'N'; here the number of inputs and number of outputs are assumed to be 'N' for simplicity in comparison.

Table 3.1 Table for Comparison of Complexity

N	Crossbar		Benes		Clos	
	I	II	I	II	I	II
2	4	X(2,2)	4	B(2,2)	4	C(2,2)
3	9	X(3,3)	9	B(3,3)	9	C(3,3)
4	16	X(4,4)	24	B(4,2)	36	C(4,2)
5	25	X(5,5)	25	B(5,5)	25	C(5,5)
6	36	X(6,6)	80	B(8,2)	63	C(6,3)
7	49	X(7,7)	80	B(8,2)	96	C(8,4)
8	64	X(8,8)	80	B(8,2)	96	C(8,4)
9	81	X(9,9)	81	B(9,3)	135	C(9,3)
10	100	X(10,10)	224	B(16,2)	135	C(10,5)
11	121	X(11,11)	224	B(16,2)	180	C(12,6)
12	144	X(12,12)	224	B(16,2)	180	C(12,6)
13	169	X(13,13)	224	B(16,2)	189	C(14,7)
14	196	X(14,14)	224	B(16,2)	189	C(14,7)
15	225	X(15,15)	224	B(16,2)	275	C(15,5)
16	256	X(16,16)	224	B(16,2)	278	C(16,8)
32	1024	X(32,32)	576	B(32,2)	896	C(32,8)
64	4096	X(64,64)	1408	B(64,2)	2668	C(64,16)
81	6561	X(81,81)	1701	B(81,3)	4131	C(81,9)
128	16384	X(128,128)	3328	B(128,2)	7680	C(128,16)

Chart 1, shown in Figure 3.4, is the graph of complexity of the three topologies versus N, the number of processors or memory blocks; for lower values of N (N <= 16). Chart 2, shown in Figure 3.5, is the graph of complexity of the three topologies versus N, the number of processors or memory blocks, for higher values of N (N >= 16).

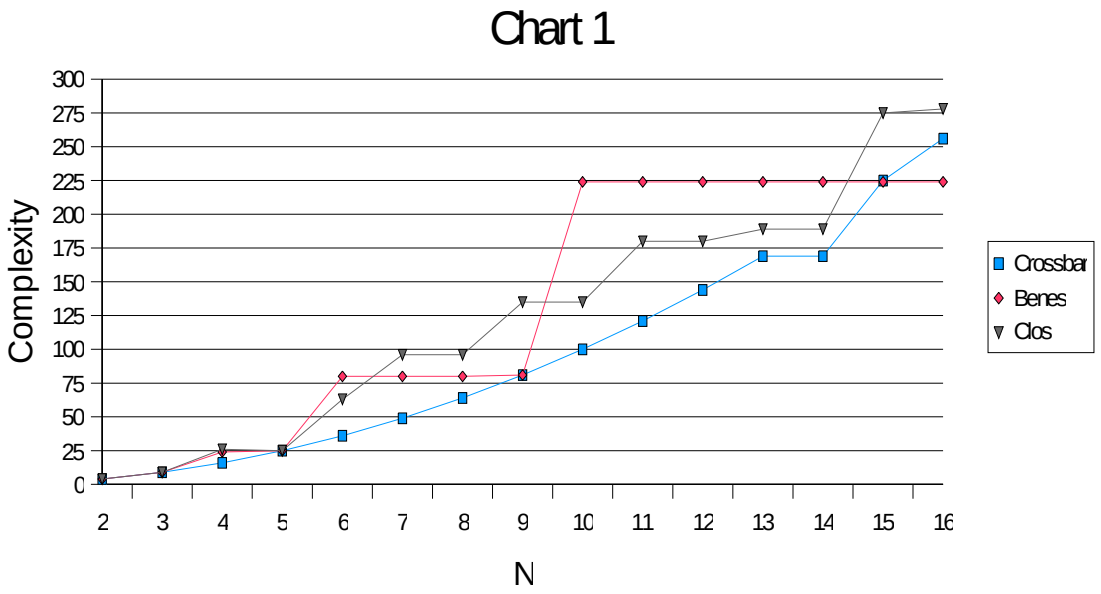

Figure 3.4, Complexity Chart for N<= 16 [6]

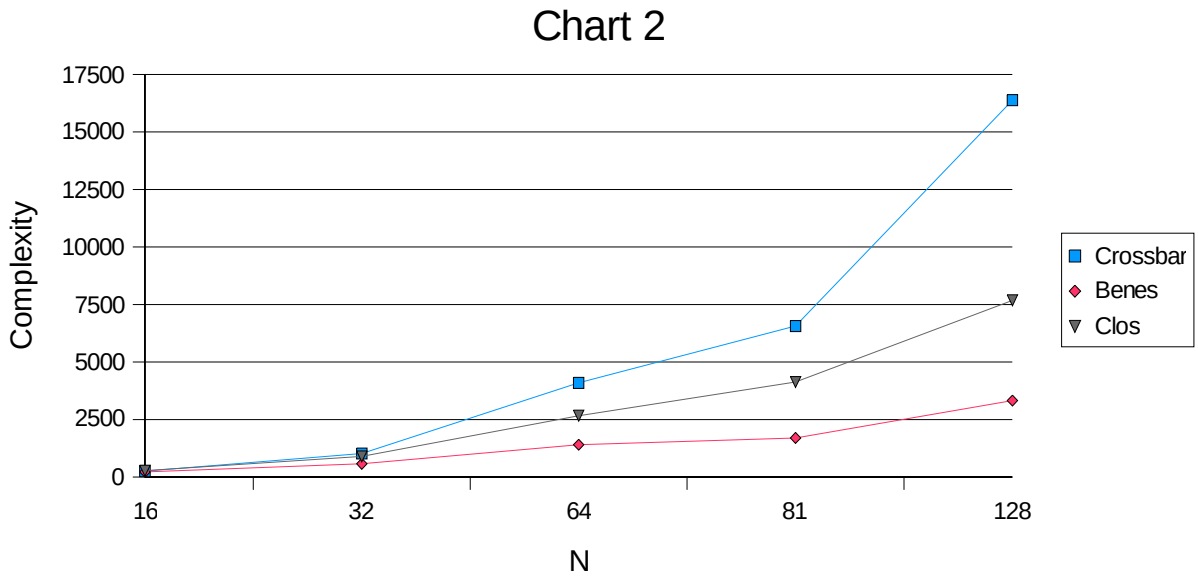

Figure 3.5, Complexity Chart for N>=16 [6]

The equations used for calculating the complexities of the three topologies are as follows:

$C_{clos} = (2N/C1 - 1)\ \{2N + C1^2\}$,

For $N/C1 = (N/2)^{1/2}$, taken to the closest integer value.

$C_{benes} = [(N/d)* (2\log_d N-1)* d^2]$,

$C_{crossbar} = N^2$

From table 3.1 and the charts in Figures 3.4 and 3.5, one can conclude that the crossbar topology has the lowest or almost equal complexity for the values of N<=16. Therefore the crossbar network is the best interconnect implementation for N<=16. This network is faster than any other network since the hardware required is less as compared to other networks. It is also a non-blocking network as there is connection capability of every input to every output. For systems having configurations of more than (16 x16) but less than (32 x32), one has to trade–off between speed and complexity, since for multistage networks like Benes or Clos, the complexity is less as compared to the crossbar network; however at the cost of speed. For systems with configurations of more than (32 x 32), the Benes network proves to be the best choice.

The HDCA system in consideration requires an interconnect network system with complexity of not more than 256 (16 x 16). Hence, from the above discussion, one can conclude that the crossbar switch is the best suitable for the HDCA system. The HDCA system as described in [5,18] has a single bus communicating between the shared memory and the computing elements (CEs). With the interconnect network acting as the communication channel between the above said components the system is as shown in Figure 3.6. The shared memory is divided into a number of blocks as can be seen in Figure 3.7. Here the number of memory blocks, ‘M’ is chosen such that $2^c = M$.

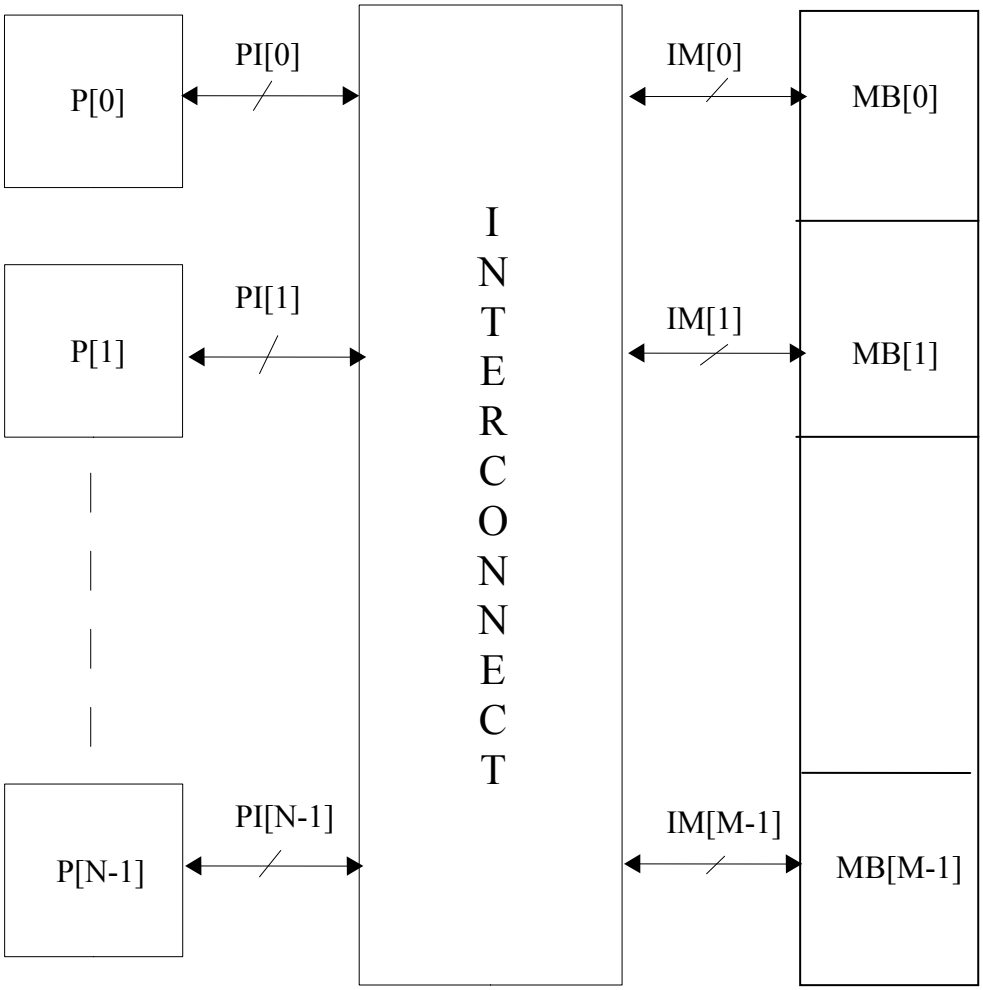

Figure 3.6, Multiprocessor Shared Memory Organization

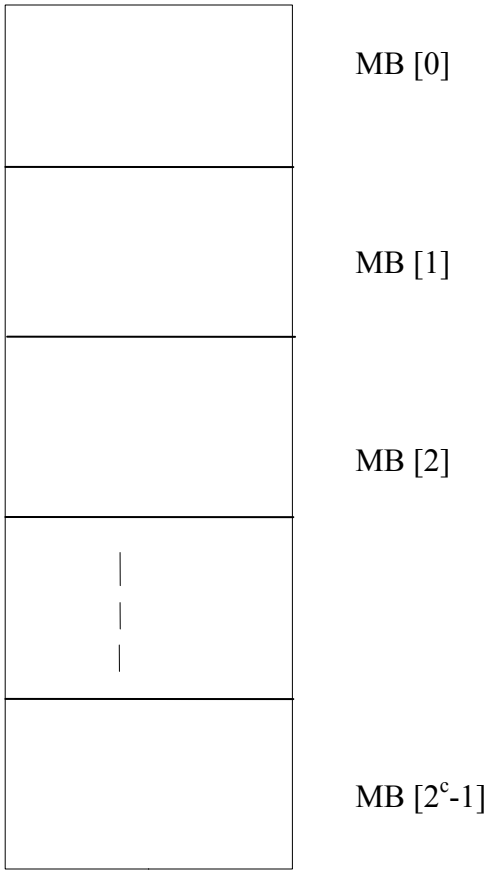

Figure 3.7, Shared Memory Organization

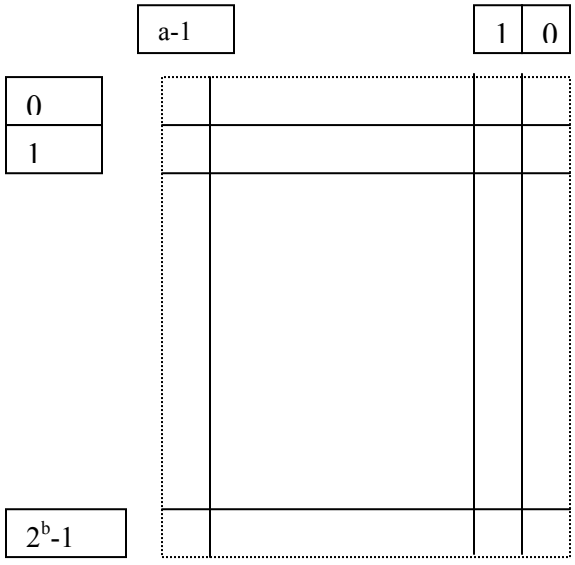

Figure 3.8, Organization of Each Memory Block

A main issue in implementing a crossbar network is arbitration of processor requests for memory accesses. The arbitration issue comes into play when two or more processors, Computing Elements in this case, request for memory access within the same memory block. There are mainly two kinds of protocols for designing a crossbar interconnect, fixed priority and variable priority protocols. Round robin protocol, First-In First-Out (FIFO), Least Recently Used (LRU) and Last-In First-Out (LIFO) are some of the fixed priority protocols used. A fixed priority protocol assigns fixed priorities to the requesting processors. In case of conflict, the processor having the highest priority ends up having its request granted all the time. In the HDCA system, a variable priority protocol is used, here priorities assigned to processors dynamically vary over time based on some parameters of the system. In the HDCA system of Figure 1.1, all processors are dynamically assigned a priority depending upon individual input queue depth (see Figure 1.2) at any given point of time. In the case of conflict, the processor having the highest (deepest) queue depth at that point of time gets the request grant. When two processors accessing the same memory block have the same queue depth and priority is then determined by "processor identification number". The HDCA system needs to settle some more arbitration issues for processor conflicts. A detailed design of the implementation is described in the next chapter.

Chapter Four

Design of the Crossbar Interconnect Network

A detailed design of the crossbar interconnect network suitable for integration into the HDCA system is described in this chapter. A general overview of the organization of the computing elements, the crossbar interconnect and the shared memory with blocks is shown in the Figure 3.6. In this figure PI[i] represents the bus structure between Processor and Crossbar Switch. IM[i] represents bus structure between Crossbar Switch and Memory Blocks. The Figures 4.1a and 4.1b shows the detailed diagram of both the bus structures.

4.1 Organization of Shared Memory

The utilized shared memory organization used is shown in Figures 3.7 and 3.8. From Figure 3.7, there are (2^c = M) memory blocks and the organization of each memory block is shown in Figure 3.9. In each memory block, there are 2^b addressable locations, of 'a' bits width. Hence the main memory which includes all the memory blocks has 2^{b+c} addressable locations of 'a' bits width. Therefore the width of the address bus of each processor is (c + b) bits wide and the data bus of each processor is 'a' bits wide.For the address bus "c" represents the most significant bits of the bus and "b" represents the least significant bits of the bus.

4.2 Basic Design of Crossbar Switch

Refering to the Figures 3.7 and 4.1a and 4.1b the signal ctrl[i] goes high when the processor is requesting a connection to a particular memory block. The rw[i] signal goes high when a processor has to write into the memory block and goes low when it has to read out from a memory block. The qdep[i] is the queue depth of the processor[i], addr_bus[i] is the (b+c) address of the memory location where 'c' is the block address with which the processor wants to communicate. flag[i] is an output of the crossbar switch; it goes high when the processor request is granted access to the memory block. The value of flag[i] is decided by a priority logic which is described in a later section. The bit widths of 'ctrl[i]', 'rw' and 'flag[i]' remain one bit, however the bit widths for

other inputs and outputs can vary. The design is adaptable to any number of processors and memory blocks.

Using Figure 4.2a, a generic representation of the interconnect to understand the working of the crossbar, lets take an example of processor[i] requesting access to memory block[j] represented by MB[j]. This means ctrl[i] goes high and the 'c' bits of addr_bus is 'j'. Processor[i] gets connected to the bus D[i][j] ,through a decode logic D[i] as shown in Figure 4.2a. The address is decoded by the decoder and connection is established between processor[i] and D[i][j]. Every memory block has a priority logic block 'prl_behav[j]' as shown in Figure 4.2 and 4.4. The function of this logic block is to grant access to a processor having the deepest queue depth among the processors requesting access to the same memory block. Once a processor gains access to a memory block indicated by the flag[i] going high, the connection between the processor[i] and the memory block MB[j] is established and data flow from or into the memory takes place depending on the value of rw[i]. This connection stays active as long as the processor holds deepest queue depth or the request for the memory block by processor that is ctrl[i] is high. For the case of processors having the same queue depth the processor having highest processor identification number gets the highest priority. Each priority logic block sets a value of 'sub_flag'; these values are re-evaluated again if the situation of two or more processors requesting access to the same memory block arises. Otherwise, there is no change in the evaluation of 'flag'. This is achieved by a combinatorial logic refer to Appendix[A2] A flow chart showing the algorithm of the priority logic block in the Figure 4.4; is shown in the Figure 4.5. The mbaddr[x] signal shown in the flow chart corresponds to the 'c' MSBs. The flag[i] signal in Figure 4.5 is shown as signal sub_flag[i] in Figure 4.4. The number of computing elements or processors is assumed to be four in this case; however this logic holds good for any number of processors. The processors whose ctrl signal is high that is those requesting access to the memory are considered for the algorithm in discussion.

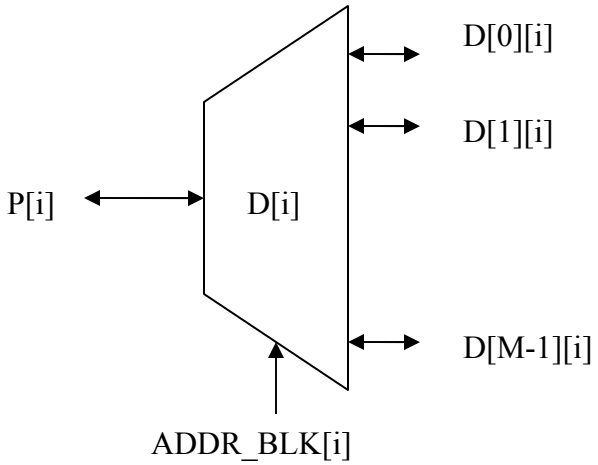

Figure 4.3, Decode Logic D[i]

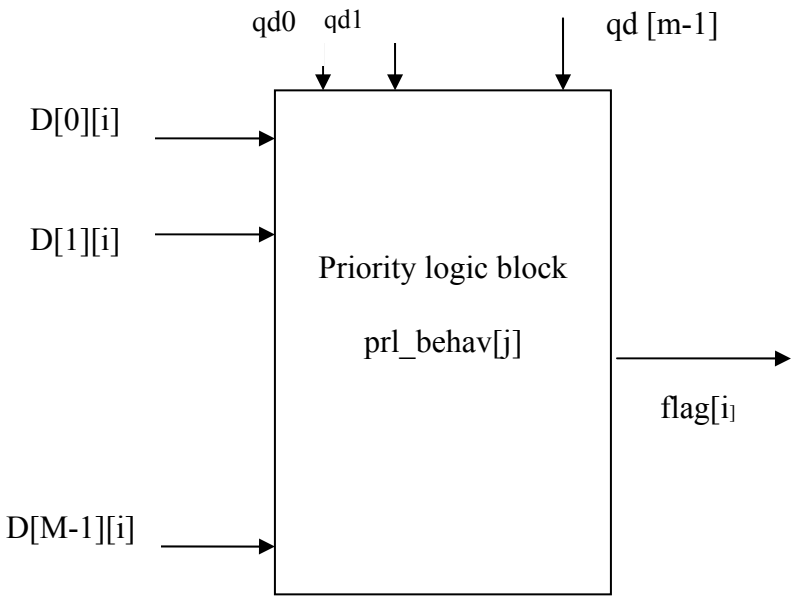

Figure 4.4, Priority Logic Block

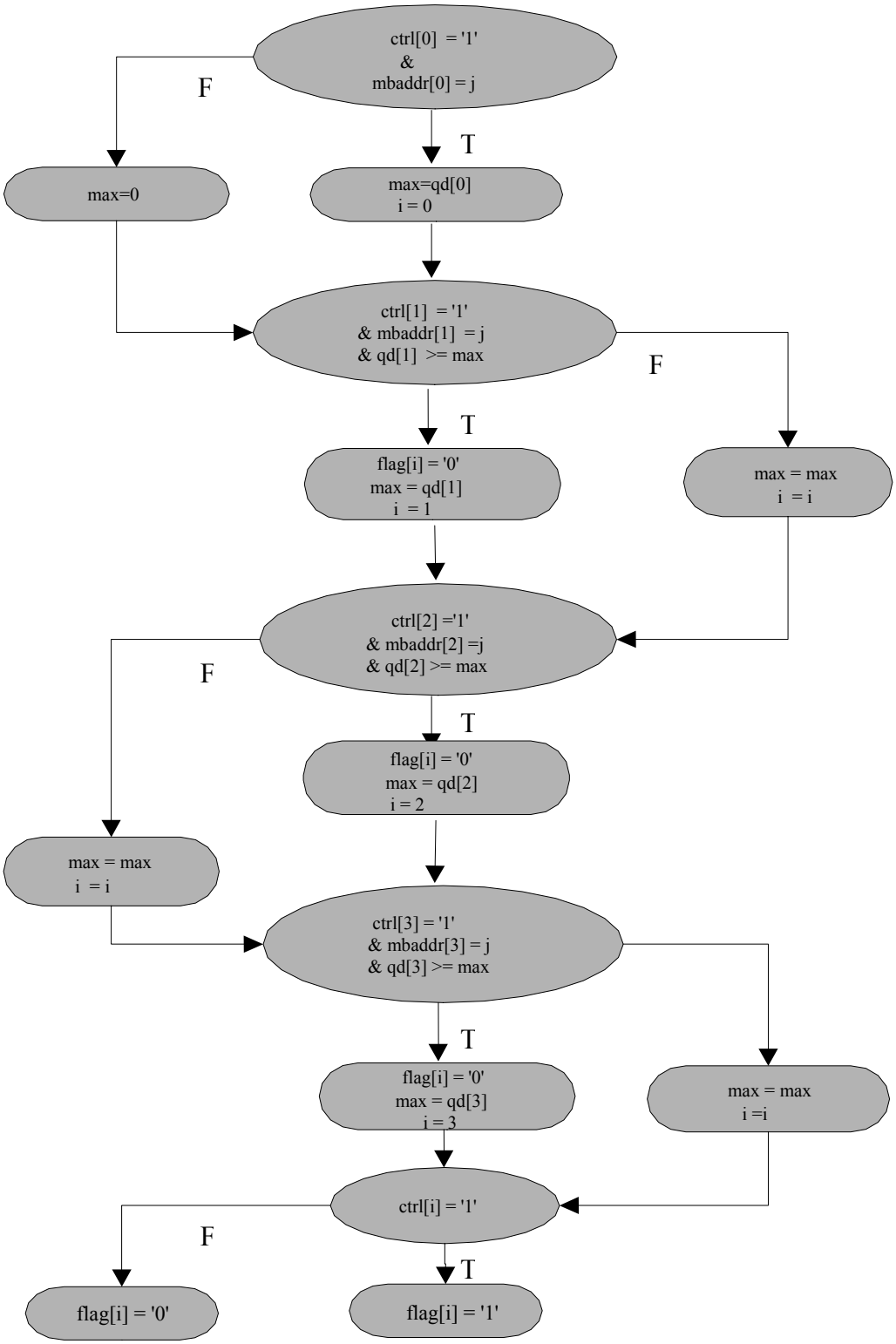

Figure 4.5, Flow Chart for the Priority Logic Block in Figure.4.4

The block compares current maximum queue depth with the queue depth of every processor starting from the 0^{th} processor. The integer values 'i'shown in the Figure 4.5 is the processor identification number, having the deepest queue depth at that particular instant. Once the comparison is done as depicted in the Figure 4.5 the flag[i] goes high, granting the access to the particular processor satisfying the logical criteria explained in the flow chart.

Thus it can be seen that the interconnect network described here gives all the processors the choice of simultaneous access to the shared memory, granted by the priority logic. The best case scenario is when all the processors have their requests granted by the priority logic. This is possible when all processors are requesting access to different memory blocks that is no two processors have the same ADDR_BLK.

The crossbar interconnect is described in VHDL using two different approaches. One is a behavioral approach; it has a single function which describes the processor prioritization done by the priority logic blocks. The VHDL code for this approach is in Appendix A1.The other is a component level (RTL level) model of the crossbar switch. This particular model has basically two types of components; the decode logic block and a priority logic block. Each processor has a pair of decoder and priority logic blocks between the processor and the memory block. The VHDL code of the interconnect is shown in Appendix A2

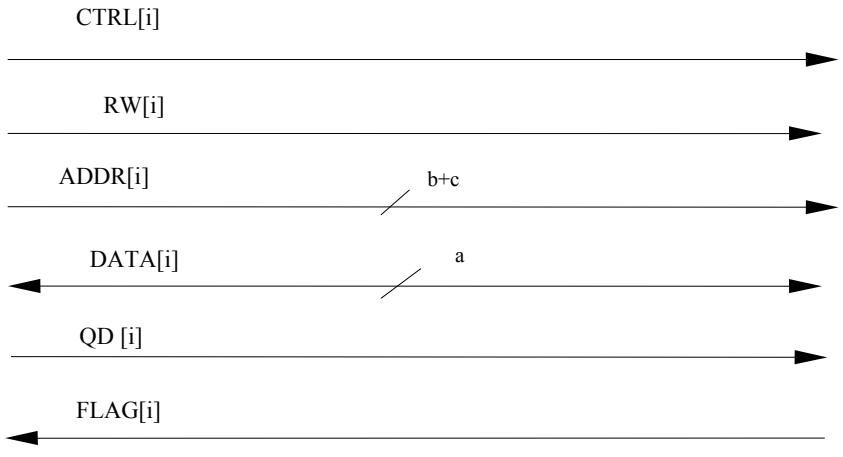

Figure 4.1a, PI[i] and D[i][j] Bus Structure

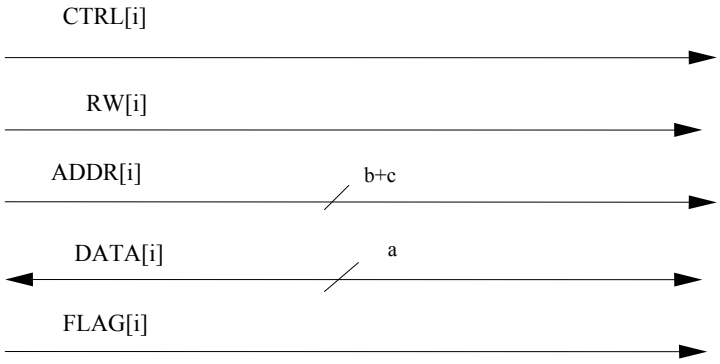

Figure 4.1b, IM[j] Bus Structure

A graph of the number of gates required in implementation of different sizes of interconnection networks was plotted as shown in Figure 4.6. It shows that it scalable for smaller designs. However the projected values of the gate count for values greater than 16 are expected to shoot up exponentially. The exact values could not be found out for networks greater than 8x8, since the network could not fit in the largest FPGA chip available in Xilinx 6.2.3i package.

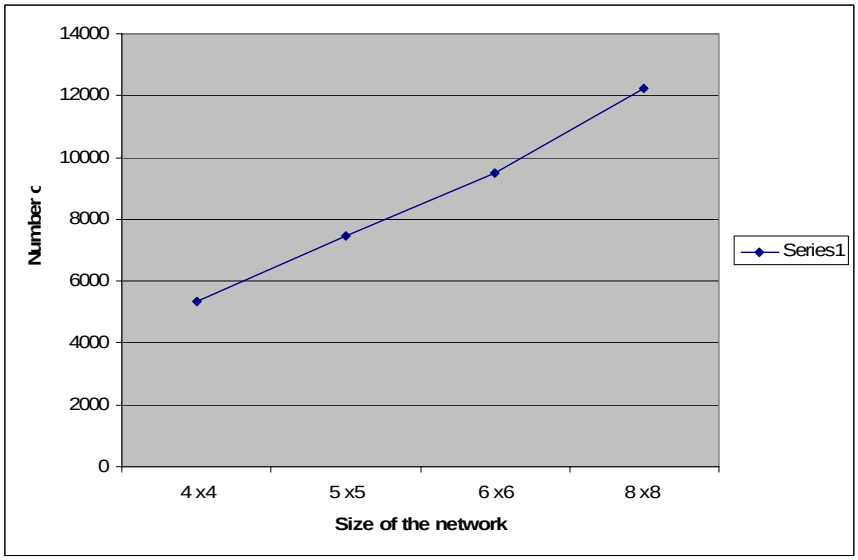

Figure 4.6, Plot Showing Gate Count vs Size of Crossbar Interconnect Network

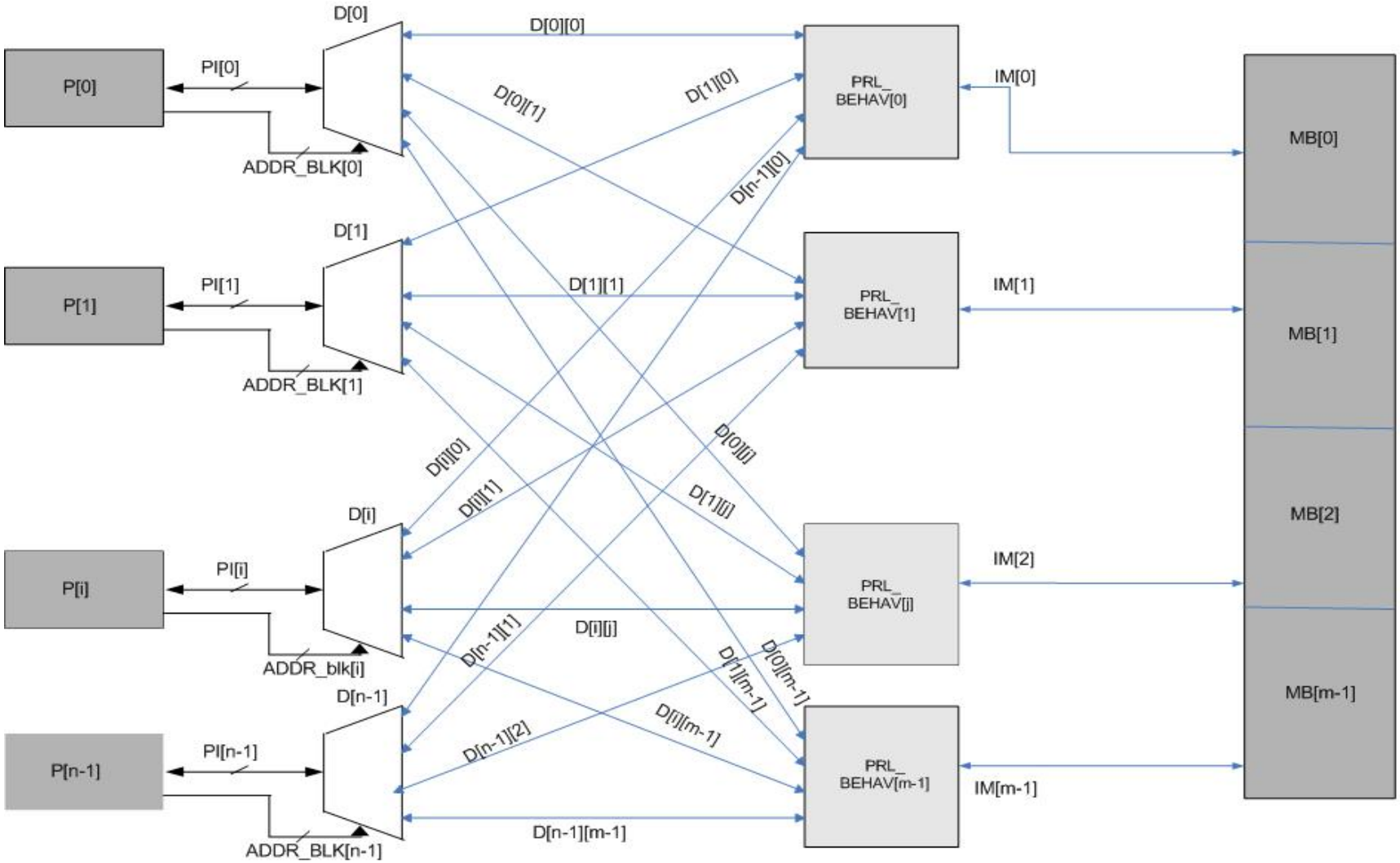

Figure 4.2a, Detailed Block Diagram Interconnection Network for N x M Network

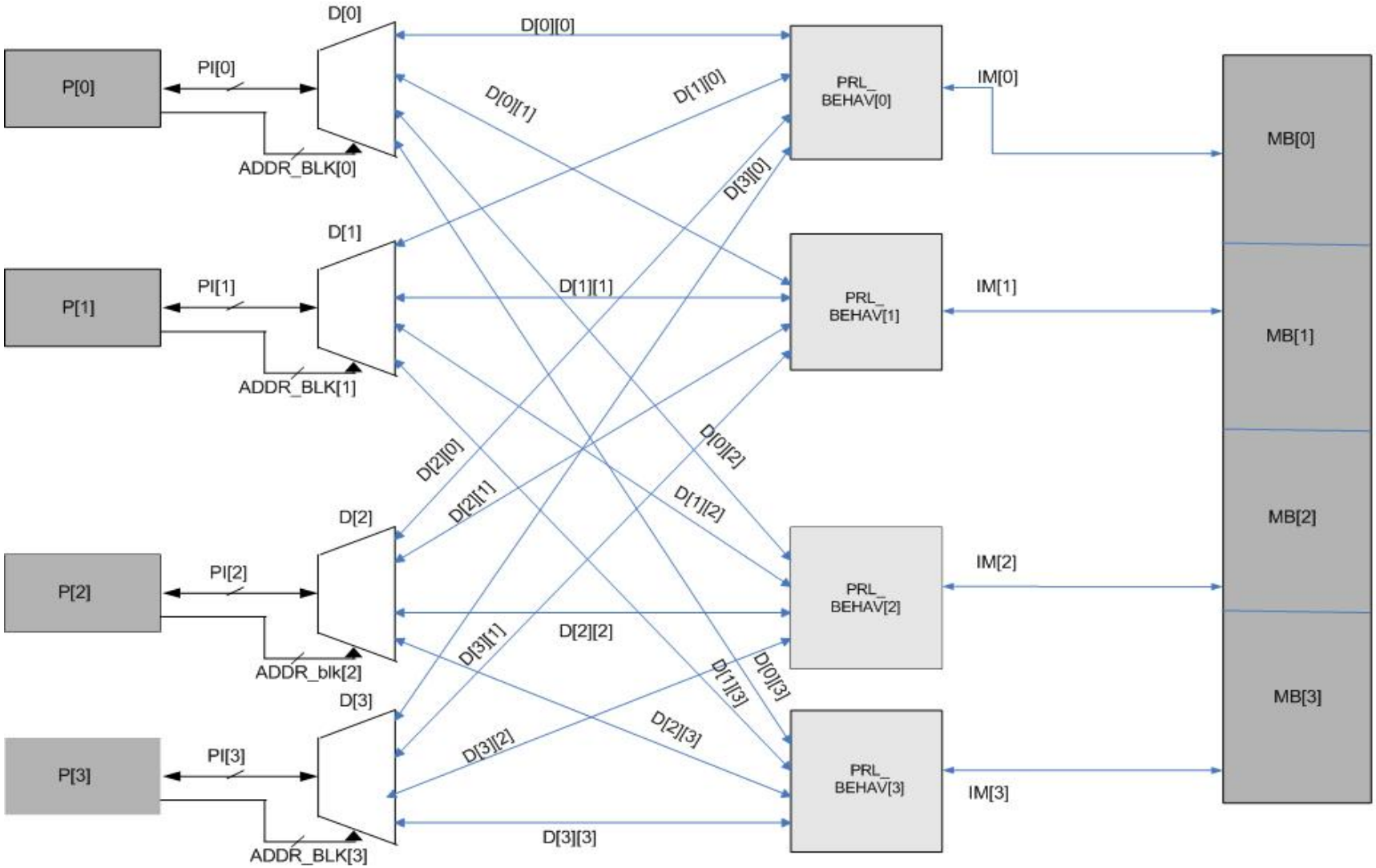

Figure 4.2b, Detailed Block Diagram of the Interconnect Network

Chapter Five

Implementation of Variable Priority Interconnection and Virtual Prototype Validation of Correct Independent Operation and Operation as the Processor-Memory Interconnect of the HDCA

5.1 VHDL Design Capture

As explained in Chapter 4, the crossbar interconnect design capture in VHDL has been achieved two different ways. One is a behavioral approach [6](see Appendix A1) and the other is a component level (RTL level) approach(see Appendix A2). The structural equivalent of the behavioral description is shown in Figure 5.1. The component level approach is structurally outlined as shown in Figure 4.2a and 4.2b of chapter 4. The variable priority crossbar interconnect of [6] was validated to function correctly. It was never integrated into the HDCA. The version of this interconnect as addressed in Chapter 4, will first be independently validated in this Chapter. It will then be validated in this Chapter to perform correctly within the HDCA.

5.1.1 Modifications to Behavioral Approach

The main VHDL code [6] has a function ‘flg’, an entity ‘main’ and a process’P1’. This code is scalable(generic parameterized coding in an HDL sense) in the sense that it is possible to increase the number of processors, memory blocks, memory locations in each memory block, data width and the width of the queue depth bus. The VHDL code for the interconnect as it will be tested in the HDCA is included in the Appendix A1. Here the number of processors, the number of memory blocks and also the number of addressable locations in each memory block is assumed to be ‘4’. Queue depth of each computing element is also assumed to be four bits wide(See Figure 4.2a)

The code described in [6] was developed to be implemented with Xilinx Foundation series software hence it had to be modified in the following manner to display correct results using the Xilinx ISE 6.2i and ModelSim 5.7g CAD software packages. In [6] the function ‘flg’ consists of two ‘for-loops’ embedded in each other. This is modified to have a single ‘for-loop’ module. The revised code is attached in the Appendix A1.

5.1.2 Implementation of 4 x 4 Crossbar Interconnect

The VHDL coded interconnect network with crossbar switch integrated in HDCA [Appendix A1 and A2] are synthesized and virtually post placed and routed using Xilinx 6.2.3i CAD Tool and Modelsim 5.7g as a simulation tool. The design is tested using Xilinx XC2V 8000 from the Virtex II family of chips. It has 8 million gates configuration. The entire design development, testing and validation is done on a system with following parameters: Intel Pentium4, 3.00 GHz and 1 GB of RAM. The operating system used is Microsoft Windows XP, service pack 2.

The resource utilization and timing summary for the functionally coded crossbar switch[Appendix A1]:

Device Utilization Summary:

Logic Utilization:	
Number of Slice Flip Flops:	64 out of 93,184 1%
Number of 4 input LUTs:	697 out of 93,184 1%
Logic Distribution:	
Number of occupied Slices:	353 out of 46,592 1%
Number of Slices containing only related logic:	353 out of 353 100%
Total Number 4 input LUTs:	697 out of 93,184 1%
Number of bonded IOBs:	78 out of 824 9%
IOB Flip Flops:	32
Number of GCLKs:	1 out of 16 6%

Total equivalent gate count for design: 5,352

Timing Summary:

Speed Grade: -5

Minimum period: 3.455ns (Maximum Frequency: 289.436MHz)
Minimum input arrival time before clock: 23.698ns
Maximum output required time after clock: 4.840ns
Maximum combinational path delay: 24.490ns

The resource utilization and timing summary for the structurally coded crossbar switch [Appendix A2]:

Device Utilization Summary:

Logic Utilization:
Number of Slice Flip Flops: 106 out of 93,184 1%
Number of 4 input LUTs: 624 out of 93,184 1%
Logic Distribution:
Number of occupied Slices: 318 out of 46,592 1%
Number of Slices containing only related logic: 318 out of 318 100%
Number of Slices containing unrelated logic: 0 out of 318 0%
Total Number 4 input LUTs: 624 out of 93,184 1%
Number of bonded IOBs: 78 out of 824 9%
IOB Flip Flops: 32
Number of GCLKs: 1 out of 16 6%

Total equivalent gate count for design: 5,256

Timing Summary:

Speed Grade: -5

Minimum period: 7.836ns (Maximum Frequency: 127.616MHz)
Minimum input arrival time before clock: 7.714ns
Maximum output required time after clock: 5.349ns
Maximum combinational path delay: No path found

5.1.3 Functional Testing of a 4 x 4 Crossbar Interconnect Network

Various scenarios tested for validation of the design are described and the results are shown in simulation tracer Figures. The values shown in the tracers are represented in hexadecimal format:

Case 1:

All processors write to different memory locations in different memory blocks. All the memory locations are being written. The table 5.1 gives the addresses and contents of each addres after each processor does a write.

Table 5.1 Shared Memory Address Space and Contents

Addr.Location	Data(case1,2)
0000[0]	1
0001[1]	2
0010[2]	3
0011[3]	4

0100[4]	5
0101[5]	6
0110[6]	7
0111[7]	8
1000[8]	1
1001[9]	2
1010[A]	3
1011[B]	4
1100[C]	5
1101[D]	6
1110[E]	7
1111[F]	8

In the cases described below the method to depict the parameters such as queue depth(qdep), read or write(rw), ctrl , processor requesting access to the memory(addr_bus) and the data written(data_in) is as follows(the individual values are also in hexadecimal):

Table 5.2 Parameters Depiction in the Cases Described

Processor Number	P3	P2	P1	P0	Value in hex combined as shown in case 1
qdep	1	2	3	4	1234
rw	1	1	1	1	F
ctrl	1	1	1	1	F
addr_bus	C	8	4	0	C840
data_in	5	1	5	1	5151

This method is used for all the cases described in this section and the section following it(section 5.1.4)

qdep is x"1234".

rw: x"F"

ctrl: x"F"

Data is written in order of 1st, 2nd, 3rd and 4th data location in each memory block.

addr_bus: x" C840" data_in: x"5151" (Value shown in the table for addr_bus and data_in as an example)

addr_bus: x" D951" data_in: x"6262"

addr_bus: x" EA62" data_in: x"7373"

addr_bus: x" FB73" data_in: x"8484"

Case2:

All the processors reading out the data written in case 1 .Hence all the parameters remain the same except for r_w. The result of both the cases is observed in the simulation tracer1(see page 47)

rw: x"0"

Case 3:

qdep :x"1234"

rw: x"0"

addr_bus: x"CD56"

Processors '0' and '1' try to access the same memory block '1'.However since the 'qdep' of processor '0' (qdep=4) is greater than processor '1'(qdep=3), processor '0' is granted access to read out the value '7' (addr. location: 6)

Similarly processors '2' and '3' try to access the same memory block '3'. However the 'qdep' of processor '2' (qdep=2) is greater than processor '3' (qdep=1).Processor '2' is granted access to read out the value '6' (addr. Location: D)

The result observed in the simulation tracer 1.(see page 47)

Case 4:

qdep: x"2244"

rw:x"0"

addr_bus: x"CD56"

This is a situation where two processors are accessing the same memory block and both of them have same 'qdep'. The processor with highest processor number is granted access. As can be seen from the simulation tracer 2 (see page 48) that processor '1' gains access to memory block '1' and processor '3' to memory block '3'.

Processor 1 reads out ‘6’from location ‘5’

Processor 3 reads out ‘5’from location ‘C’

Case 5:

qdep:x”1234”

rw:x”F”.

addr_bus:x”CD56”

data_in: x” AAAA”

Situation similar to case 3 as described above.

Here processor ‘0’ and processor ‘2’ gain access to the memory blocks ‘1’ and ‘3’ respectively.

Processor ‘0’ writes in ‘A’ at location ‘6’

Processor ‘2’ writes in ‘A’ at location ‘D’

Case 6:

All the parameters kept same except for rw.

rw:x”0”

The values written in case 5 are observed in this case. The result can be viewed in the Figure simulation tracer 2(see page 48).

Data_out: x”5A6A”

Case 7:

Situation where all the four processors access the same memory block 2.

addr_bus:x”BA98”

rw:x”F”

data_in:x”CCCC”

The qdep is the same for all.

qdep:x”2222”

According to the logic designed the processor ‘3’ with the highest processor number gains access to the memory block ‘2’.The result observed in the simulation tracer 2(see page 48).

Case 8:

Read out the value written in case 7.The processor ‘3’ gained access and wrote ‘C’ at location ‘B’.

The result observed in simulation tracer 2(see page 48).

Case 9:

All the cases described so far had *ctrl* as *“F”*

In this case we observe the results changing the ctrl.

ctrl:x”3” => “0011” in binary.

rw: x”F”

data_in: x”BBBB”

addr_bus: x”FA32”

qdep :x”4565”

Processor ‘2’ and ‘3’ are not requesting access to the memory.

Processor ‘0’ and ‘1’ are requesting access for the same memory block ‘0’.

However processor ‘1’ gains the access since its qdep(=6) is greater than qdep(=5) processor ‘0’. Processor ‘1’ writes ‘B’ at location ‘3’

The result observed in the simulation tracer 3(see page 49).

The next stage is reading out the value written

Data out: x”CDBA”

Addr_bus: x”FA32”

Case 10:

Case where none of the processors are requesting access to the memory blocks.

ctrl:x”0”

The result observed on the simulation tracer 3 (see page 49).

There is no change in the value.

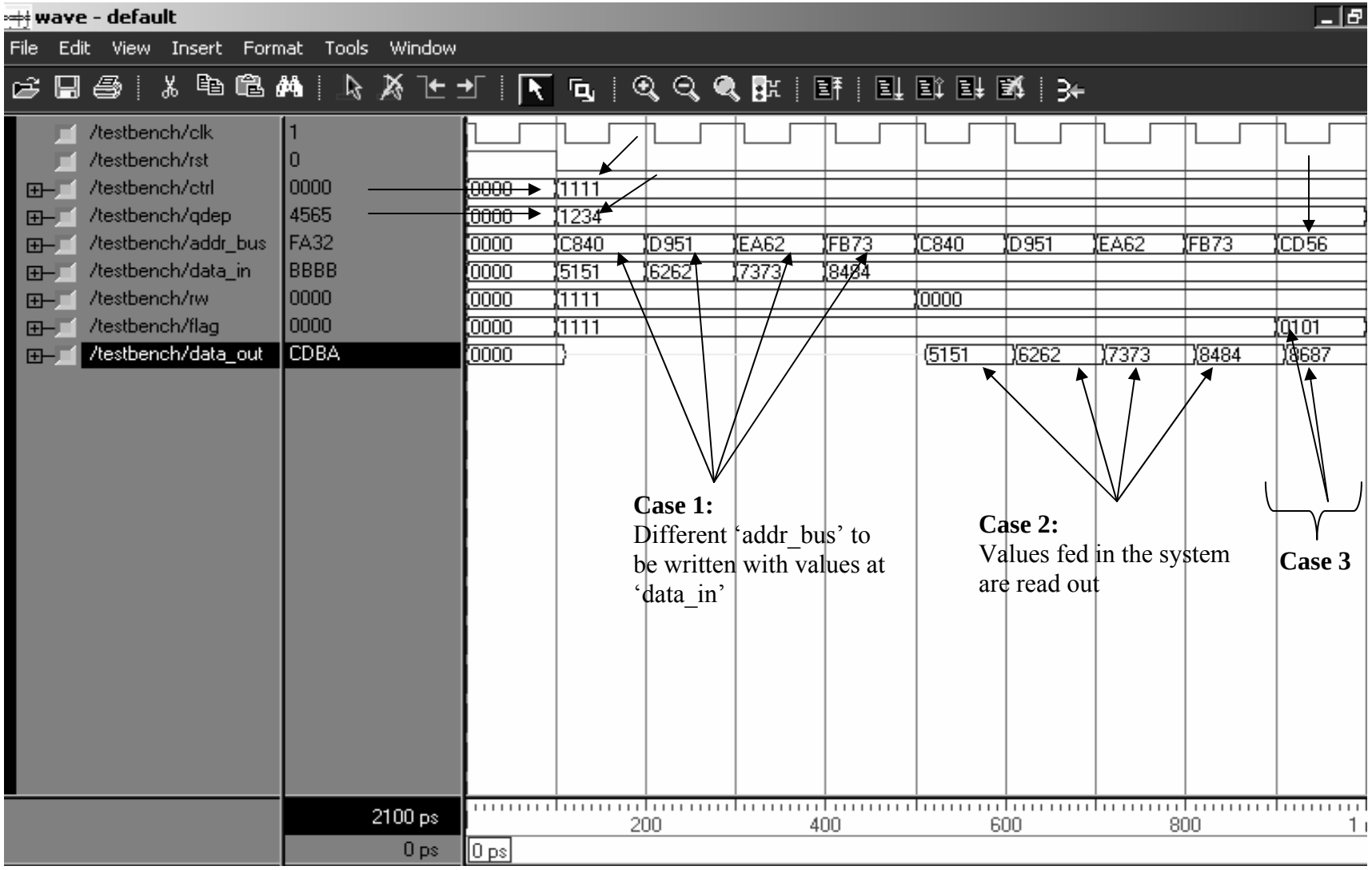

Figure 5.1, Simulation Tracer 1

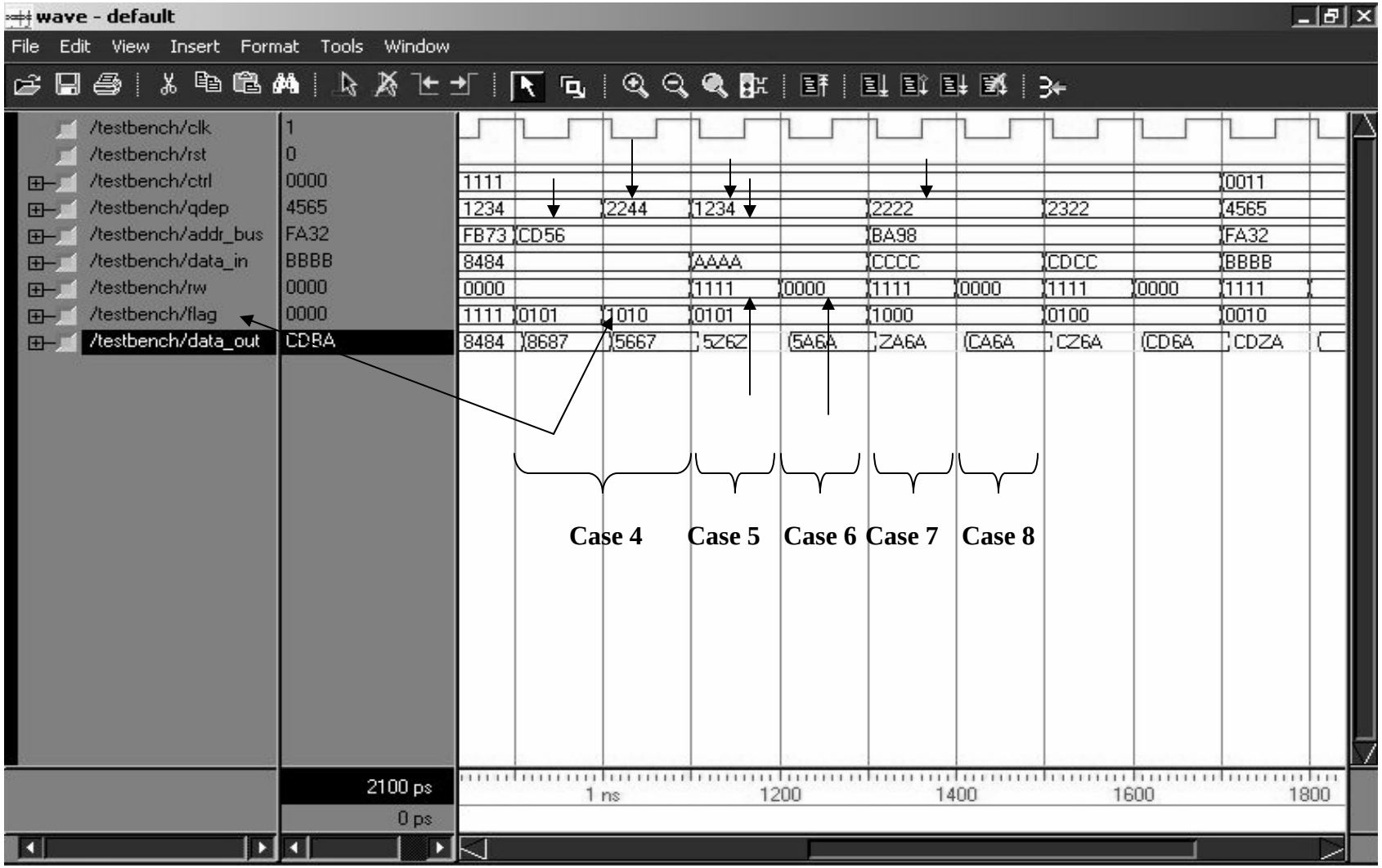

Figure 5.2, Simulation Tracer 2

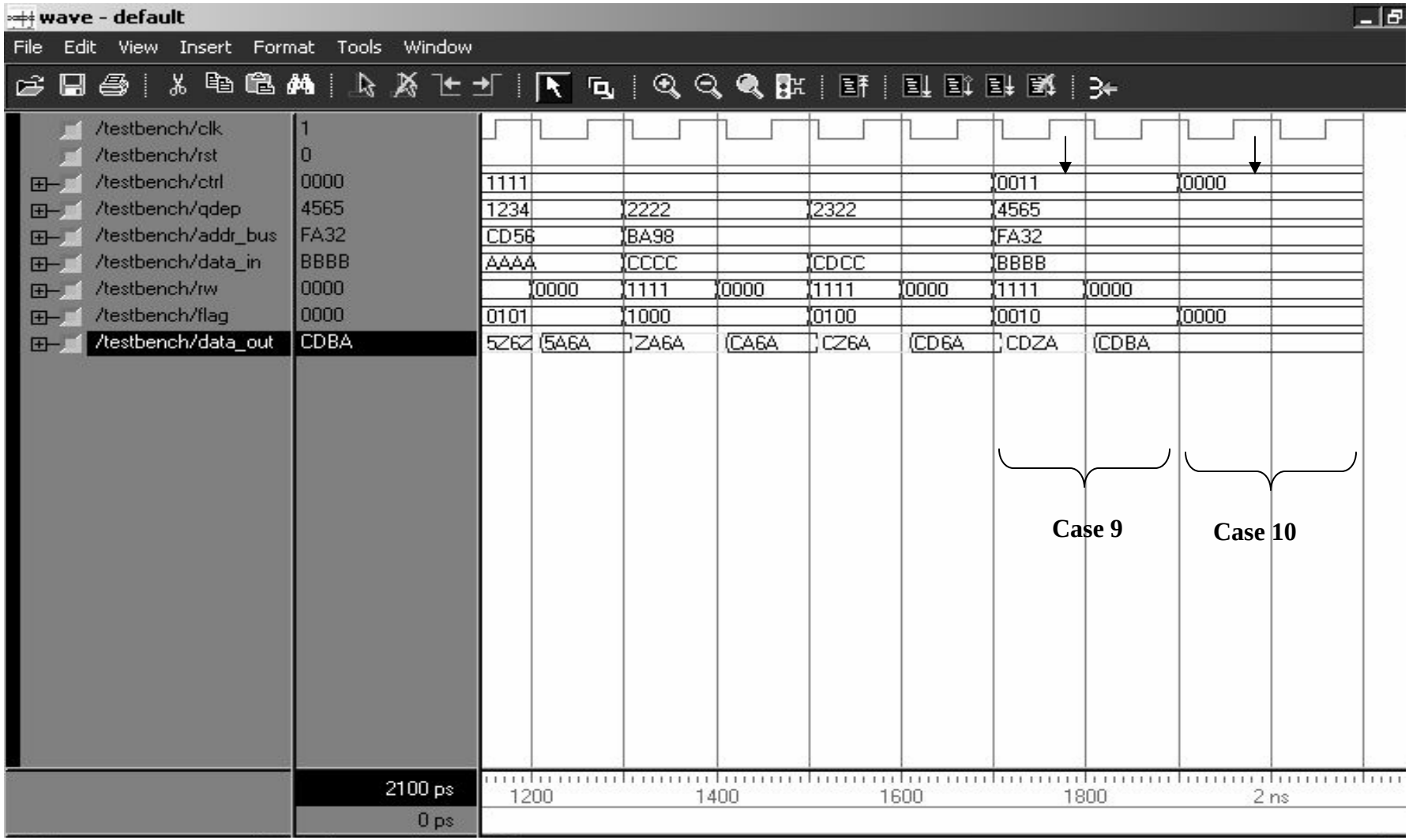

Figure 5.3, Simulation Tracer 3

5.1.4 Component Level Description and Testing

The main VHDL code for this module 'gate_ic_a' consists of two components, 'dec_ic_a' and 'prl_behav' as can be seen in the Appendix A2. This design is also scalable as the code in Appendix A1 described above. The number of processors, the number of memory blocks and also the number of addressable locations in each memory block is assumed to be '4'. Queue depth of each computing element is also assumed to be four bits wide.

Various test cases for the verification of design are described and the results are shown in the simulation tracer Figures, it should be noted the values shown in the tracers are in hexadecimal format.

Case 1:

All the four processors are accessing different memory blocks. Eventually all the memory locations in all the blocks are written into with data. The table gives the address and memory locations of the values fed in.

Table 5.3 Shared Memory Address Space and Contents

Addr.Location	Data(case1,2)
0000[0]	F
0001[1]	E
0010[2]	D
0011[3]	C
0100[4]	B
0101[5]	A
0110[6]	9
0111[7]	8
1000[8]	7
1001[9]	6
1010[A]	5
1011[B]	4
1100[C]	3
1101[D]	2
1110[E]	1
1111[F]	0

qdep is x"1234".

rw: x"F"

ctrl: x"F"

Data is written in order of 1st, 2nd, 3rd and 4th data location in each memory block.

addr_bus: x" C840" data_in: x"37BF"

addr_bus: x" D951" data_in: x"26AE"

addr_bus: x" EA62" data_in: x"159D"

addr_bus: x" FB73" data_in: x"048C"

Case2:

All the processors read out the data written in case 1 .Hence all the parameters remain the same except for r_w. The result of both the cases is observed in the simulation tracer4

rw: x"0"

Case 3:

qdep :x"1234"

rw: x"0"

addr_bus: x"EF01"

Processors '0' and '1' try to access the same memory block '0'.However since the 'qdep' of processor '0' (qdep=4) is greater than processor '1'(qdep=3), processor '0' is granted access to read out the value 'E' (addr. location: 1)

Similarly processors '2' and '3' try to access the same memory block '3'. However the 'qdep' of processor '2' (qdep=2) is greater than processor '3' (qdep=1).Processor '2' is granted access to read out the value '0' (addr. Location: F)

Hence the value of flag is *x"5*

The result observed in the simulation tracer 5.

Case 4:

qdep: x"2244"

rw:x"0"

addr_bus: x"EF01"

This is a situation where two processors are accessing the same memory block and both of them have same ‘qdep’. The processor with highest processor number is granted access. As can be seen from the simulation tracer 5 that processor ‘1’ gains access to memory block ‘0’ and processor ‘3’ to memory block ‘3’.
Processor 1 reads out ‘F’ from location ‘0’
Processor 3 reads out ‘1’from location ‘E’

Case 5:

qdep:x”1234”

rw:x”5” => “0101” in binary.

addr_bus:x”EF01”

data_in: x” AAAA”

Situation similar to case 3 as described above.

Here processor ‘0’ and processor ‘2’ gain access to the memory blocks ‘0’ and ‘3’ respectively.

Processor ‘0’ writes in ‘A’ at location ‘1’

Processor ‘2’ writes in ‘A’ at location ‘F’

Case 6:

All the parameters kept same except for rw.

rw:x”0”

The values written in case 5 are observed in this case. The result can be viewed in the Figure simulation tracer 5.

Data_out: x”1AFA”

Case 7:

Situation where all the four processors access the same memory block 2.

addr_bus:x”7654”

rw:x”F”

data_in:x”1111”

The qdep is the same for all.

qdep:x"2222"

According to the logic designed the processor '3' with the highest processor number gains access to the memory block '2'.Value '1' is written at location '7'.The result observed in the simulation tracer 6.

Case 8:

Read out the value written in case 7.The processor '3' gained access and wrote '1' at location '7'.

The result observed in simulation tracer 6.

Case 9:

All the cases described so far had *ctrl* as *"F"*

In this case we observe the results changing the ctrl.

ctrl:x"C" => "1100" in binary.

rw: x"F"

data_in: x"EEEE"

addr_bus: x"FE32"

qdep :x"1232"

Processor '0' and '1' are not requesting access to the memory.

Processor '2' and '3' are requesting access for the same memory block '3'.

However processor '2' gains the access since its qdep(=2) is greater than qdep(=1) processor '3'. Processor '2' writes 'E' at location 'E'

The result observed in the simulation tracer 6.

The next stage is reading out the value written

Data out: x"1EFB"

Addr_bus: x"FE32"

Case 10:

Case where none of the processors are requesting access to the memory blocks.

ctrl:x"0"

The result observed on the simulation tracer 6.

There is no change in the value.

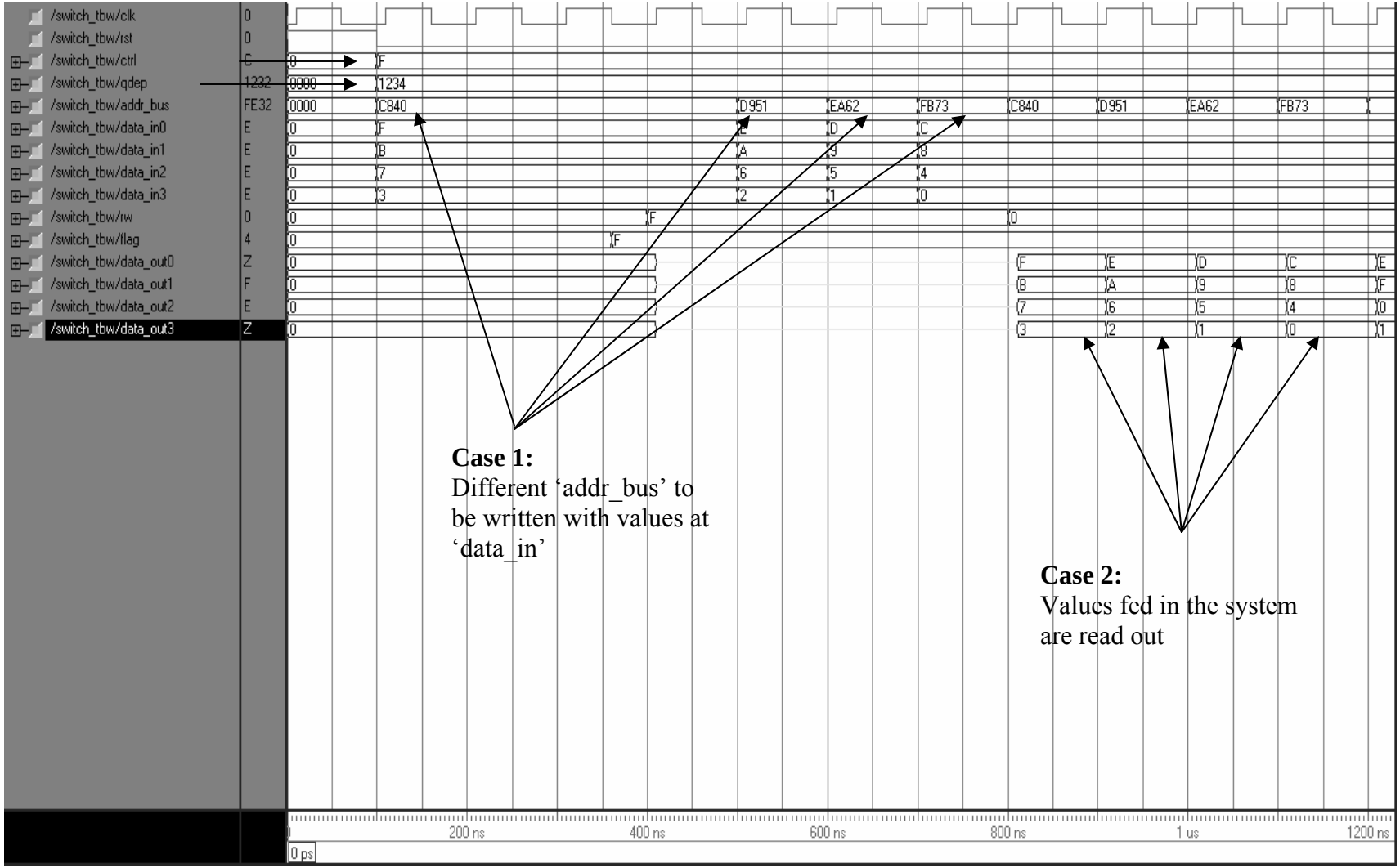

Figure 5.4, Simulation Tracer 4

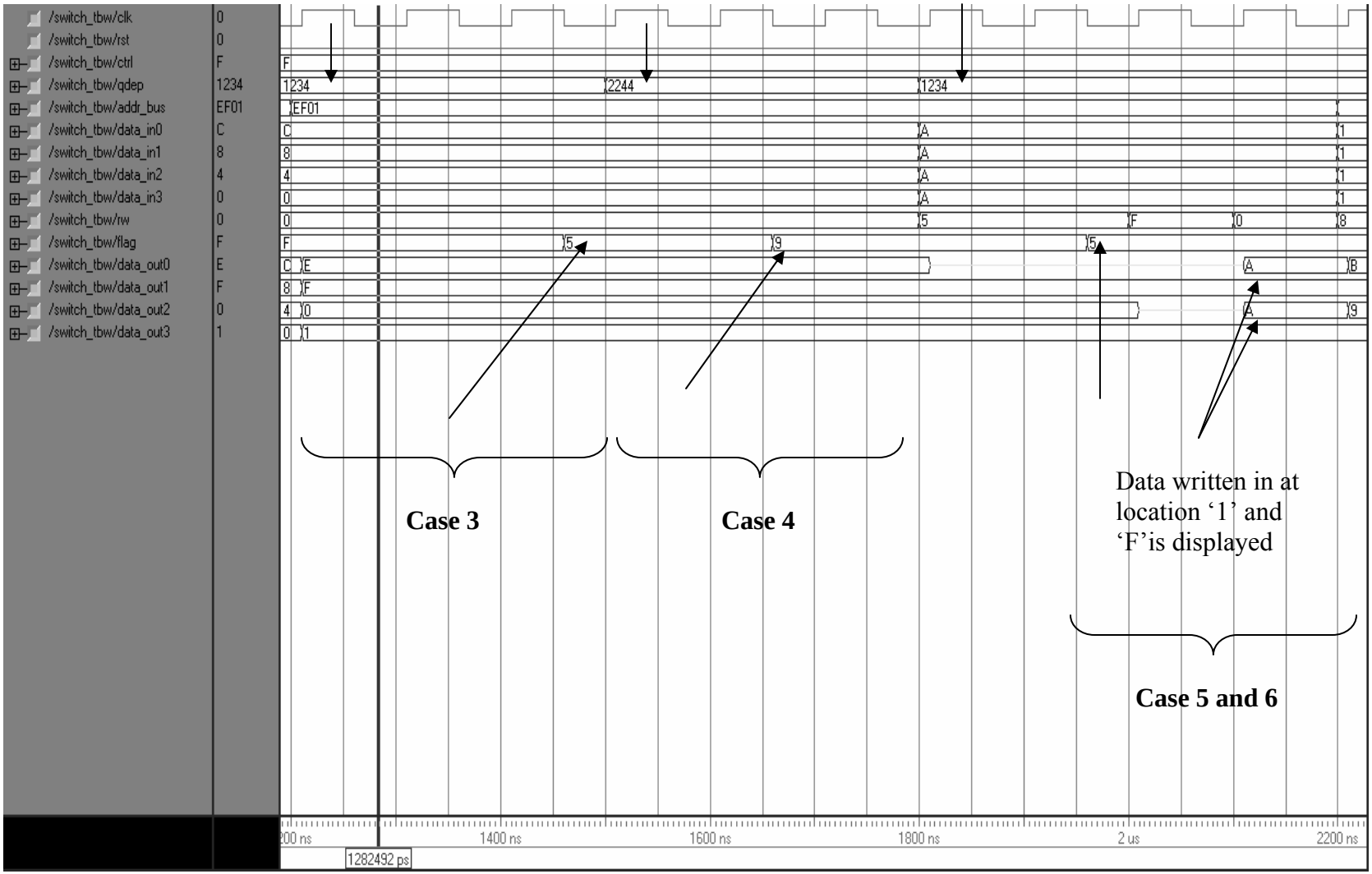

Figure 5.5, Simulation Tracer 5

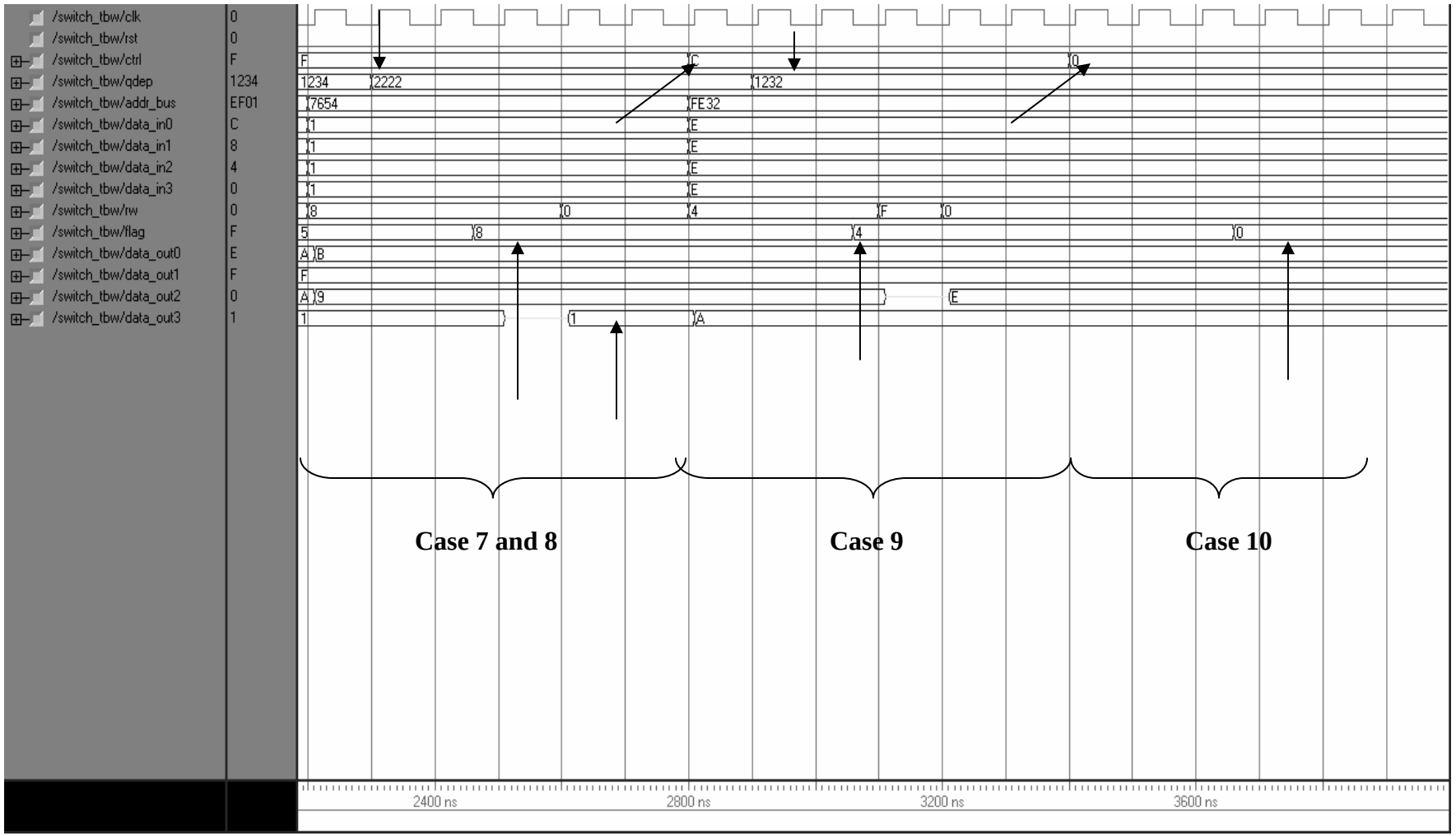

Figure 5.6, Simulation Tracer 6

A brief overview of the cases discussed in 5.1.1 and 5.1.2 can be obtained from the following graph in Figure 5.7 shown below.

The parameters which decide on the priority logic as described in section 5.1.1 and 5.1.2 are ‘r_w’, ‘qdep’, ‘ctrl’, ‘addr_blk’ and processor identification number. The chart is shown considering that all the processor’s ‘ctrl’ signal is at logic ‘1’ (request for access to the data memory), ‘r_w’ could be either a ‘0’or ‘1’ (processor could either read or write from the data memory). Hence the chart is plotted taking ‘qdep’, ‘addr_blk’ and processor identification number as the parameters. The processors which granted access are shown in solid color bars and those which are denied are transparent or clear.

At time ‘1’:

Processor 0 has queue depth of ‘4’and requests access to block 0(Blk0).

Processor 1 has queue depth of ‘3’and requests access to block 1(Blk1).

Processor 2 has queue depth of ‘2’and requests access to block 2(Blk3).

Processor 3 has queue depth of ‘1’and requests access to block 3(Blk3).

Since all the processors are accessing different memory blocks all of them are granted access as can be seen from the chart.

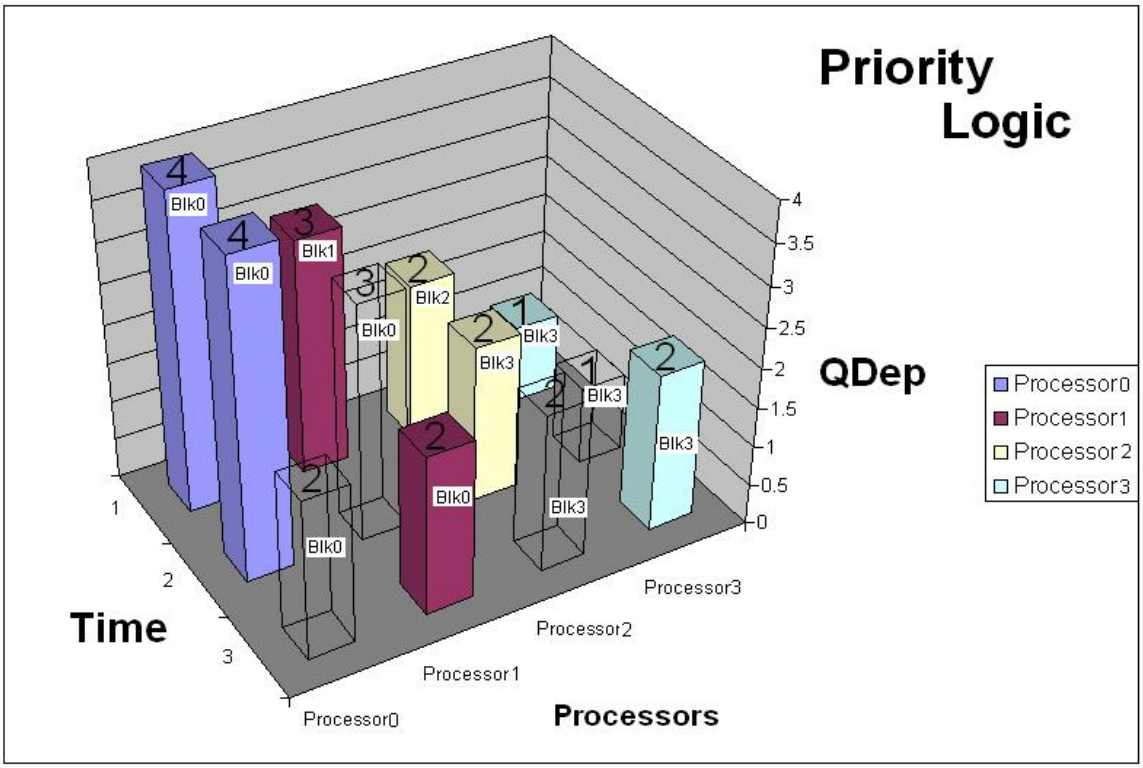

Figure 5.7, Brief Overview of Priority Logic for the Interconnection Switch.

At time ‘2’:

Processor 0 has queue depth of ‘4’and requests access to block 0(Blk0).

Processor 1 has queue depth of ‘3’and requests access to block 0(Blk0).

Processor 2 has queue depth of ‘2’and requests access to block 3(Blk3).

Processor 3 has queue depth of ‘1’and requests access to block 3(Blk3).

In this case as can be seen processor P0 and P1 are requesting access to the same memory block 0, however since the queue depth of processor P0 (qdep=4) is higher than processor P1(qdep=3), the processor P0 is granted access to block 0 and processor P1 is denied access. Similarly processor P2 and P3 are requesting access to the same memory block 3, the processor P2 is granted access to the memory block 3 since the queue depth of processor P2 (qdep=2) is higher than processor P3 (qdep=1).

At time ‘3’:

Processor 0 has queue depth of ‘4’and requests access to block 0(Blk0).

Processor 1 has queue depth of ‘4’and requests access to block 0(Blk0).

Processor 2 has queue depth of ‘2’and requests access to block 3(Blk3).

Processor 3 has queue depth of ‘2’and requests access to block 3(Blk3).

In this situation again processor P0 and P1 are requesting access to the same memory block 0, the queue depth for both the processors is the same (qdep=4). Hence according to the logic described in chapter 4, processor with higher identification number (Processor P1 in this case) is granted access. Processor P2 and Processor P3 are accessing the same memory block 3 and have same queue depth (qdep =2), hence similar to the situation discussed here for processor P0 and P1 the processor P3 is granted access since it has the highest processor identification number in this case.

5.1.5 Validation of crossbar switch via HDCA system

The crossbar switching network as described in Appendix A2 is embedded in the second phase “virtual” prototype of the HDCA system. The interfacing of the crossbar switch with the entire system involved major modifications to the first phase HDCA system [5] and some enhancements to the crossbar network in Appendix A2.

5.1.5.1 Changes and Enhancements to the First Phase Prototype

The base functional model of the first phase prototype [5] of the HDCA is unable to exhibit correct results at all times using Xilinx ISE 5.2i. On careful observation it is observed that small components which constitute towards the working of the entire system were not functioning correctly. In order to get the system to work as desired some major modifications and addition of some components had to be done to the first phase system. Due to frequent software related issues in the Xilinx ISE5.2i a decision was made to shift to the higher version, Xilinx ISE6.2.3i which is a much more stable version. In this section a brief description of the modifications made are listed component wise.

PE Controller

The following problems were noted in the first phase system [5]. In the state OP11 where in values are added immediately to the register value [7], the system failed to give out correct results. As a solution to this problem a multiplexer M5 is added to the existing Memory – Register Architecture [5]. This change allowed the output value of the Instruction Register IR0 to be connected directly to the register R3. The changes made have been separately shown in the Figure 5.8. Also the subtraction operation performed was giving incorrect results. The state OP5 was unable to give correct subtraction results. Appropriate changes are made to render correct results. These changes have been incorporated in the second phase system [7].

Another problem was encountered in the data input process into the ALU of the PE. The bidirectional data bus was directly connected to the ALU input which was a combinational logic, due to this unwanted data was also fed into the system along with correct values. Finally it resulted in an output with unwanted values and incorrect results. As a solution to this problem a register is introduced in front of the combinational logic circuit, the ALU. This change allowed only the correct values to enter the combinatorial logic and outputted correct results. The diagram with both the changes made is shown in Figure 5.9.

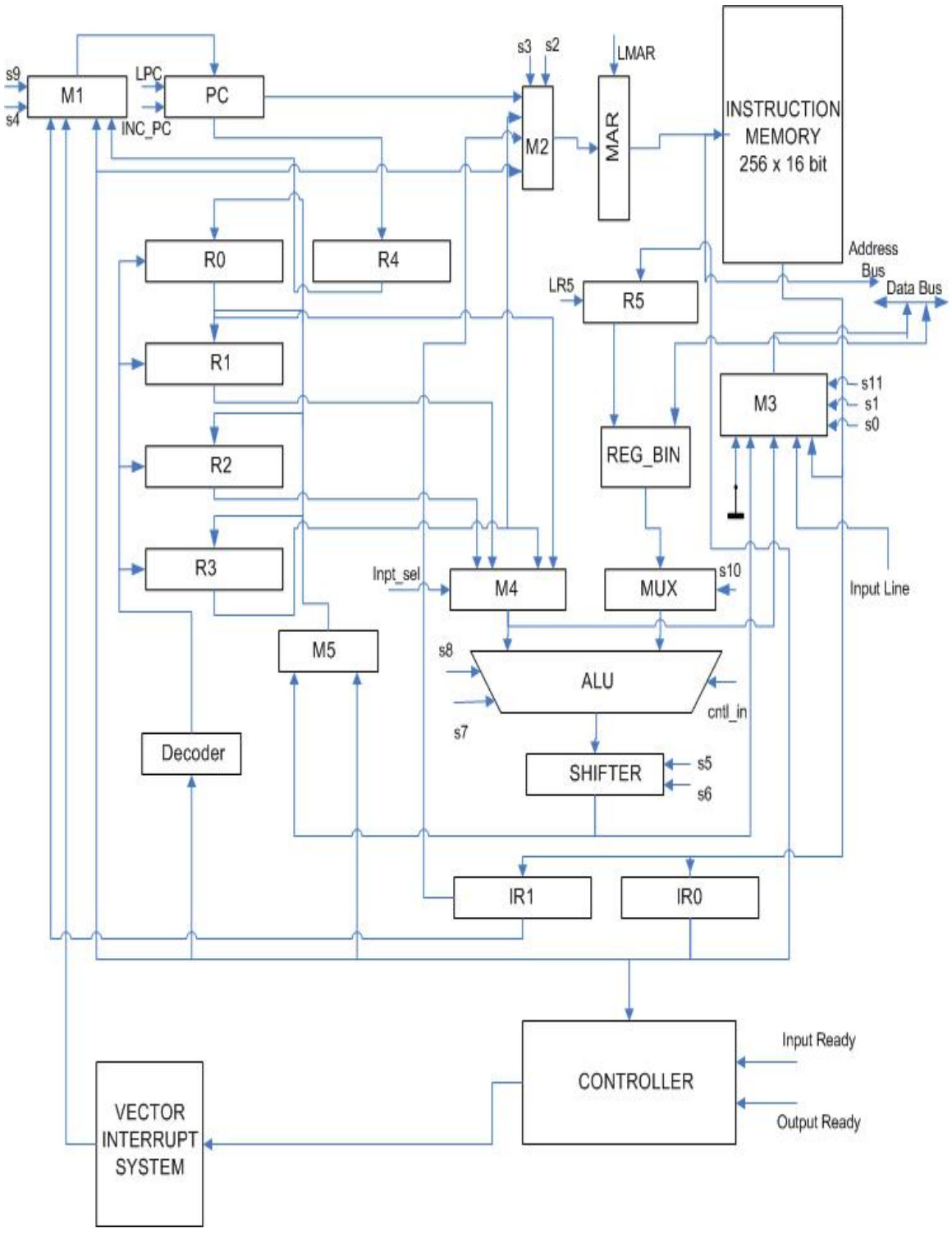

Figure 5.9, Memory/Register Architecture with Added Features

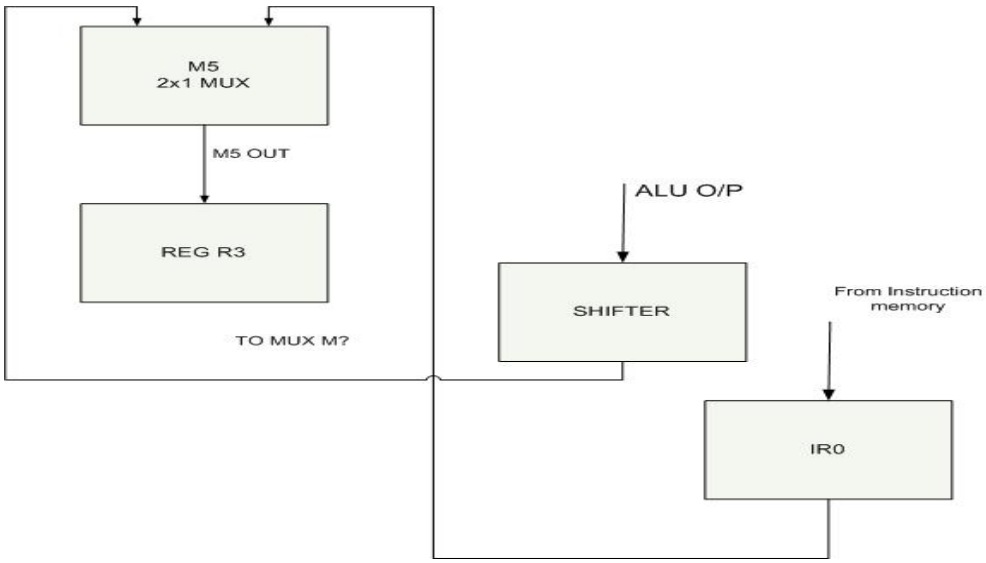

Figure 5.8, Changes Made to PE Controller- Additional Mux M5

Interface Controller

For pipelined execution of the applications running on HDCA the control logic module had to be modified. In the work done in [1982 paper], the HDCA system is capable of running multiple copies of the same application simultaneously on the system. Different copies of the same application are distinguished on the basis of the 'Time Stamp' field of the command token as shown below in Figure 5.6. For the entire set of token format refer [7]

Command Token

Hold Field	Physical Location	**Time Stamp**	Process Number	XXXXXXXX	Data Address
31	30 24	23 21	20 16	15 8	7 0

Figure: 5.10a Command Token Format of the HDCA System

An application developed in [5] is designed to meet the pipelined nature of the HDCA, however on running an application it gvives out incorrect results. On a closer examination of the behavior this discrepancy was due to the signal 'outbuf' in the control logic module. At the end of every process execution a 'Send PRT' and a 'StopL' token is being issued by the CEs which completed the process to the PRT Mapper. The signal 'outbuf' as mentioned above is being used in the formation of these tokens. With multiple

copies running on the system simultaneously, it was observed that the value of first copy of 'outbuf' was getting overwritten by the consecutive copy of 'outbuf' before the value is used. This resulted in loss of data for the first application and hence abrupt termination of the first application. The code for the formation of 'Send PRT' and 'StopL' tokens can be obtained from Appendix A of [5].

A major modification had to be introduced in the system to fix this particular problem. Logic is established to differentiate among the command tokens with the help of the 'Time Stamp' field of the command token in Figure 5.6. An array like structure is introduced to store the data required in the formation of the 'Send PRT' and 'StopL' tokens. The command token format is also changed in the wake of this change. The bits from 15 down to 8 in the token which were don't cares(X) are changed to all '1s'. Along with this change a new process 'get_data' is introduced in the control logic module in order to parse the command tokens appropriately. The new format of the command token is shown in the Figure 5.7

Command Token

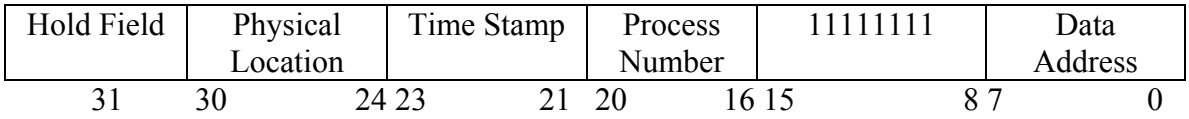

Figure 5.10b: New Token Format for the Command Token of the HDCA

As can be seen from the format above the 'Time Stamp' field is 3 bits wide allowing 8 copies of the same application to run simultaneously on the system.

Another capability of this system as discussed in [3] is the 'Dynamic Node Level Re-configurability' where in during the execution of an application the queue builds up and on reaching a set threshold value, a new CE is configured on the fly to take up the extra load of the overloaded CE. One of the problems faced while testing this property was loss of command tokens. The system failed to parse them, resulting in a sudden termination of the entire copy of the corresponding application. A new delay state is added to the control logic module.

Dynamic Load Balancing Circuit

As a part of verifying the correct functioning of the HDCA system, numerous applications are run on it. While running one of the applications as shown in Figure 5.11, it was observed that the join operation of the processes P2 and P3 to process P4 was showing incorrect results.

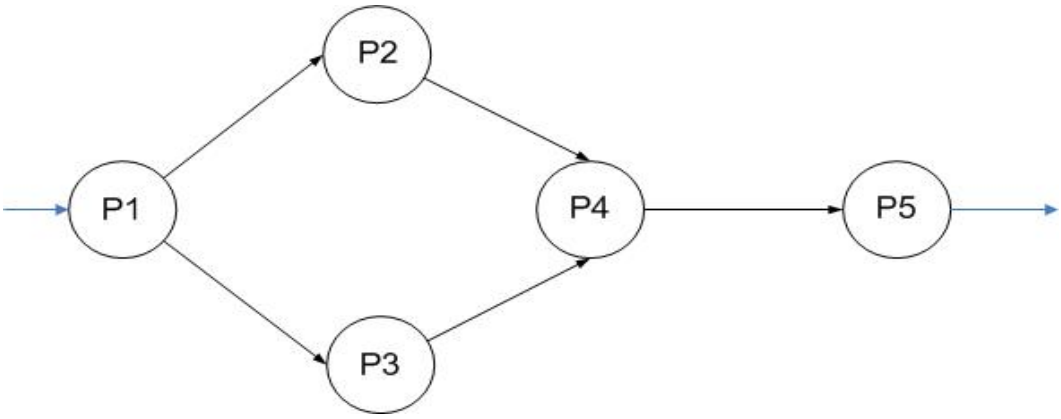

Figure 5.11, An Example of a Process Flow Graph

On close scrutiny the problem was found in the 'dynamic load balancing circuit' module. It can be seen from the Figure above that during the join operation of P2 and P3, the register R6 as shown in Figure 2.4 of [7], is utilized to store values of the Physical Location, Process Number and the Data Location. This register is used specifically for the join operation; it stores the values of Process Number and Physical Location of the current process, P2 in this case. These values are used by the consecutive process, P3 in this case so that they map to the same resultant process P4 here. The problem in this join operation was that the system failed to understand that the operation is a join operation owing to the fact that the values stored in R6 were not being assigned to the next process. The change is documented in the Appendix A2.

Input ROM

An input ROM is designed and introduced in the HDCA system in order to input data values on which the application runs. The data is requested into the system in a particular fashion and hence a core generated module could not be used. It is observed that the data is requested every third clock cycle by the system, to facilitate transfer of

values from ROM a special signal ‘valid’ as output. Only the values that are output every third clock cycle are considered to be valid and are sent inside the system.

Multiplier CE

This CE is another important addition to the HDCA system enhancing its heterogeneity. Multiplication forms a major part in the applications for Digital Signal Processing and the like; earlier the applications running on the application were limited because of the lack of multiplier CE. Therefore a new Multiplier CE is developed and integrated along with the other CEs. There are various ways in which a multiplier can be designed, a few of them are Booth’s algorithm and core generated multiplier. After having carried out a survey of multipliers it is observed that multipliers designed using Booth’s algorithm consumed more power as compared to the core generated multipliers. Since the factor of consumption of low power was of prime importance a design decision was made to use the multiplier as shown in the Appendix A2. As can be seen the multiplier is not a core generated, however on synthesis it is clear that the system has utilized the core generated multiplier to infer the coded multiplier.

Crossbar Interconnect Network

As described in the chapter 4, in which the crossbar switch is found to be the best interconnection for the HDCA system, it is introduced and integrated with the architecture. This is a step forward in the effort to make the HDCA system more scalable and effective. One major problem faced in the first phase of the system was that of bus contention in the light of two or more processors trying to access the data memory. This issue is addressed effectively by the addition of the crossbar interconnect network. The following section consists of two applications run on the HDCA system with the crossbar embedded into the system. A design decision was made to use the component level crossbar network explained in chapter 4. It should be noted here that depending on which CE accesses the memory block (refer Chapter 4) the values stored in the memory can be viewed at the ports “mem_out_0, mem_out_1, mem_out_2, mem_out_3” for processes executed by CE0,CE1,CE2 and CE4 respectively. Similar argument holds good for their

respective address locations which can be observed at “mem_ad_out_0…3”(4 different ports). These ports are shown in all the following Figures.

5.1.6 Application 1 Described with Acyclic Process Flow Graph

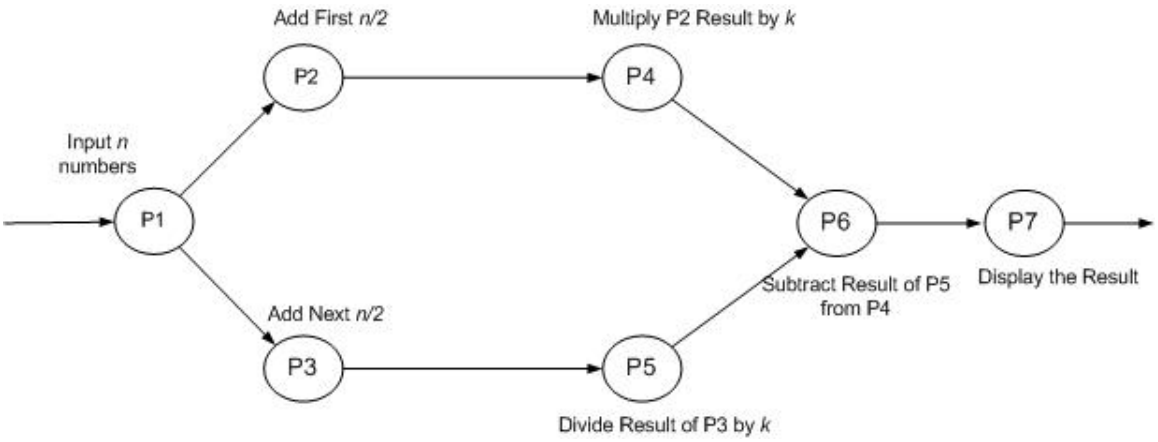

Figure 5.12, Process Flow Graph for Application 1

Each process can be described in more detail as follows:

P1 – Input of *‘n’* numbers into the Shared Data Memory from the InROM.

P2 - Add First *‘n/2’* numbers inputted and store the result in Data Memory.

P3 – Add the next *‘n/2’* numbers inputted and similar to P2 store the value in data memory.

P4 – Multiply the result of P2 by value *‘k’* stored in the instruction memory of the Multiplier CE.

P5 – Divide the result of P3 by value *‘k’* stored in the instruction memory of the Divider CE

P6 – Subtract the result of P5 (Multiplication) from P4 (Division) and store the value in the data memory.

P7 – Output the final value of the result obtained in process P6.

The process flow graph described is tested by running two copies of the same application, giving two command tokens. The initialization tokens and the instruction sets are described in detail in Appendix B. For the application described the number of values inputted ‘n’ is assumed to be 10(unsigned) and ‘k’ is assumed to be 2(unsigned). The

command tokens are inputted after the initialization tokens. The two command tokens given are: x“0101FF03” with time stamp as “000” and x“0121FF11” with time stamp as “001”. They address the PRT mapper to map the first process P1 for both the copies. Here since none of the CEs are being used the PRT mapper allocates CE0 for both the copies of P1, as CE0 has a higher priority over CE1 on process P1.Figure 5.13 shows the two command tokens being issued to the CE0.

The first instruction issued by the interface controller of each CE is shown in the waveforms at the ports “db_pe_icm0_fin0 for CE0 and similarly for other CEs. These ports taken out are from the connection signal between individual CE and its Controller as shown in Figure 1.1. CE0 begins execution of process P1, it inputs 10 values from the input bus ‘inpt_data0’ into the shared data memory, the Figure 5.14 shows the inputting of first 5 values(all 2sin this case) into the data locations starting from x”03”of shared memory block. The Figure also indicates CEs accessing, a particular block in the memory, for instance in this case CE0 is accessing block ‘blk0’ (refer to the last two signals of the waveform in Figure 5.14). The inputting of the remaining 5 values is shown in Figure 5.15. It can be seen that the ‘rq_ipt0’ goes high whenever CE requests a value to be entered in a particular location. Consequently the ‘idv0’signal is made high, to allow the inputting of values. The values stored in memory can be viewed at the “mem_out_0”.

After the process P1 is executed, as seen in process flow graph (Figure 5.12) process P1 forks to two processes, resulting in two command tokens (x“01020003” and x“01030003”) being issued to the PRT mapper. This is depicted in Figure 5.16

PRT Mapper chooses the most available CE and allocates the processes. In this case the process P2 is mapped to CE0 since it is the most available and process P3 is mapped to CE1. The reason P3 is allocated to CE1 is that this CE is made the most available for process P3 by changing ‘ram address’ field in Load PRT Mapper Token [7]. It should be noted here that this value could be changed to get different results; this application is run as per values in Appendix B. The Figure 5.17 shows the command tokens being issued to the CEs, process P2 to CE0 (x“0302FF03”) and P3 to CE1 (x“0203FF03”). At the end of process P1 for the copy 1 of the application, the execution of process P1 for 2nd copy of the application begins, the instruction is issued by the CE0

(“9C11 3003”). In the meantime CE1 starts execution of the process P3 for the first copy (“9C03 3024”). In this case CE0 and CE1 are accessing the same memory block ‘blk0’, however not at the same time hence there are no conflicts. All these developments can be seen in the Figure 5.18.

At the end of the execution of process P1 of copy 2, two command tokens are generated by CE0 and issued to the PRT mapper, similar to copy1. It is shown in Figure 5.19. The PRT Mapper makes a selection of the most available CE and allocates the processes to the particular CEs. Figure 5.20 shows that the PRT Mapper allocates P2 to CE0 and P3 to CE1 giving out the tokens x“0323FF11” and x“0223FF11”.

The execution of process P2 of copy 1 begins after the end of process P1 of copy 2. The instruction x“9C03 3017” is being issued to the CE0. This is shown in Figure 5.21. In Figure 5.22 it can be seen that the instruction for process P3 is issued by CE1 x“9C11 3024” and also a command token x“ 01050003” for the process P5 is being issued to the PRT Mapper as the execution of process P3 ends. The process P5 is a division operation as shown in Figure 5.12, the PRT Mapper allocates the process P5 to the Divider CE as can be seen from the token x“0405FF03”. The stored data values in the data memory, the result of the addition of first 5 and the last 5 values before and after the division and multiplication results and final results are shown in the appendix B.

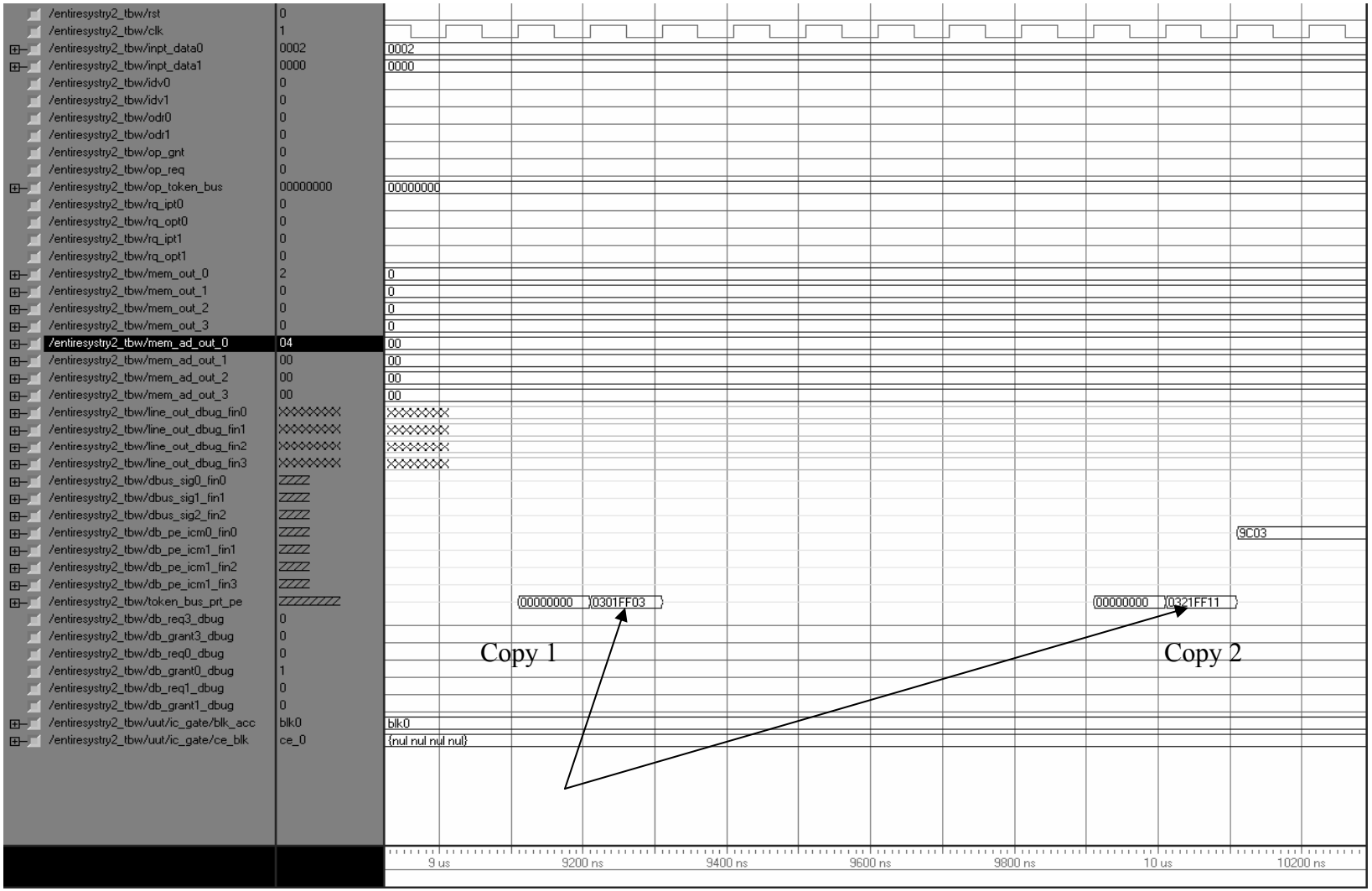

Figure 5.13, Command Tokens for Both Copies of P1 to CE0 Issued by PRT Mapper

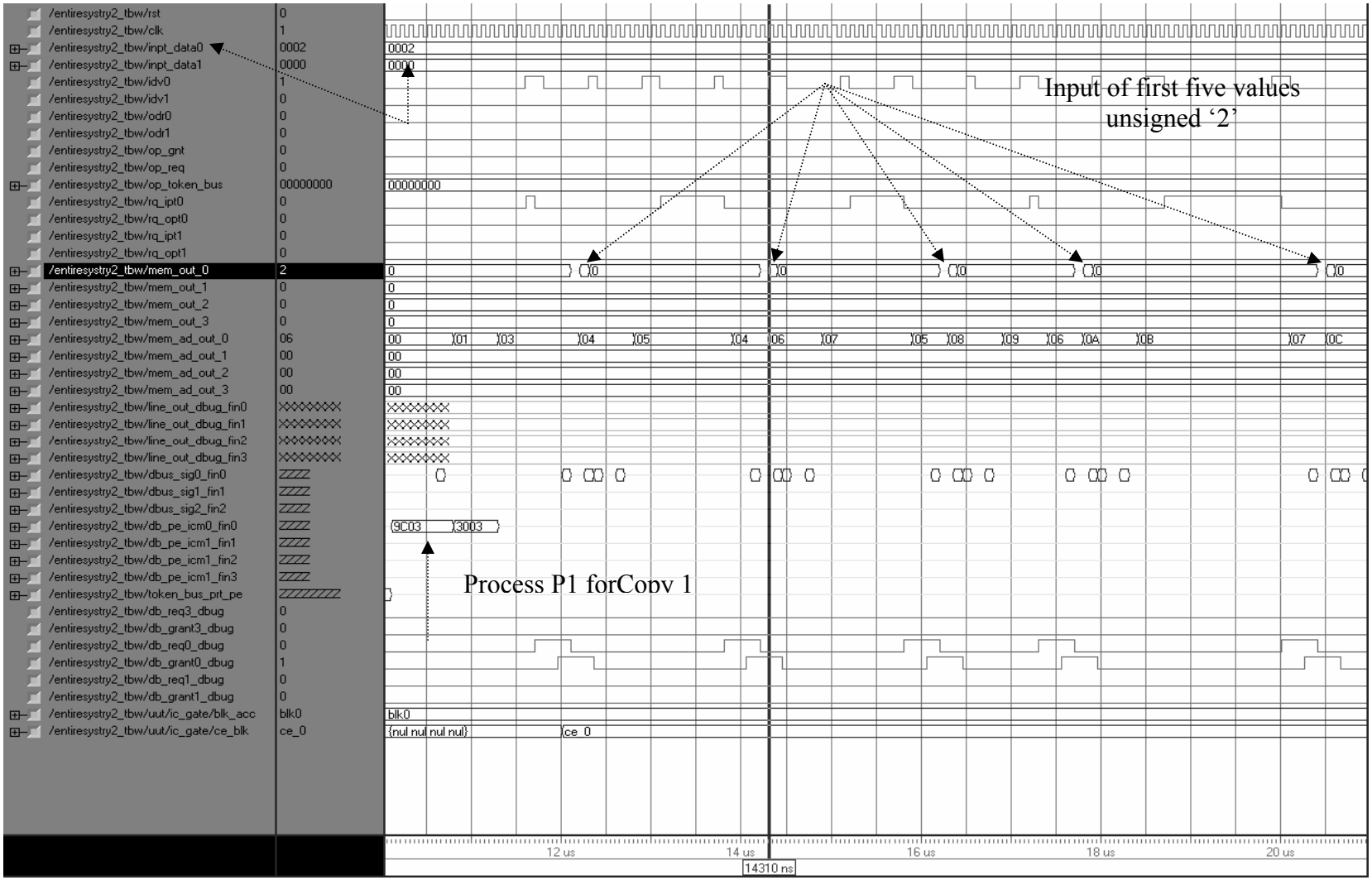

Figure 5.14, First Instruction and Input of First Five Values.

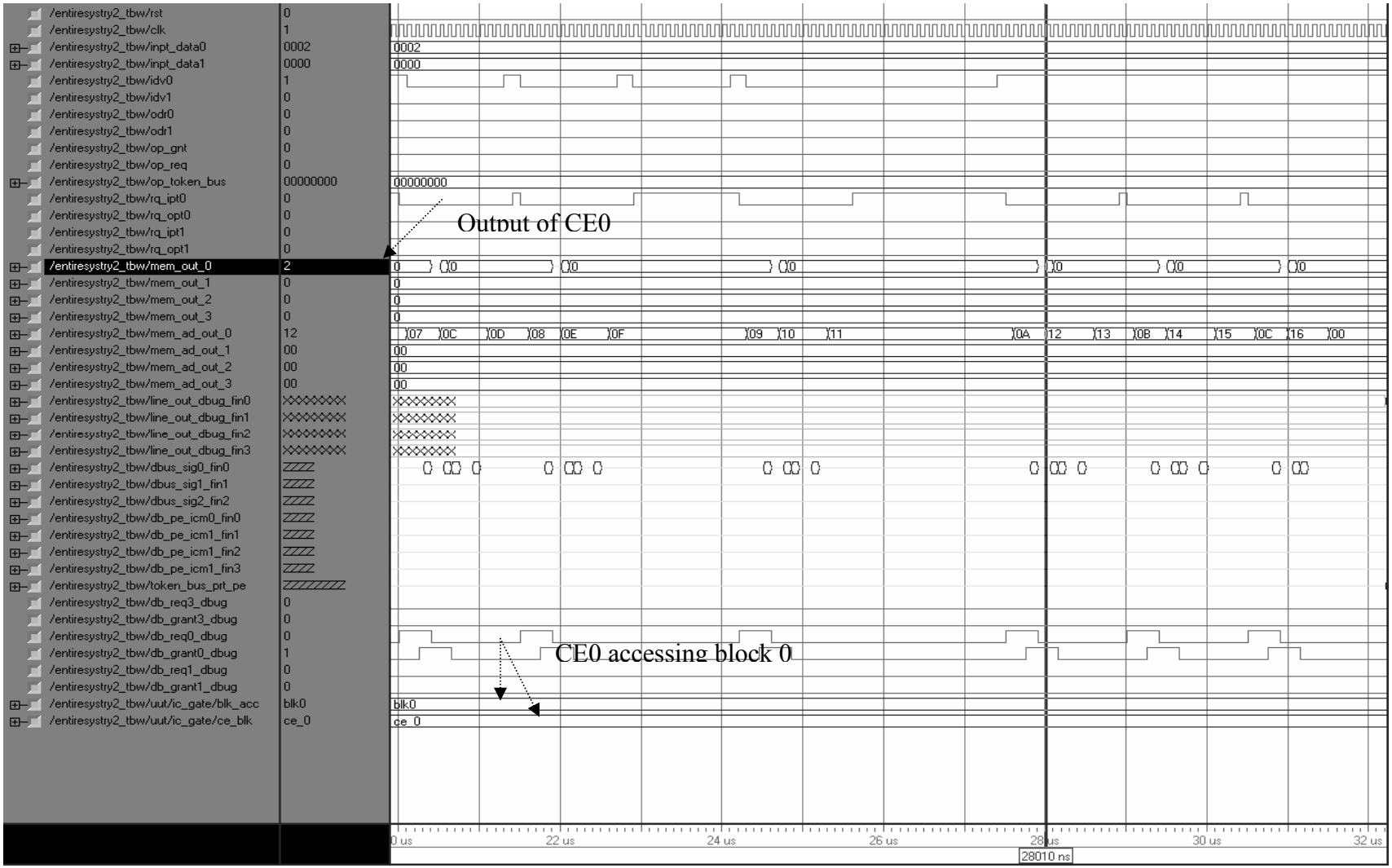

Figure 5.15, Input of Last 5 Values for Process P1 of copy 1

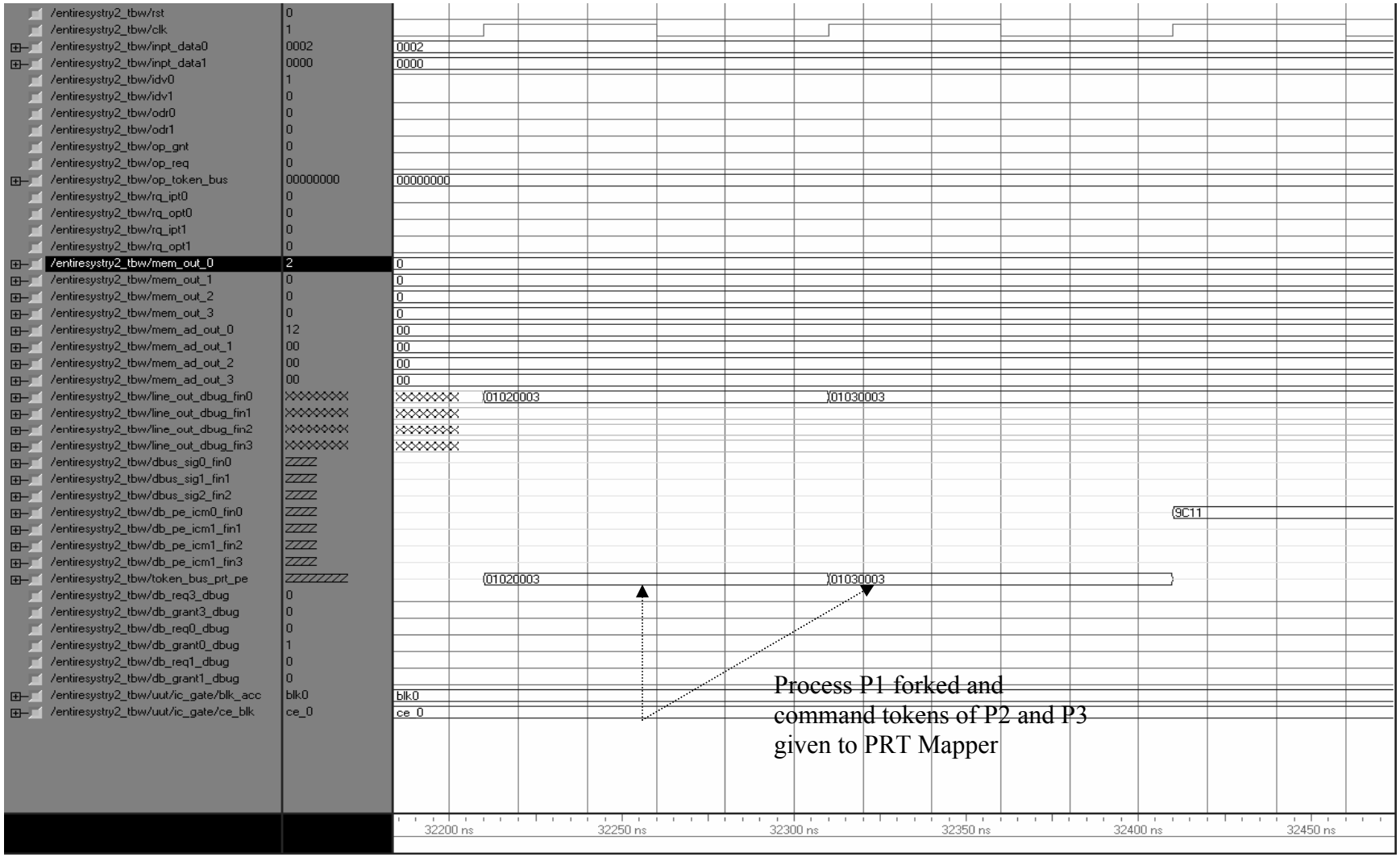

Figure 5.16, Two Command Tokens Issued to PRT Mapper of Copy 1

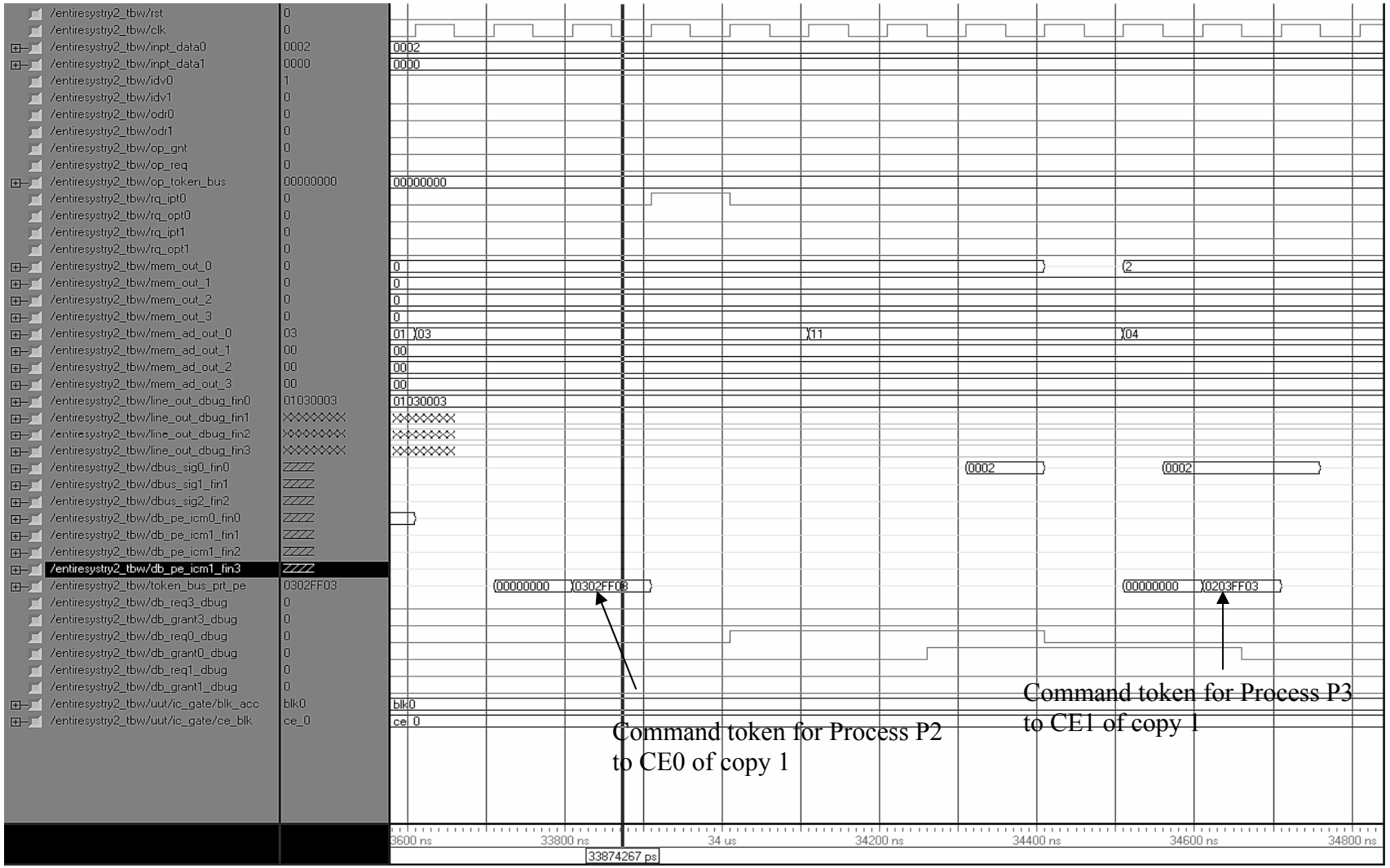

Figure 5.17, Command Tokens Issued to CE0 and CE1 by PRT Mapper of Copy 1

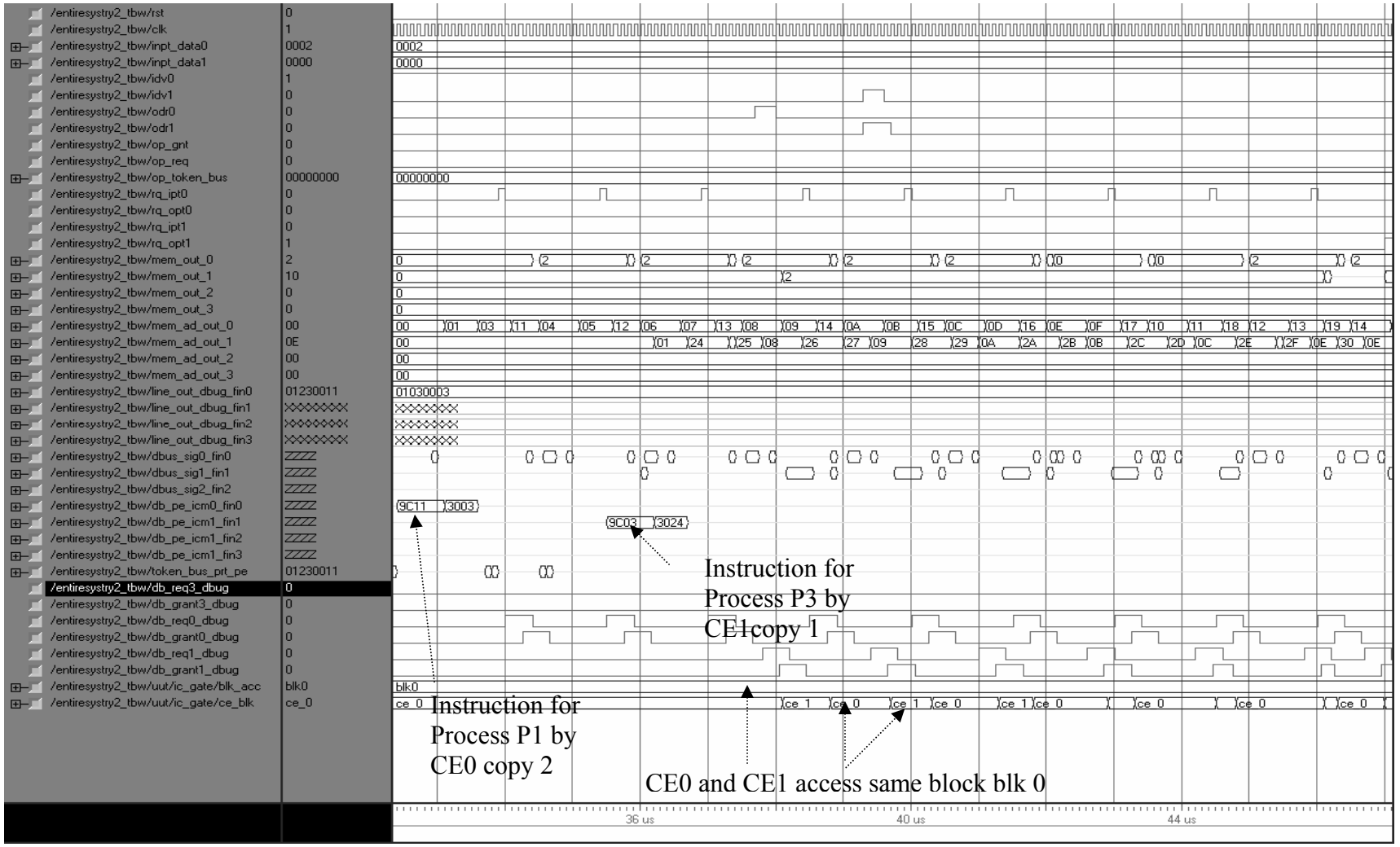

Figure 5.18, Instructions for Process P1 of Copy 2 and for Process P3 of Copy 1

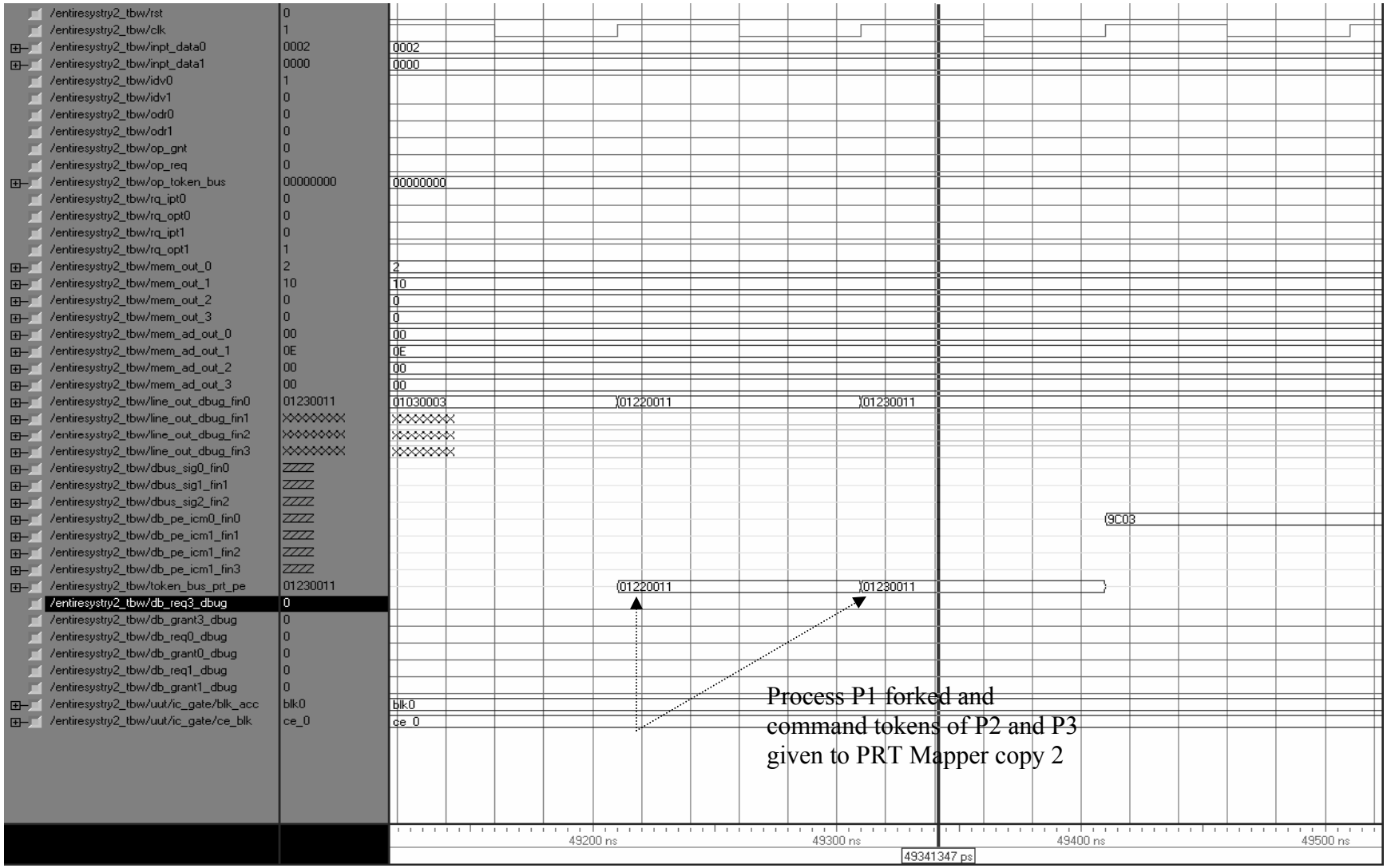

Figure 5.19, Two Command Tokens Issued to PRT Mapper for Copy 2

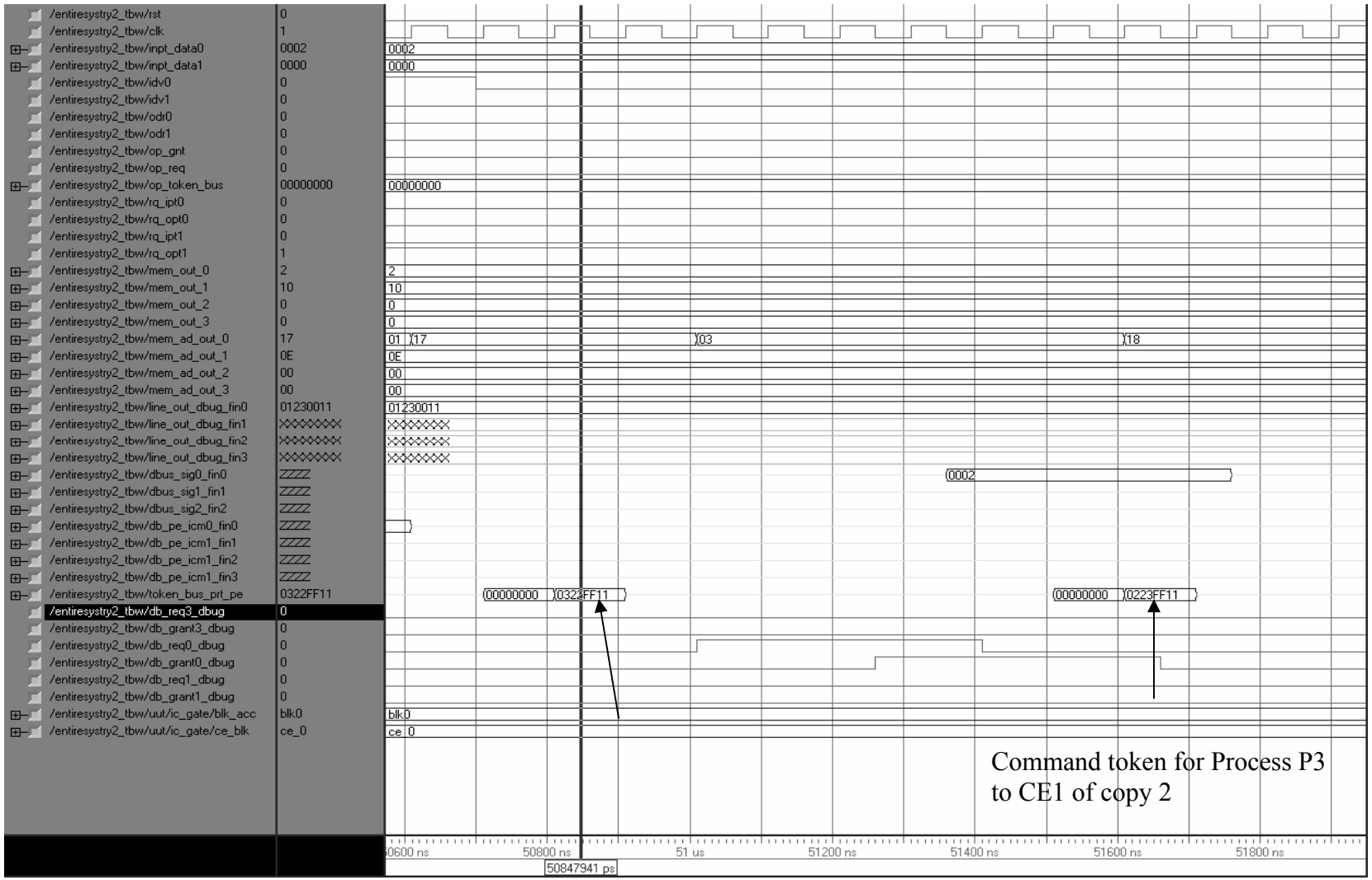

Figure 5.20, Two Command Tokens Issued to CEs by PRT Mapper for Copy 2

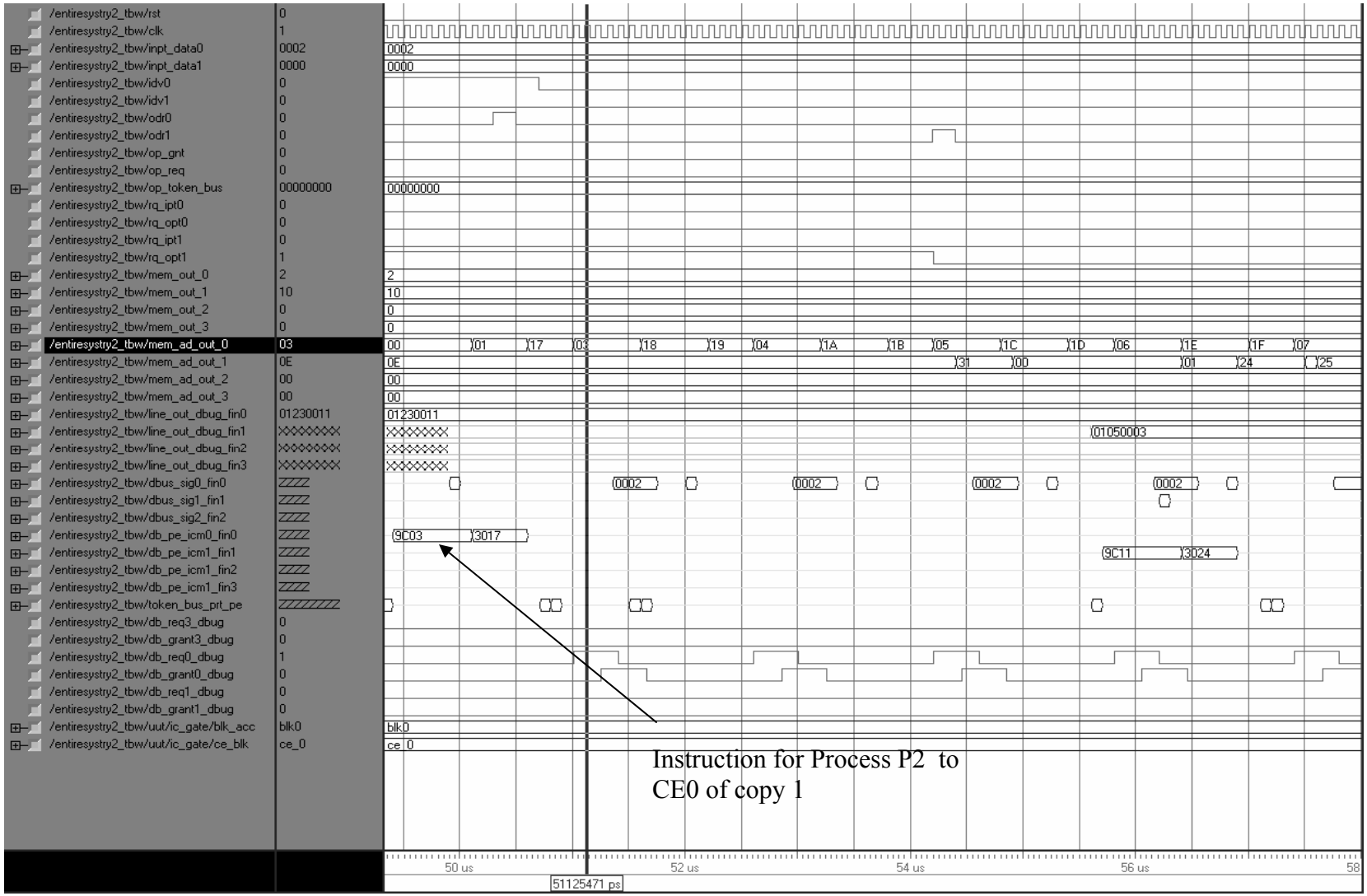

Figure 5.21, Instruction Issued by CE0 of Process P2 for Copy 1

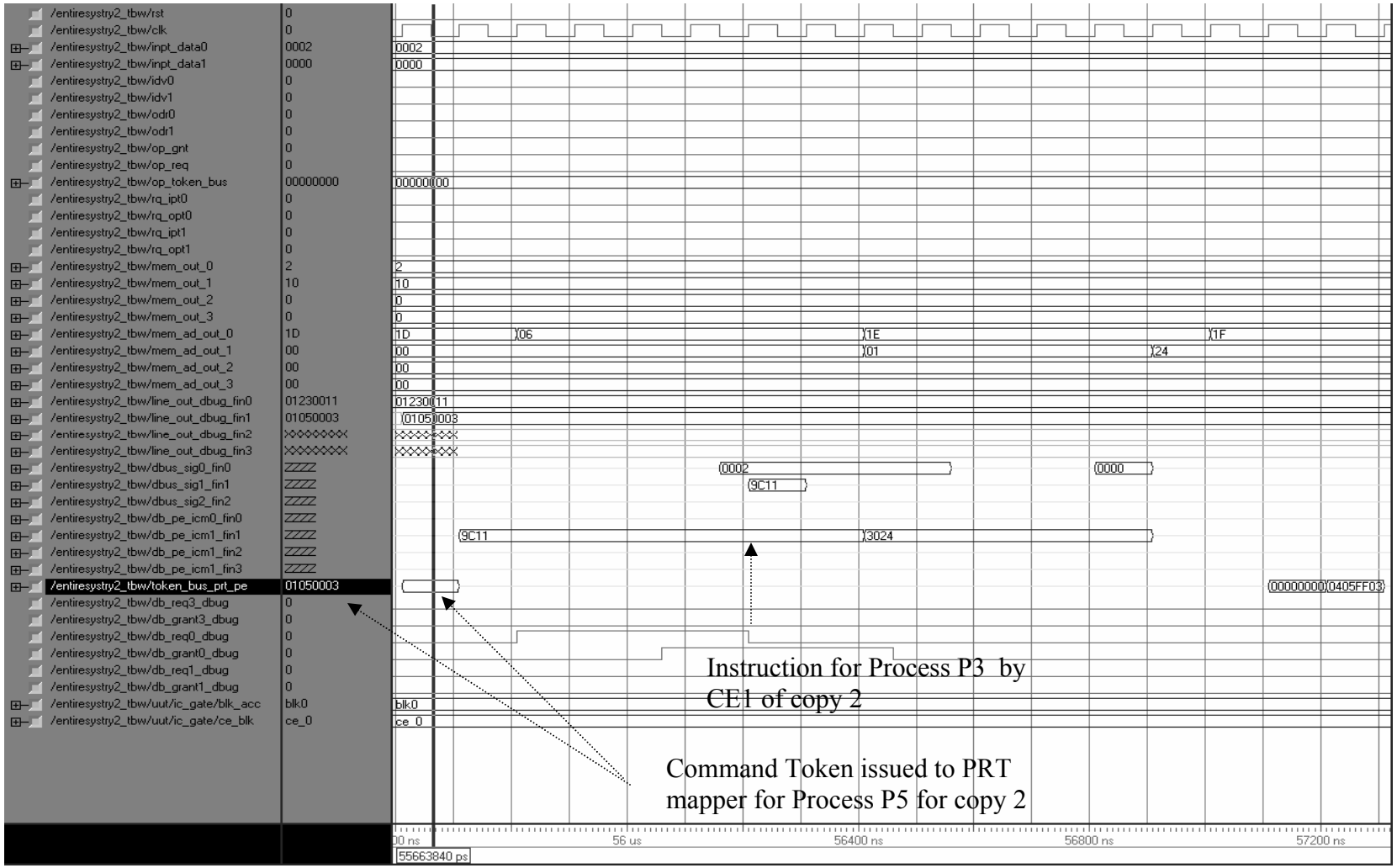

Figure 5.22, Instruction by CE1 of P3 Copy 2 and Commend Token from PRT to CE2

The division operation is shown in the Figure 5.23, the division of value unsigned‘10’ (result of addition of last 5 values of process P3) at x”0E”by unsigned’2’, the result unsigned ‘5’ is obtained after a 20 clock cycle delay and is stored at the same location of ‘10’ that is x”0E”. To exhibit perfect division operation the ports of the divider CE are taken out and shown in the lower part of the same Figure 5.23. At the end of the execution the Divider CE sends the command token to the PRT Mapper x”81060003” which signifies that it is a join operation. The PRT Mapper waits for the P4 process to execute and issue similar token to the PRT Mapper.

After the execution of process P2 by CE0 it sends the command token x”01040003” to the PRT Mapper. The next process is P4, multiplication operation; hence the PRT Mapper allocates the process to the Multiplier CE (CE4). It issues a command token x”0504FF03” to CE4. This is shown in Figure 5.24. The Figure also shows the issuance of instruction for process P3 for copy 2 to CE0 “db_pe_icm_0_fin0”.

Figure 5.25 elaborately shows the multiplication operation after the issue of the multiplier instruction. The value unsigned ‘10’ stored at location x”0D” (addition of first 5 values in process P2) is multiplied by unsigned ‘2’. The result, unsigned ‘5’ is stored at same location x”0D”. These values can be seen at port “mem_out_3” and locations at “mem_ad_out_3” of the waveform. The ports of the multiplier CE are taken out and shown in the waveform of Figure 5.25 to further exhibit the functioning of the multiplier CE. At the end of process P4 a command token is being issued by CE4 to the PRT Mapper x”81060003” which indicates a join process P6 similar to the Division process explained earlier.

The process P2 of copy 2 after execution sends a command token for process P5 to PRT Mapper and eventually the PRT Mapper sends the command token to the Divider CE. This is shown in Figure 5.26. The detail division operation is shown in Figure 5.27.Here the unsigned value ‘10’ stored at x”1C”, refer to Appendix B is divided by unsigned value of ‘2’. The result is stored in the same location x”1C” as shown in the waveform of Figure 5.27.Values observed at “mem_out_2”at location “mem_ad_out_2”. Similar to the division operation of copy 1 the divider ports are taken out to confirm perfect functioning of the divider CE. The following command token for process P6 is issued by CE2 to PRT mapper.

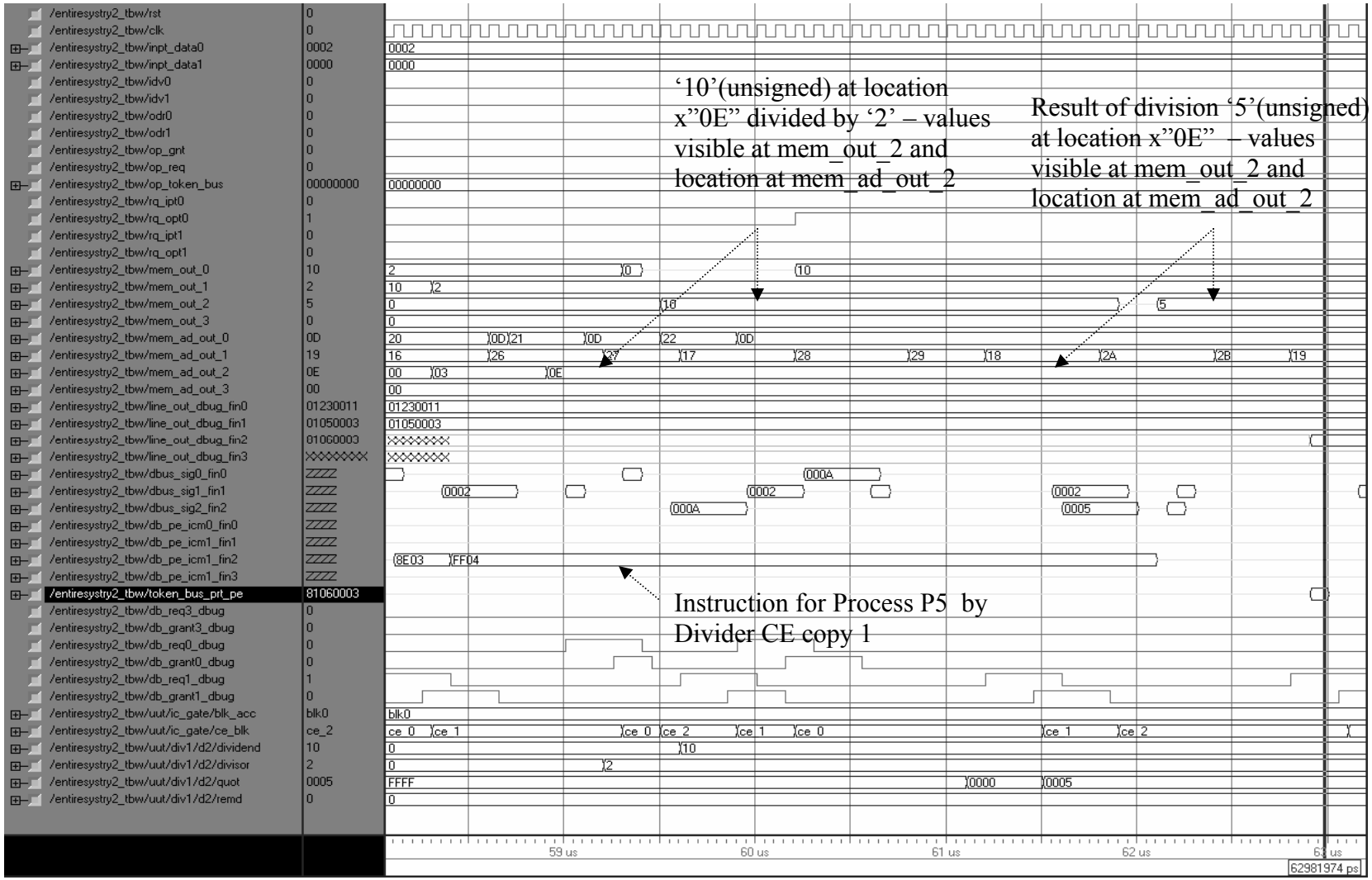

Figure 5.23, Division Op. for P5 with Results, Command Token to PRT Mapper Copy 1

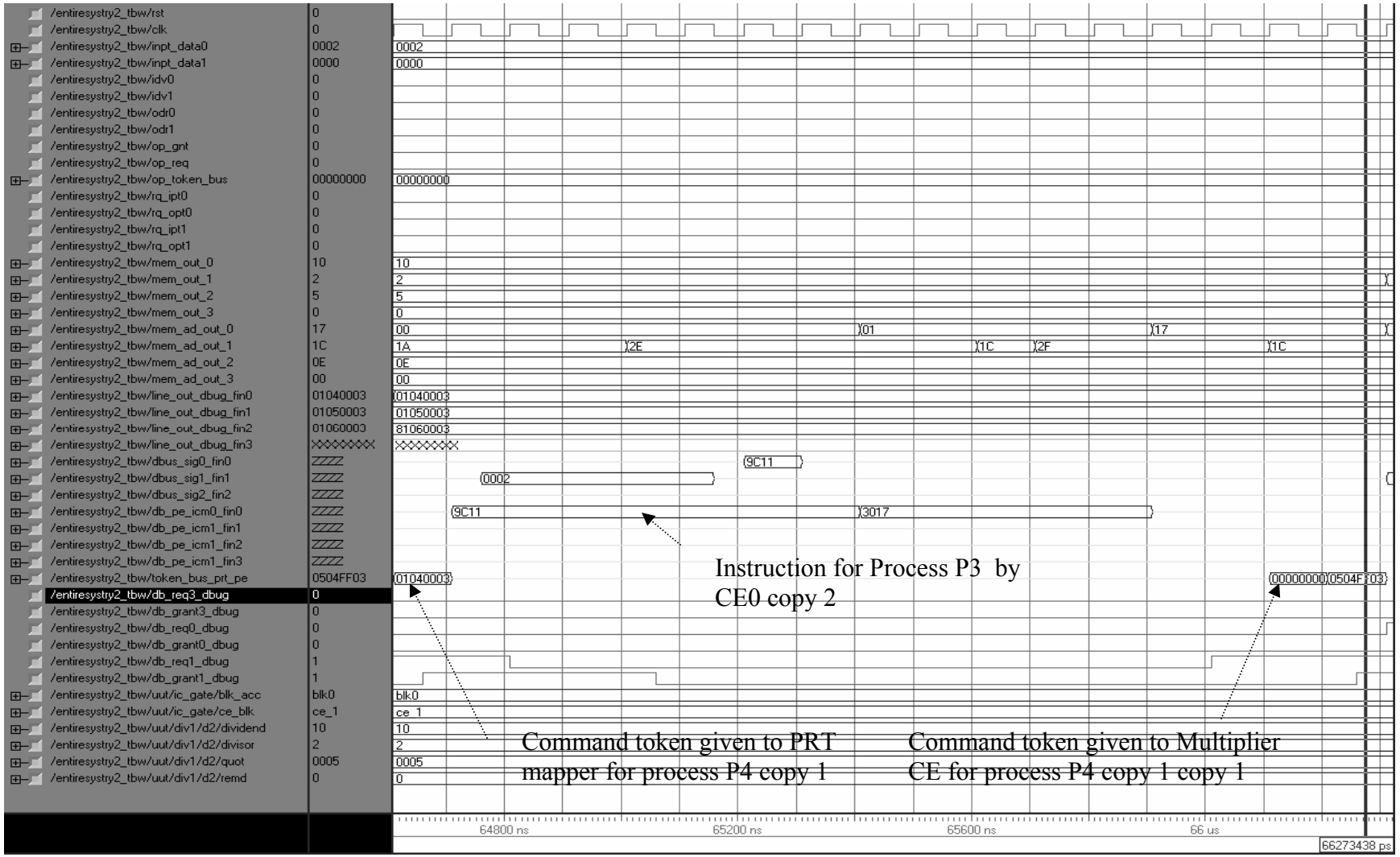

Figure 5.24, Command Tokens for P4 to PRT Mapper, from PRT to CE4 for Copy 1

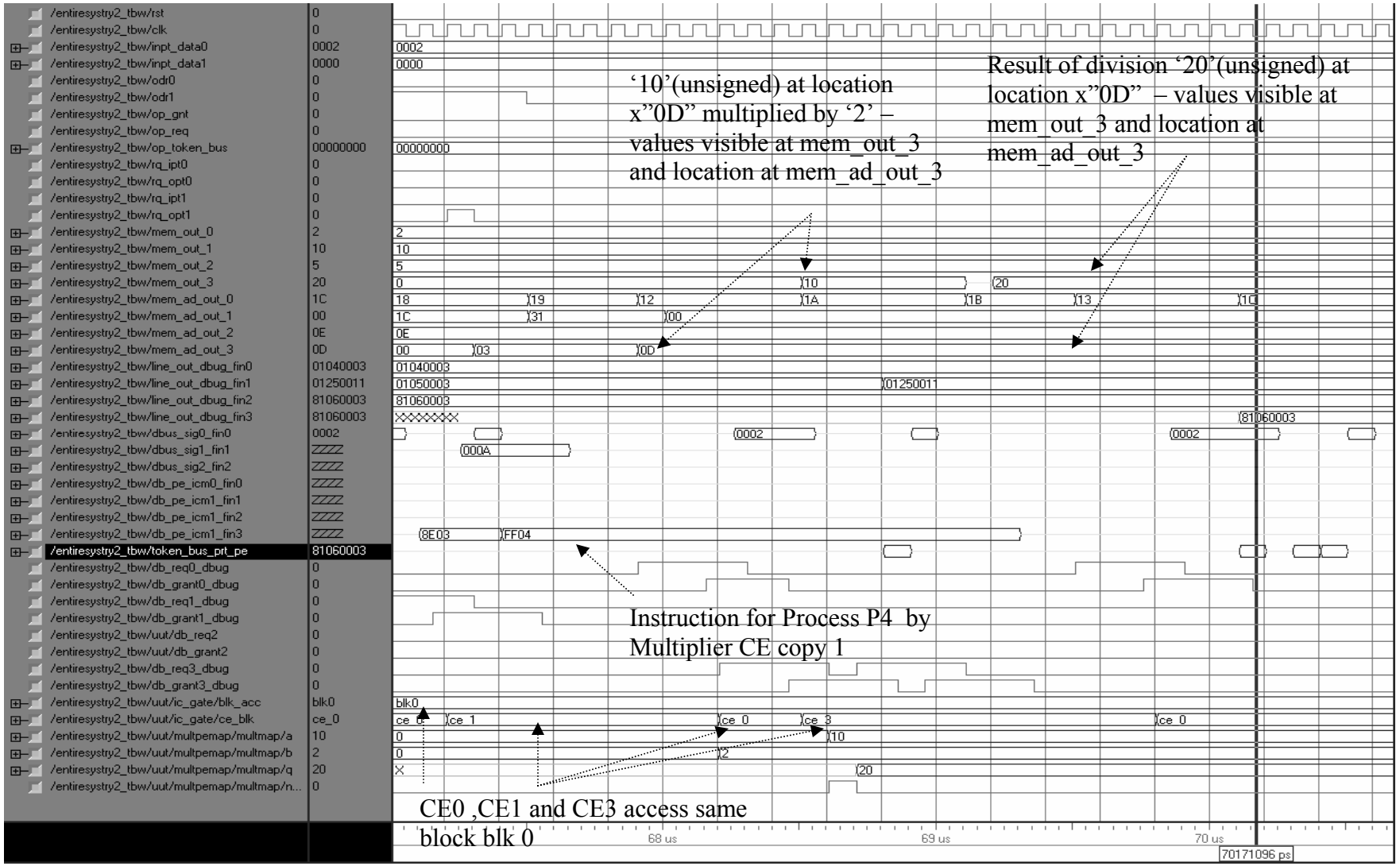

Figure 5.25, Multiplication Operation by CE4, Token Issued to PRT Mapper Copy 1

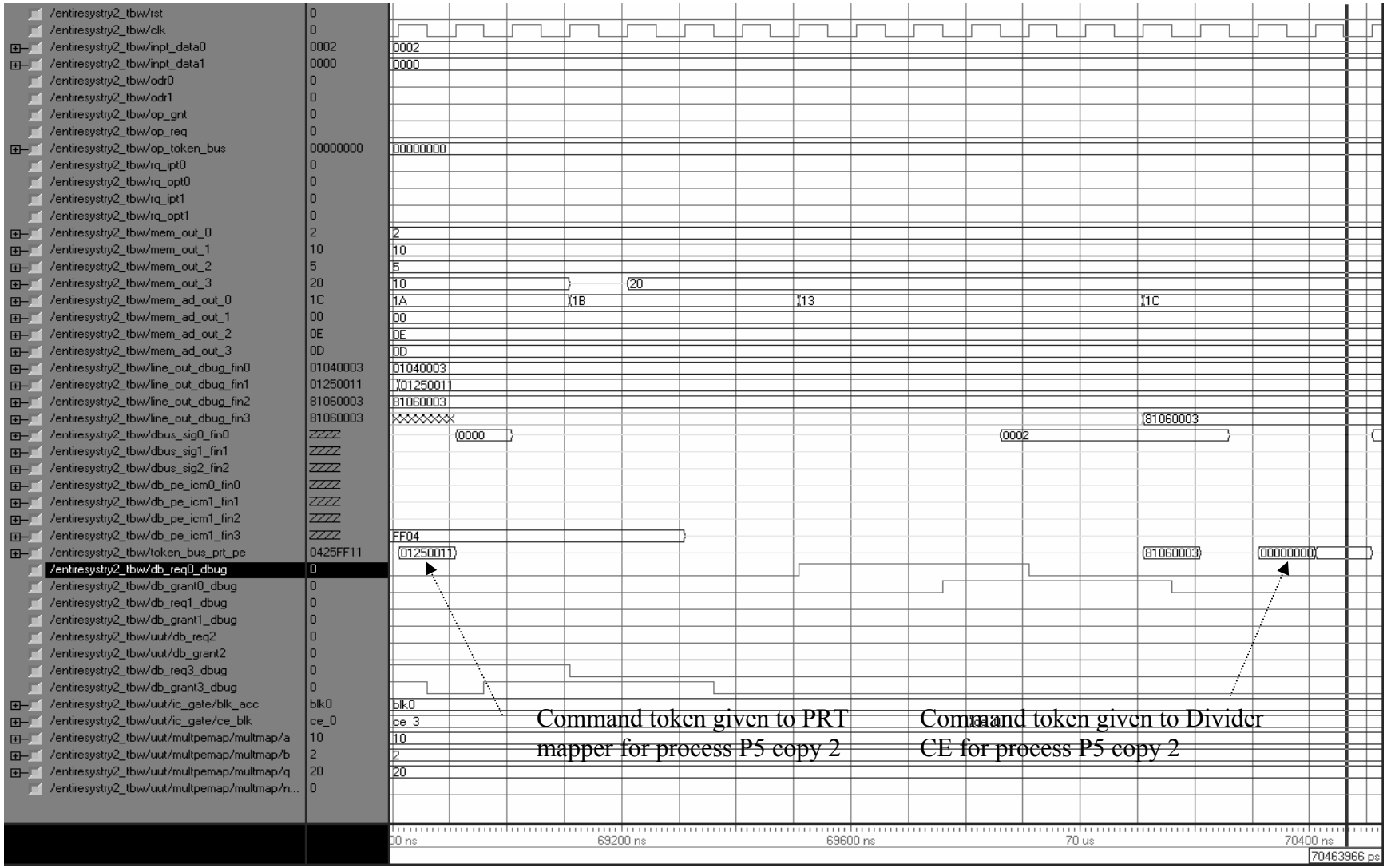

Figure 5.26, Command Token for P5 Issued to PRT Mapper, from PRT to CE2 Copy 2

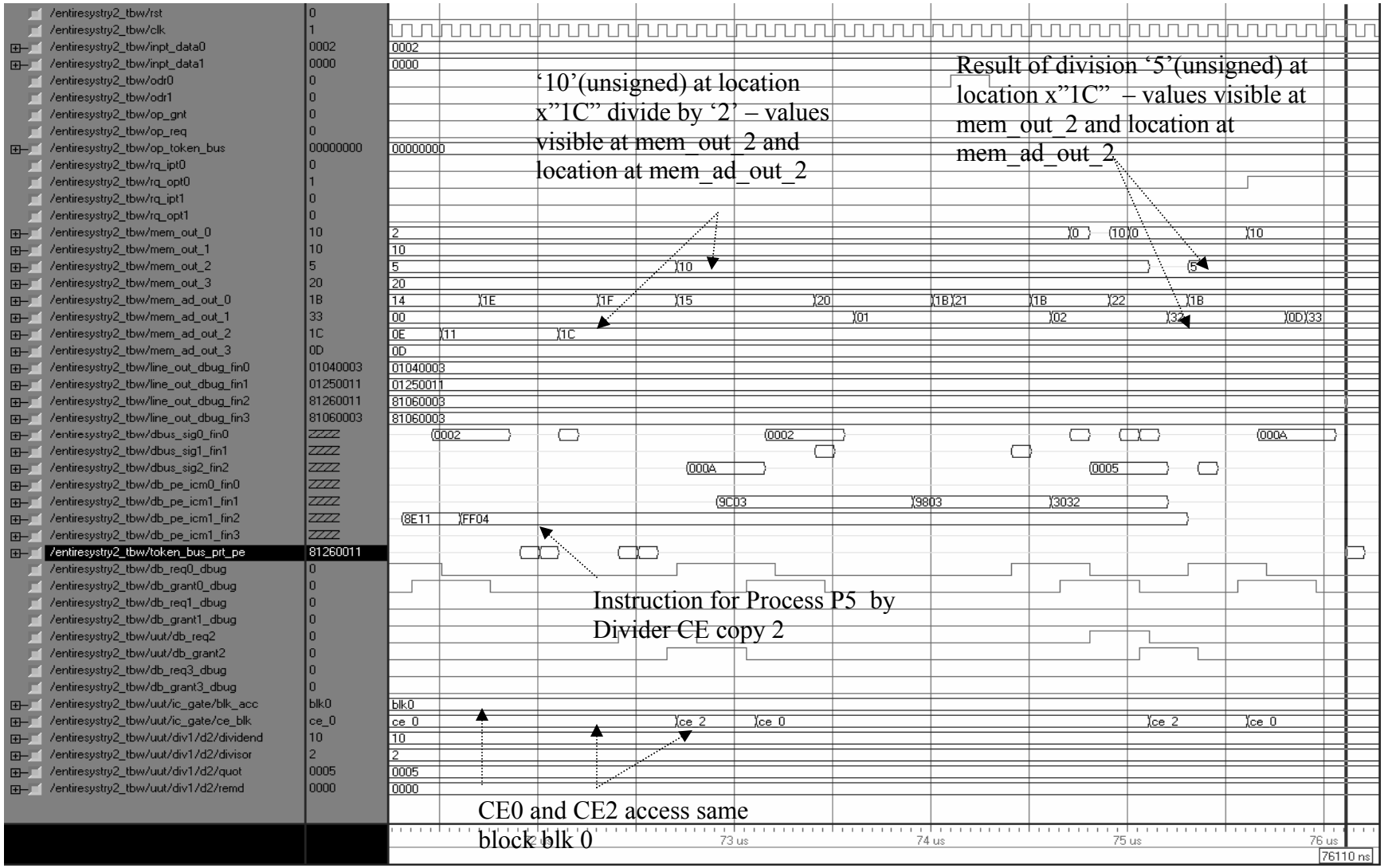

Figure: 5.27 Div Op.for P5 with Results, Command Token to PRT Mapper for Copy 2

In Figures 5.25 and 5.27 it can be seen that two CEs are accessing the same memory block ‘blk0’.In the meanwhile the PRT Mapper allocates CE1 to compute the process P6 of copy 1 as can be seen from Figure 5.28. The instruction x”9C03 9803 3032”for the join operation is issued to CE1.From Figure 5.12, it is understood that in the process P6 the values obtained from the result of process P4 are subtracted from result of process P5. The value unsigned ‘5’ (result of division) at location x”0D”, is subtracted from unsigned ‘20’ (result of multiplication) at x”0E”and the result unsigned ‘15’ is stored at location x”0F”. The result of the process P6 is outputted by the process P7. At the end of process P6 a command token is issued to the PRT mapper and consequently the PRT Mapper allocates the process P7 to CE0. The instruction for process P7 is issued by CE0 x”9C03 3039”. This process outputs the results of the subtraction operation in P6. Hence the result can be seen as explained earlier in Figure 5.29. Also it can be seen in the Figure the command token for process P4 of copy 2 being issued to PRT Mapper and eventually a command token x”0524FF11”seen at port “token_bus_prt_pe” issued to the Multiplier CE to execute the process P4 of multiplication.

After the command token for the process P4 is issued to Multiplier CE, the instruction is issued and multiplication takes place. The value unsigned ‘10’ stored as a result of the execution of process P2 at x”1B” is multiplied by unsigned ‘2’, the result is stored at the same location x”1B”. At the end of process P4, a command token x”81060011” for the join process P6 is issued. This is shown in Figure 5.30.

The PRT Mapper allocates the next process P6 to CE1 since it is the most available CE at that point of time. The instruction for the process P6 is given out to CE1, x”9C11 9811 3032”.This can be see from Figure 5.31. Subtraction operation takes place. The result of the process P5 (division) is subtracted from the result of P4 (multiplication) similar to the copy 1. The result is stored at x”1D”, refer to the Appendix B. The value of the final result is outputted in the next process P7. Hence after the execution of process P6 the command token to execute process P7 is given to the PRT Mapper which in turn allocates CE0, it issues the instruction x”9C11 3039”. The result unsigned ‘15’ can be seen in Figure 5.32 at port “mem_out_0” at location “mem_ad_out0” (x”1D”).

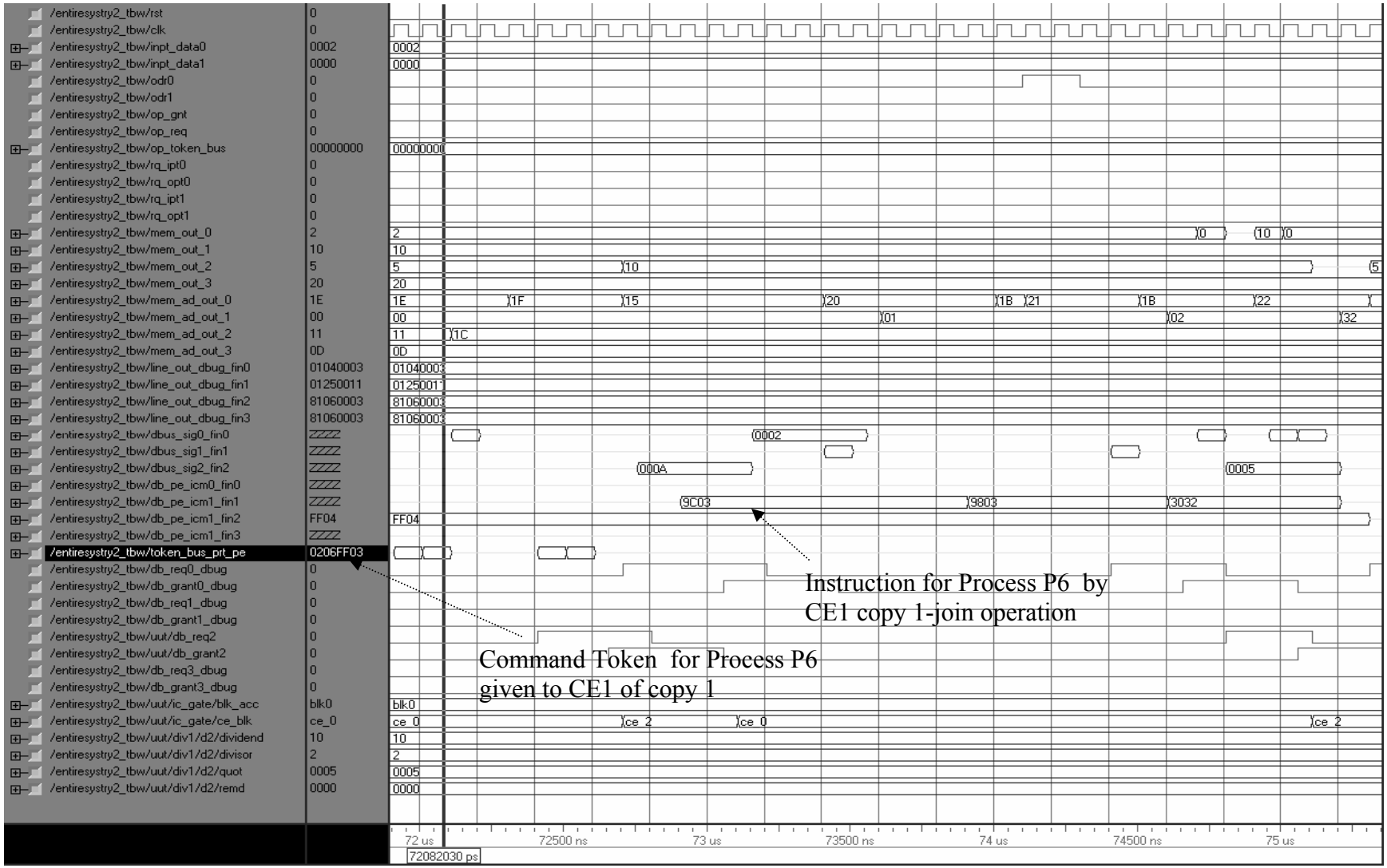

Figure 5.28, Join Instruction for Process P6 for Copy 1

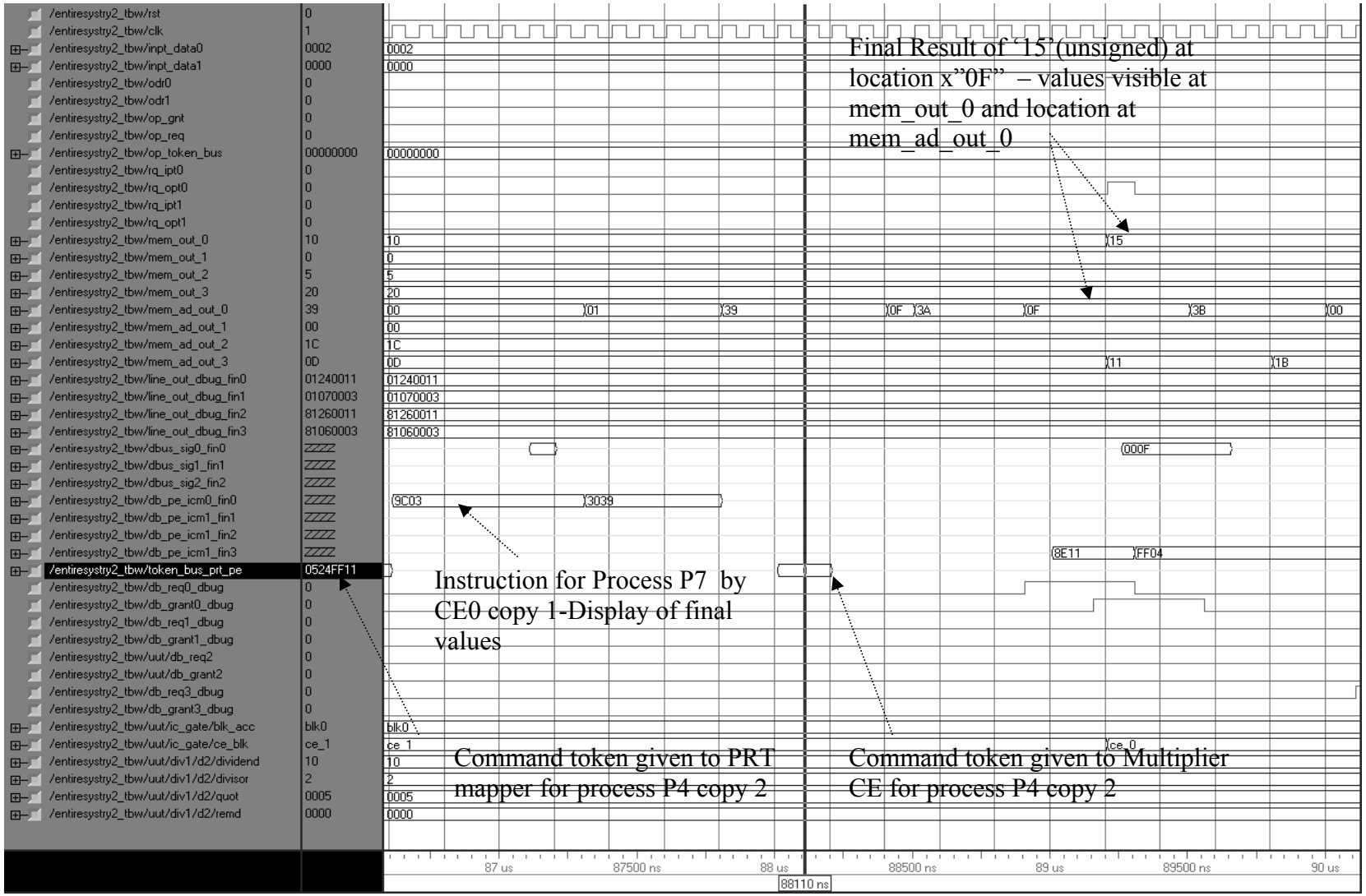

Figure 5.29, Process P7 Instruction and Final Output of the Result of P6 for Copy 1

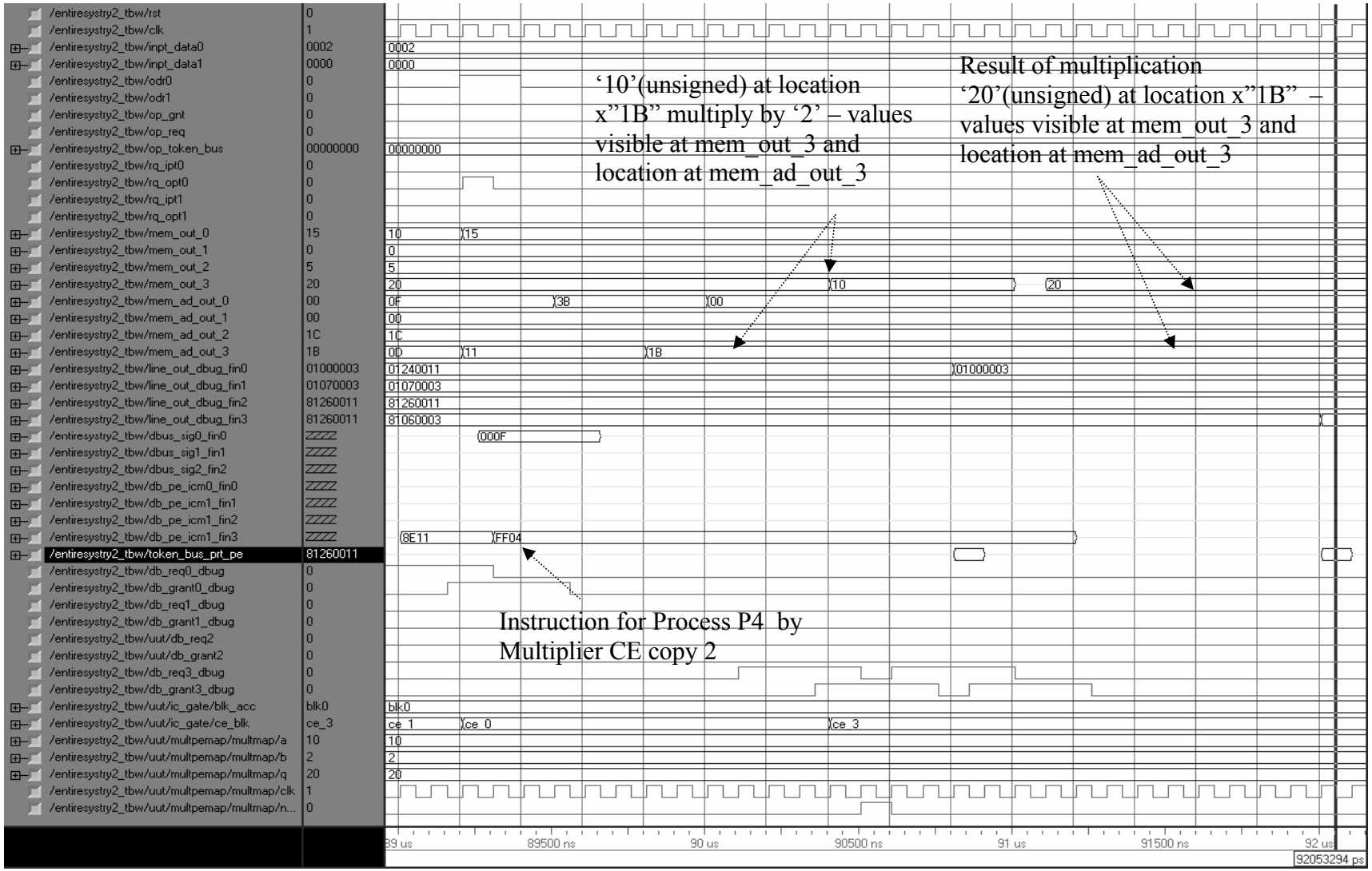

Figure 5.30, Multiplication Process Result, Command Token to PRT Mapper Copy 2

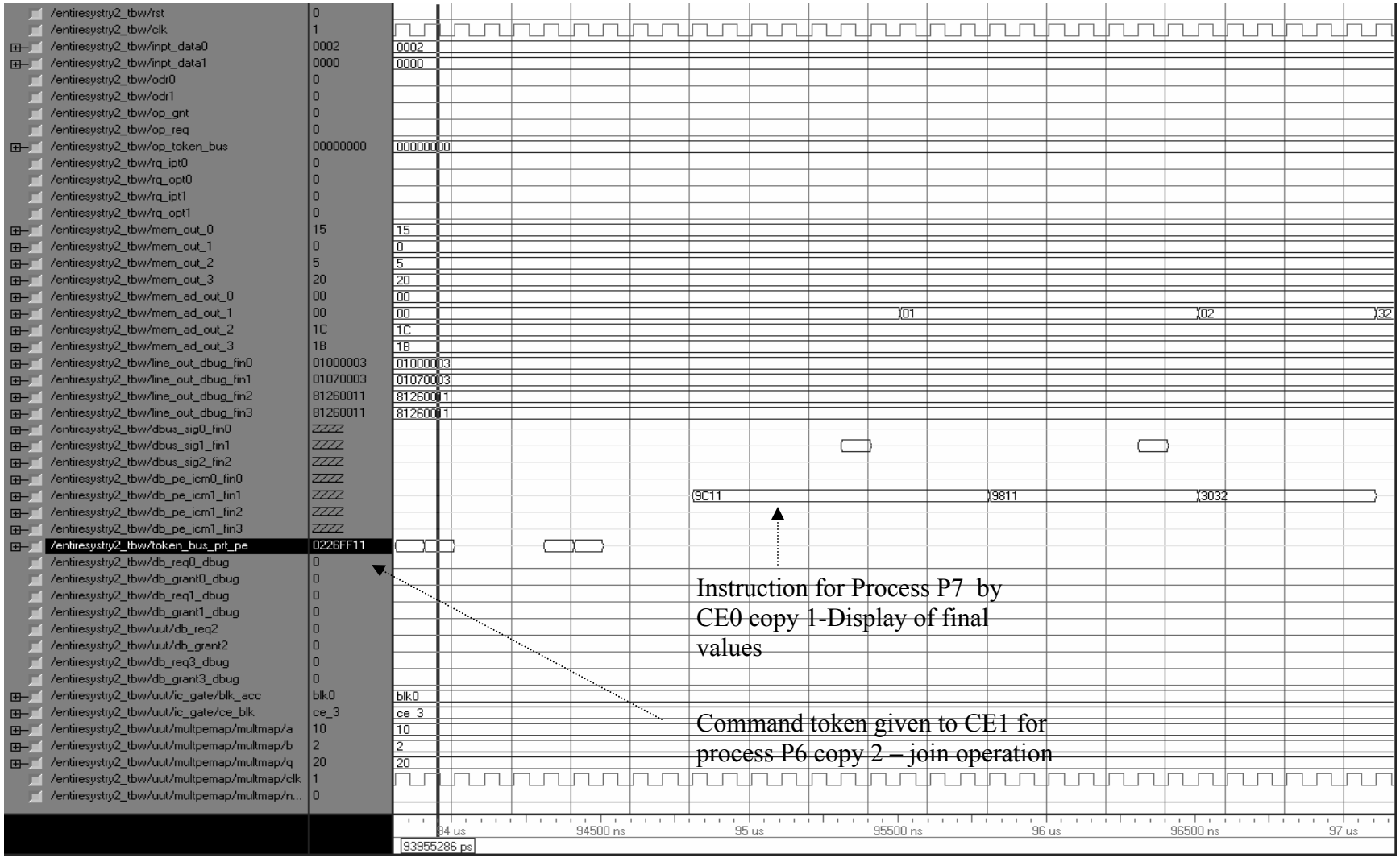

Figure 5.31, Join Process P6 Instructions for Copy 1

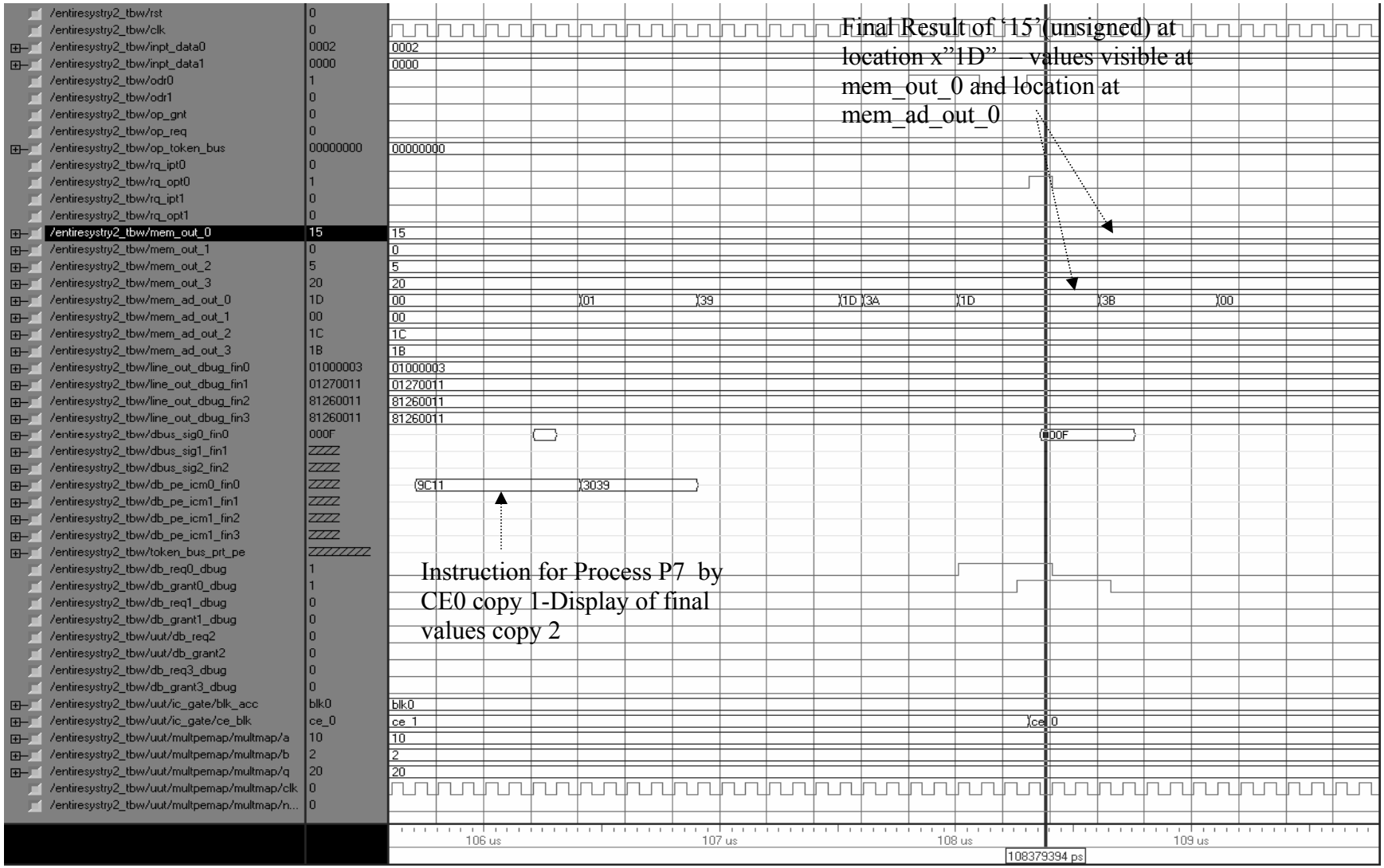

Figure 5.32, Process P7 with Final Value of the Result of Process P6 for Copy 1

5.1.7 Application 2 Described with Cyclic Flow Graph

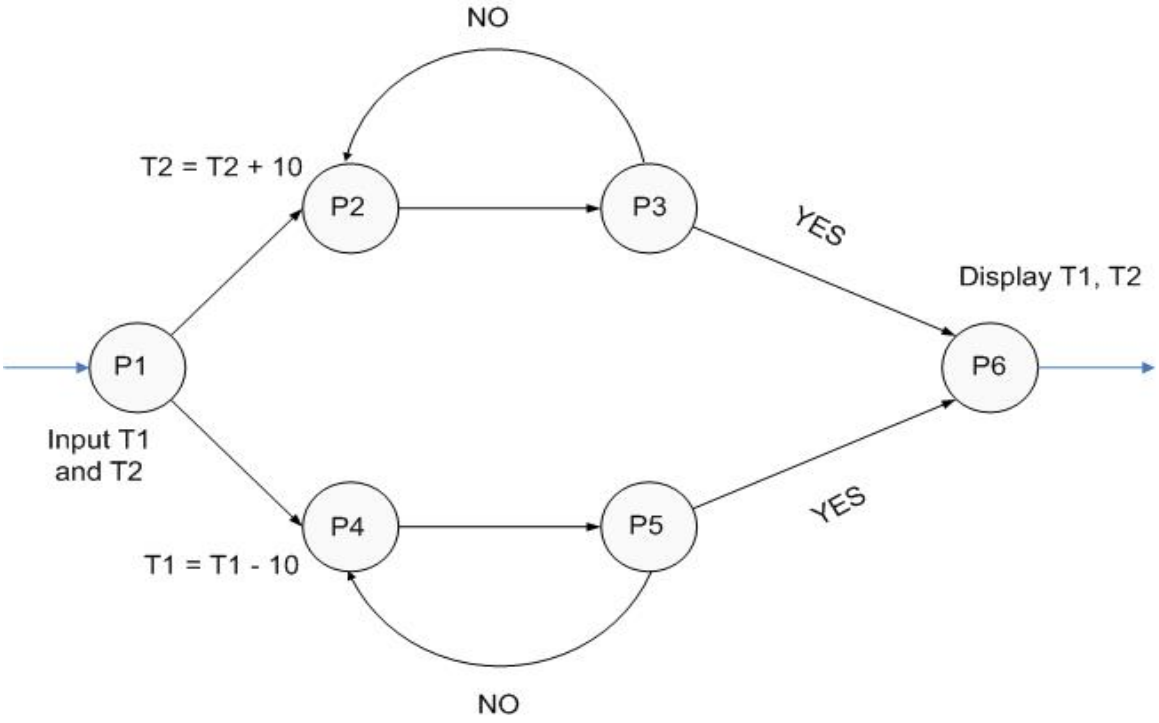

Figure 5.33, Application for Swapping of two Values

Each process shown in figure 5.33 is explained in more detail as follows:

P1 – Input 2 large values say T1 and T2 with T1>T2

P2 – Add unsigned '10' to T2 to get a new value of T2.

P3 - Check if T2 =T1 orig. If yes, branch to P6 (Exit PN), display both T1 and T2
Else branch to P2 again (feedback loop)

P4 – Subtract unsigned '10'from T1 and update the new value of T1.

P5 – Check if T1=T2 orig. If yes, branch to P6 (Exit PN), display both T1 and T2
Else branch to P4 again (feedback loop)

P6 – Display the values of T1 and T2 and then exit. The values should be swapped with respect to their original locations.

There are certain modifications done to the HDCA system to function correctly for the loop application described here. The figure 5.33 shows that process P6 is a join operation it is executed only when the condition for the Exit PN is satisfied. Each time the execution of the application loops back (P3 loops back to P2 and P5 to P4) the command token generated and issued to the PRT Mapper by the CEs executing processes

P3 and P5 have the join bit field set to logic ‘1’[5]. However the next process is process P2 for P3 and process P4 for P5 rather than process P6, the actual join process. The join process check in the controller of PRT mapper is modified to handle this issue. When the condition for the Exit PN is satisfied the system should be able to properly join the two processes P4 and P5 in this case. In order for this part to function correctly the ‘StopL’ token format is modified. The bits from 15 down to 8 in the ‘StopL’ token are all at logic ‘0’ state. The bit 15 in this case is modified to be at logic’1’.This bit is used by the PRT Mapper to indicate that the token is for the real join operation. The system works perfectly with these changes made and outputs correct results refer to Appendix B.

The process flow graph shown above is tested by running a copy of an application. The initialization tokens and instruction set for processes of the application are described in detail in Appendix B. For this application the value of T1 (original) is unsigned ‘100’and value of T2 (original) is unsigned ‘60’. As can be seen from the description of processes the values of T1 and T2 change and are different from original as the application progresses, hence the values are inputted twice into the data memory. This ensures that one of the pairs of values inputted remains intact. A command token (x”01010003”) is inputted into the system. It triggers the system to begin the application. It addresses the PRT mapper to map the process P1 to a CE which is the most available at that point of time. It is observed that PRT mapper allocates process P1 to CE0 as it has higher priority over CE1. The instruction is issued by controller of CE0 which prompts the CE0 to input values from the input ROM, it should be noted here that a new component ‘inrom’ has been added into the system in order to facilitate inputting different values of data into the system’s data memory. Figure 5.34 shows the instruction issued to CE0 indicated by port name “db_pe_icm0_fin0” (connection between interface controller module and PE) refer to Figure 1.1; the signal is taken out as the port name. The execution of the process P1 begins by inputting values from ‘inrom’. Figure 5.34 shows that two values unsigned ‘60’ and ‘100’ are input at locations x”03” and x”04”. The values can be seen at the “mem_out_0” since CE0 is executing the process. The Figure 5.31 shows the inputting of the next 3 values into the data memory. They are unsigned ‘10’, ‘60’ and ‘100’ at locations x”05”,”06” and “07” respectively. In the set of waveforms that follow it should be noted that the ports “db_req0_dbug, db_req1_dbug,

db_req3_dbug" are the signals coming from CEs 0, 1 and 3 respectively. These signals go high when the CE requests access to the memory. Similarly ports "db_grant0_dbug, db_grant1_dbug, db_grant3_dbug" are signals coming from the crossbar network to the CEs that are granted access.

At the end of process P1, it forks to two processors P2 and P4 as can be seen from 5.33. Hence two command tokens are issued by CE0 to PRT Mapper; it maps the two processes to CEs depending on the availability. It is observed that process P4 is allocated to CE0 and process P2 to CE1. In this case for process P2, CE1 is made the most available in a similar way as described in the application 1, this setting could be changed. It can be seen in Figure 5.36, the instruction for P4 is issued to the CE0 and instruction for P2 is issued to CE1. The execution of the process P2 can be seen at "mem_out_1" and corresponding address at "mem_ad_out_1" since CE1 is executing it. Similarly it can be seen that CE0 is executing the process P4 on observing line "mem_out_0"and location at "mem_ad_out_0".The result for process P2 which is unsigned '70'is stored at location x"03" and result of process P4, unsigned '90' is stored at location x"04".

At the end of each process P2 and P4, command token is issued by the CEs to the PRT Mapper which in turn finds the most available CE and maps the processes to it. Figure 5.37 shows the instruction (x"9C03 3014") for process P3 is being issued to CE0. In this process the values unsigned '100' at x"07" and value at x"03" are compared for equality. In this case the values are not equal since unsigned '100' ≠ unsigned '70'.The execution loops back to process P2 as per the condition stated in the explanation of the processes.

Similar to process P3 which executes after P2, process P5 is executed after process P4. In process P5 also the values unsigned '60' at x"06" and value at location x"04" are compared. If the values do not match, the execution loop backs to execute process P4 again. At this time the values are not equal hence the process P4 is executed again. This is shown in Figure 5.38. It also shows the instruction for process P2 issued by the interface controller of CE1 to CE1.

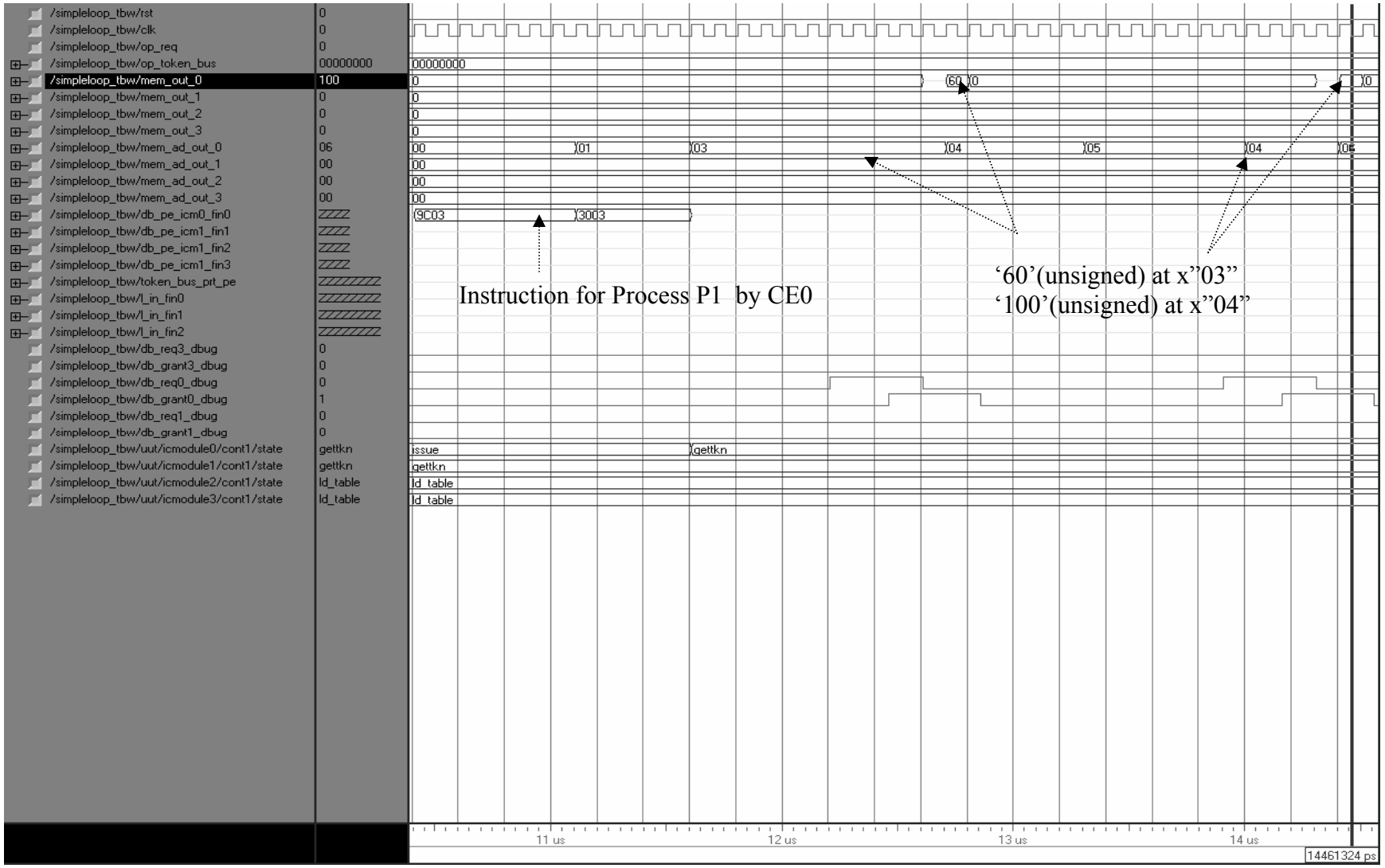

Figure 5.34, Instruction for Process P1 and Input of First two Values

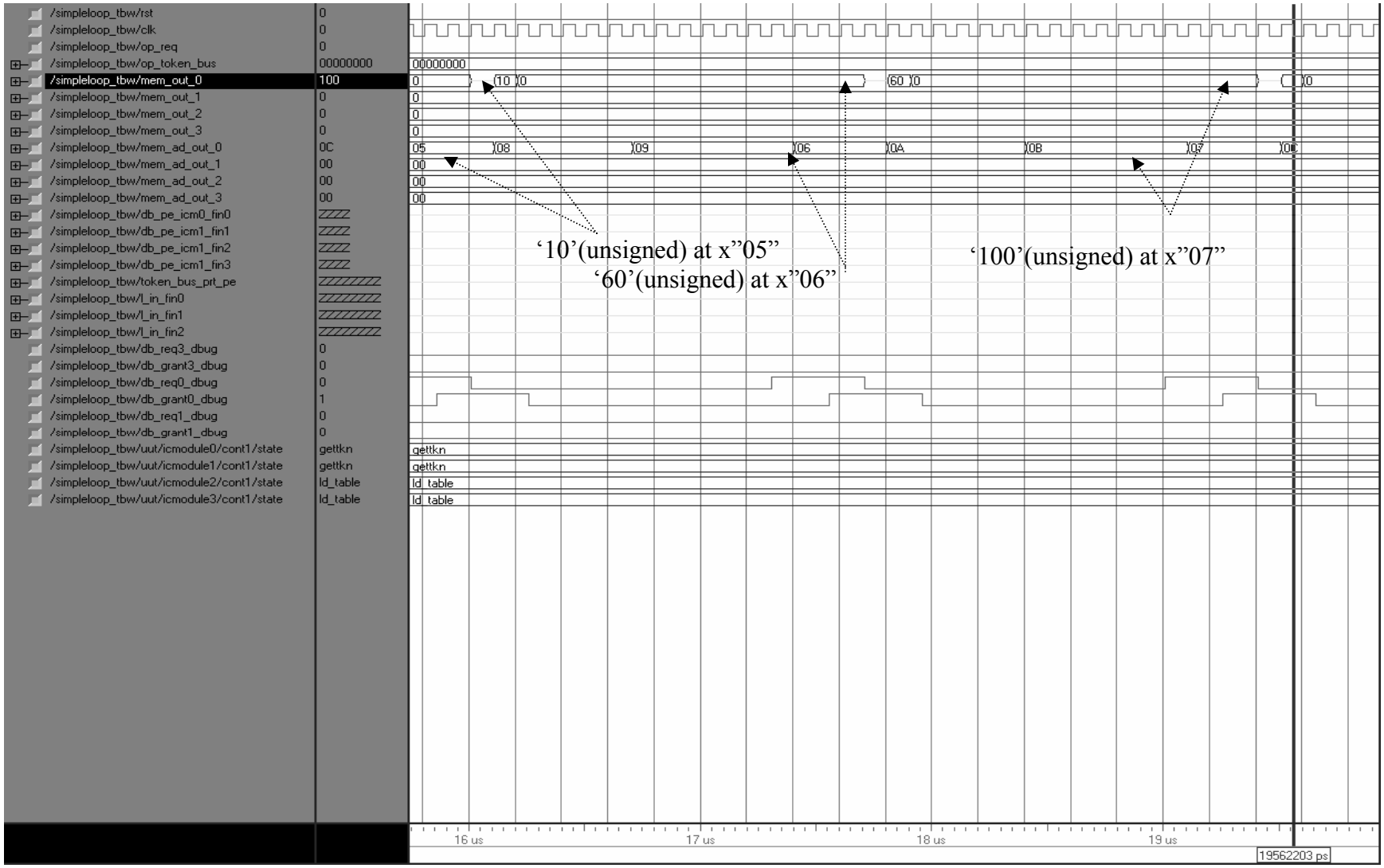

Figure 5.35, Process P1 Inputs of Last 3 Values

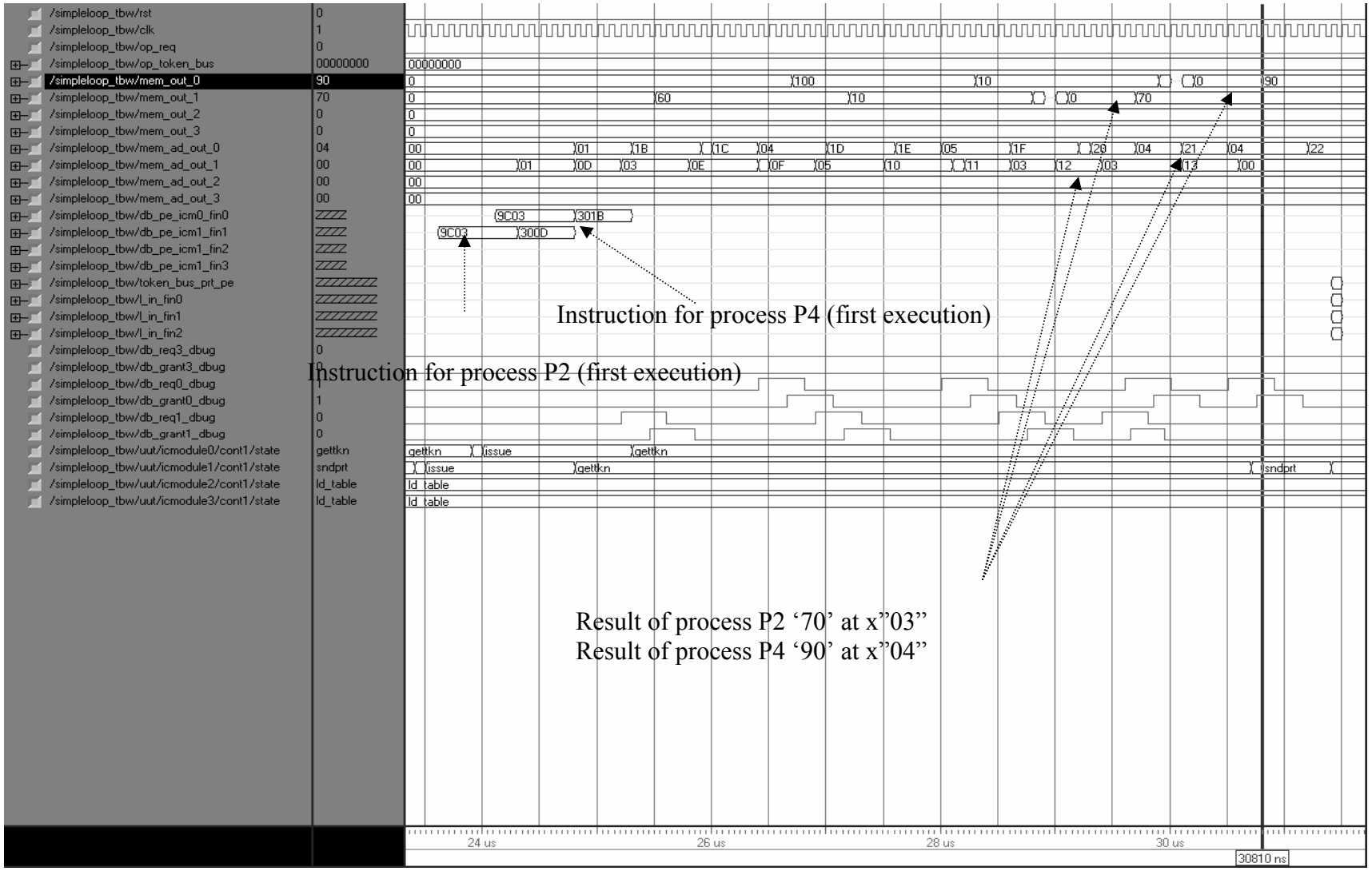

Figure 5.36, Instructions for Processes P2 and P4 with Results

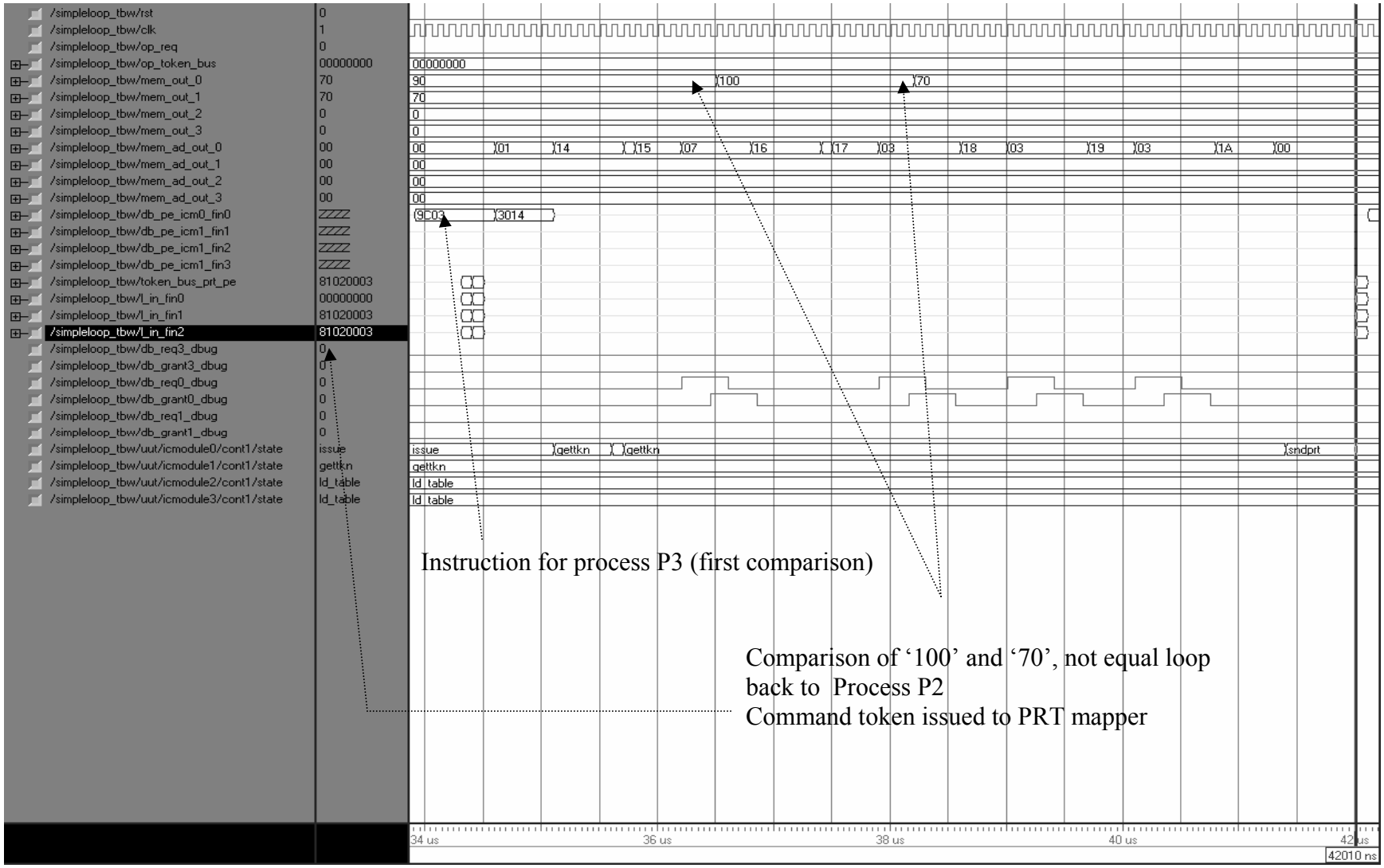

Figure 5.37, Process P3: First Comparison

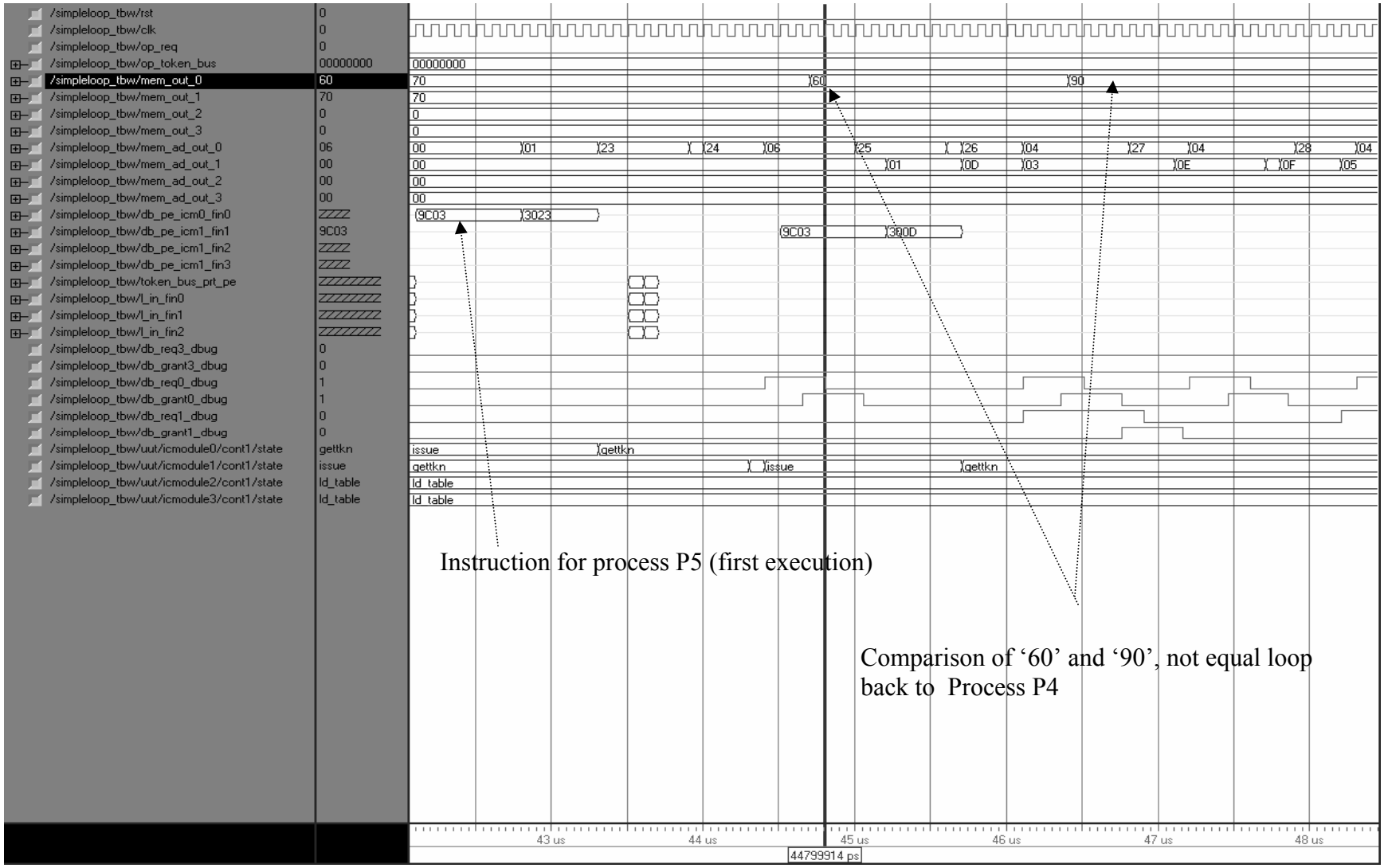

Figure 5.38, Process P5: First Execution

The detail execution of process P2 is shown in Figure 5.39. The addition operation takes place and at this point of time the result calculated is unsigned ‘80’ (70 + 10) and is stored at location x”03”. It can be concluded by observing the ports “db_req0_dbug, db_req1_dbug” and corresponding “db_grant0_dbug, db_grant1_dbug”that CE0 and CE1 are accessing the same memory block (block 0). However since the CEs are accessing the same memory block the processor requesting the memory access early in time is granted access. After execution of P3 and looping back to P4, the PRT Mapper allocated process P4 to CE0 as can be seen in Figure 5.40. The subtraction of unsigned ‘10’ from value at location x”04”takes place. In this case the result calculated is unsigned ‘80’ (90 -10) at location x”04”.

At the end of process P2, the updated value is checked once again against the original value. This is done in process P3 as seen earlier. The execution of process P3 for the second time is shown in Figure 5.41. The CE0 executes this process as is seen from the instruction issued by interface controller of CE0 to CE0. Since the values compared are not equal, the application loops back to execute process P2 again. The command token x”81020003” is issued by CE0 to PRT Mapper (note the “token_bus_prt_pe” port which is a signal connecting CEs and Token Mapper, refer to Figure 1.1).

After the execution of process P4 as explained earlier, the original value at location x”06” which is unsigned ‘60’ is compared with the new value obtained as a result of process P4. This is shown in Figure 5.42. The value at location x”04” is now unsigned ‘80’. It is evident that the values being compared are not equal, hence the application execution has to loop back to execute process P4. The command token for P4 is issued by CE0 to PRT Mapper as shown at port “token_bus_prt_pe”: x”81040003”. The Figure also shows the instruction for process P2 issued to CE1 by its interface controller. The execution of the process P2 and the result is shown in detail in Figure 5.43. As can be seen from port “mem_out_1” unsigned ‘80’ at x”03”(“mem_ad_out_0”) is added with unsigned ‘10’ at x”05” result unsigned ‘90’is stored at the x”03”.

The process P4 is also executed again for the third time. The execution is done by CE0. It is shown in Figure 5.44. The value unsigned ‘80’ at x”04” obtained from earlier execution is subtracted by unsigned ‘10’ at x”05” and the result unsigned ‘70’is stored at the same location x”04”.

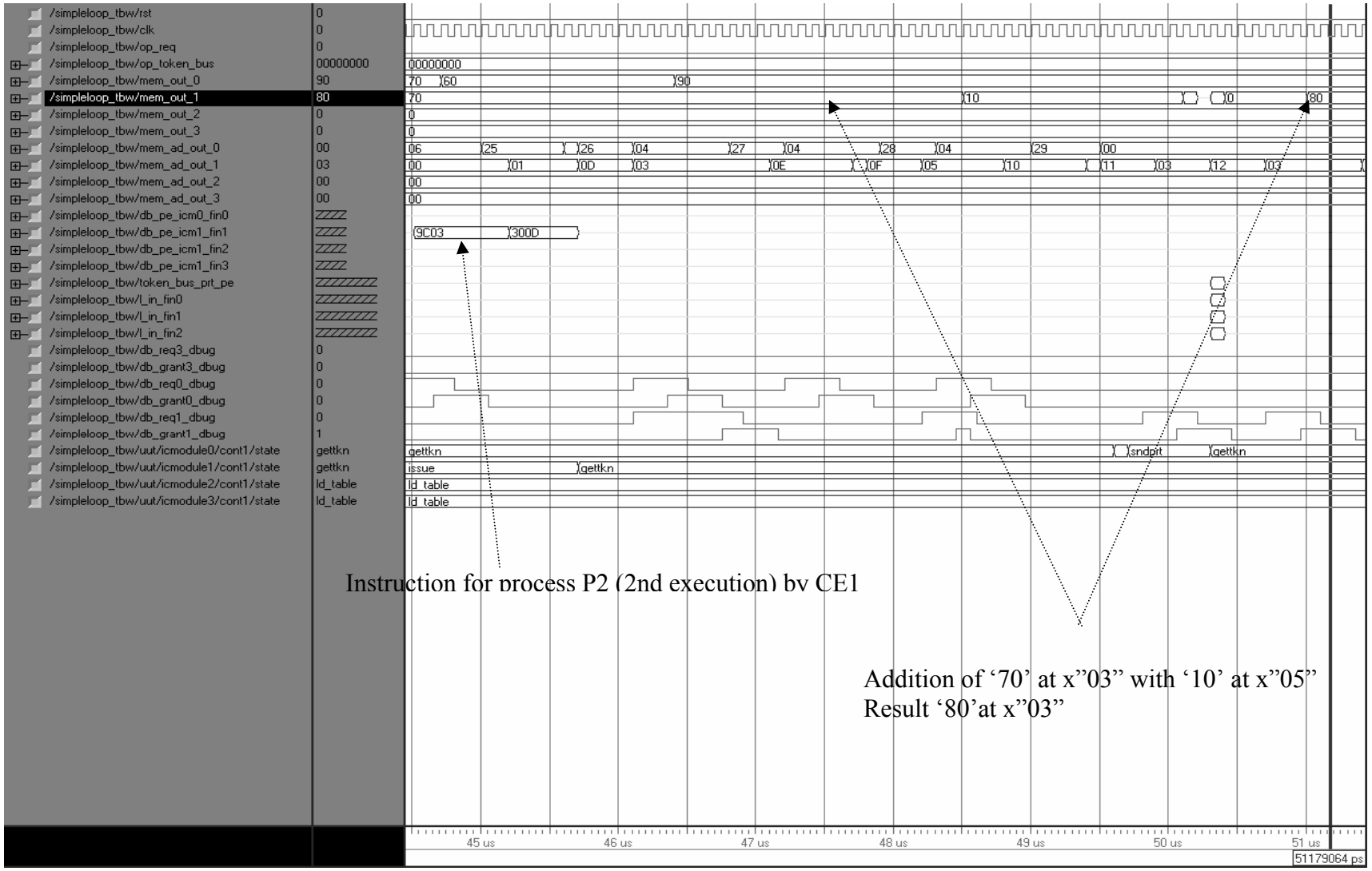

Figure 5.39, Process P2: Second Execution

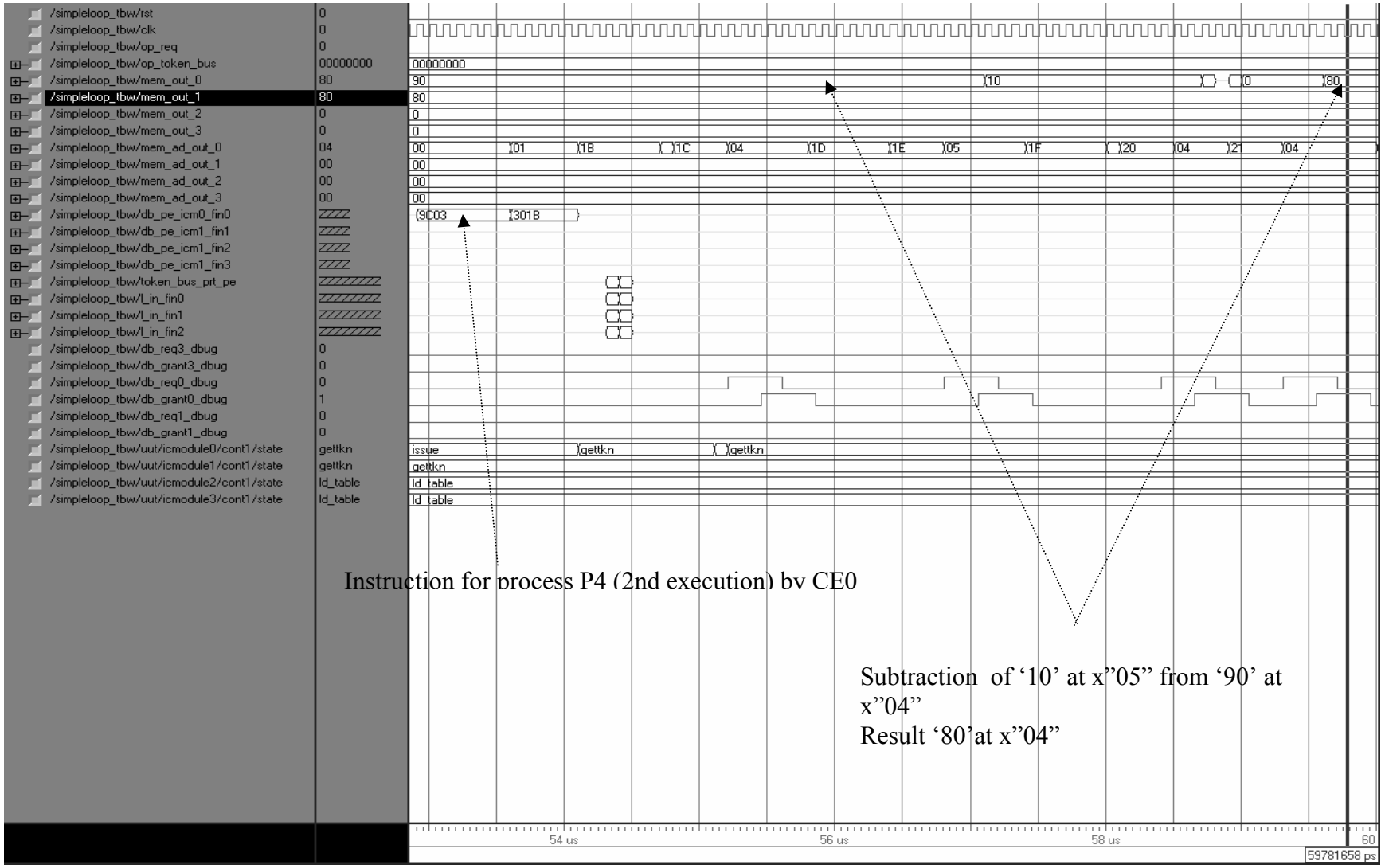

Figure 5.40, Process P4: Second Execution

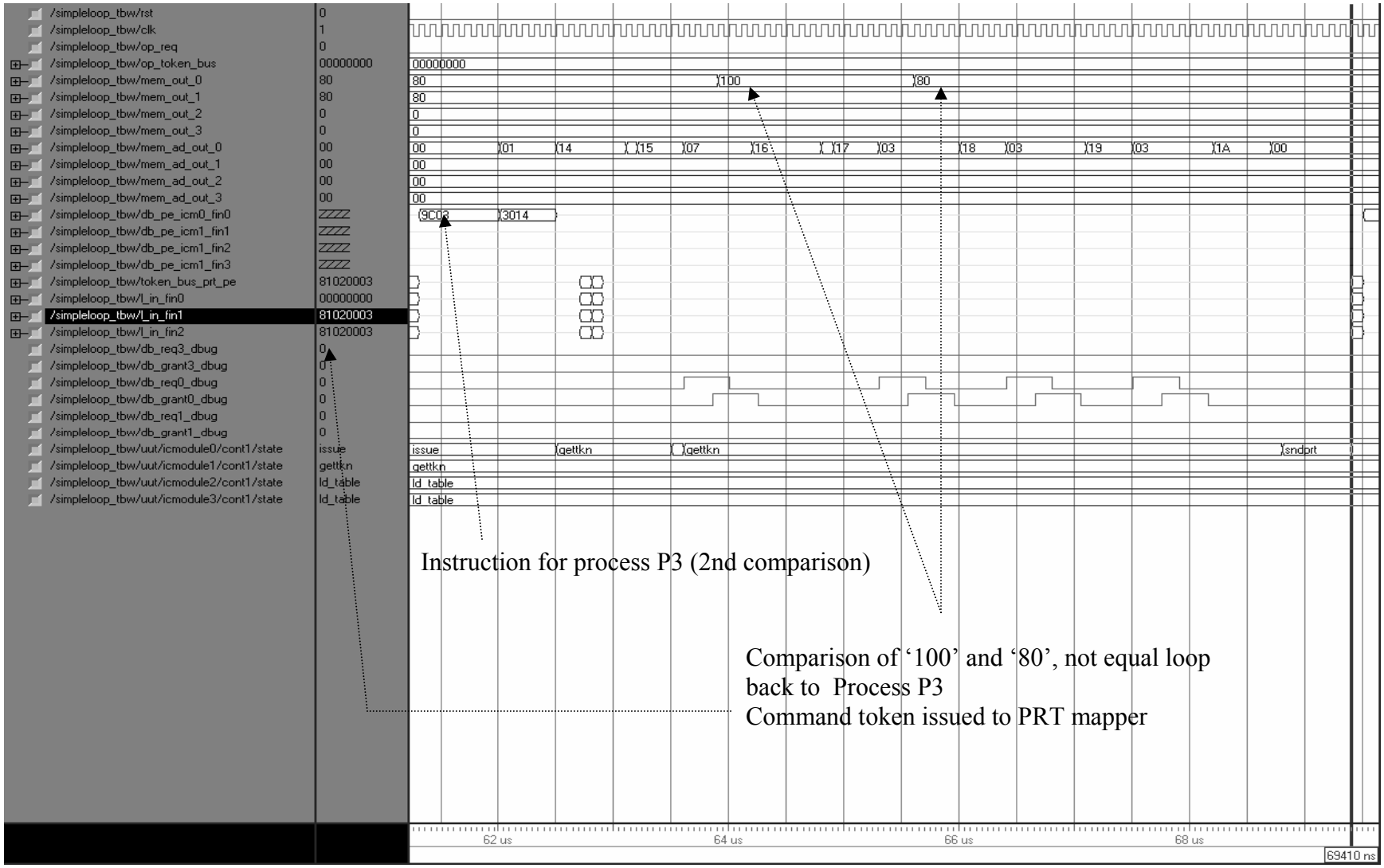

Figure 5.41, Process P3: Second Execution

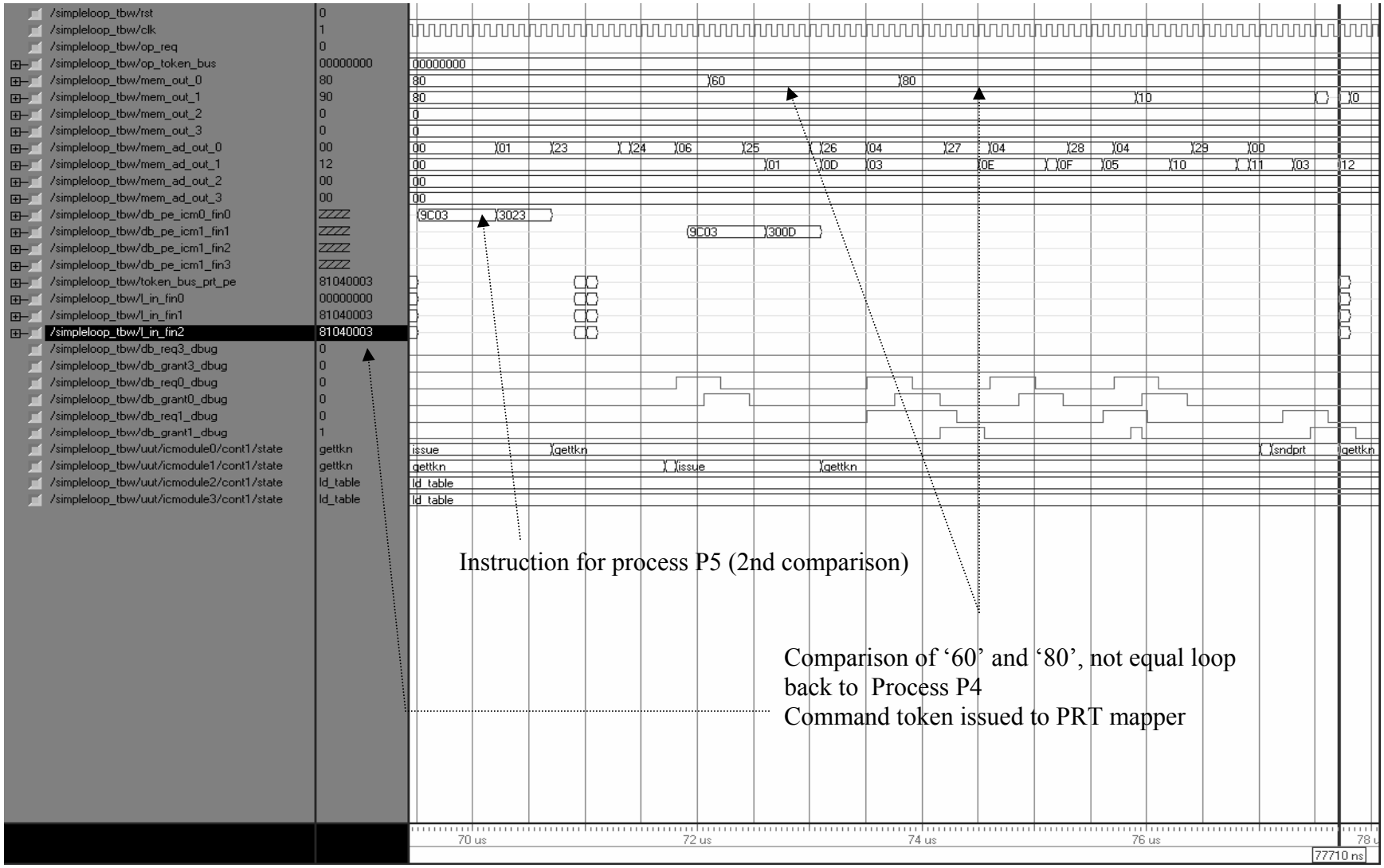

Figure 5.42, Process P5: Second Execution

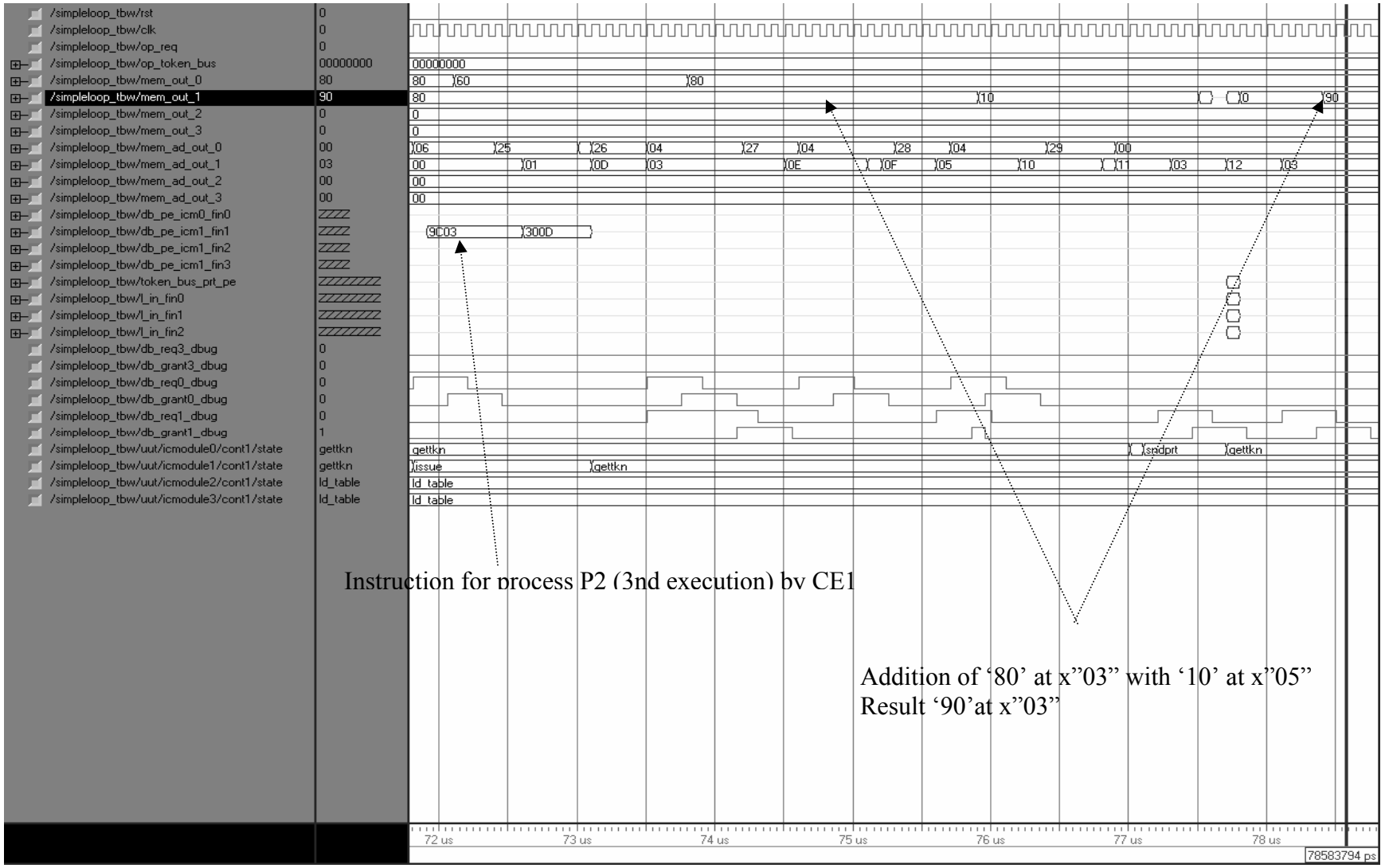

Figure 5.43, Process P2: Third Execution

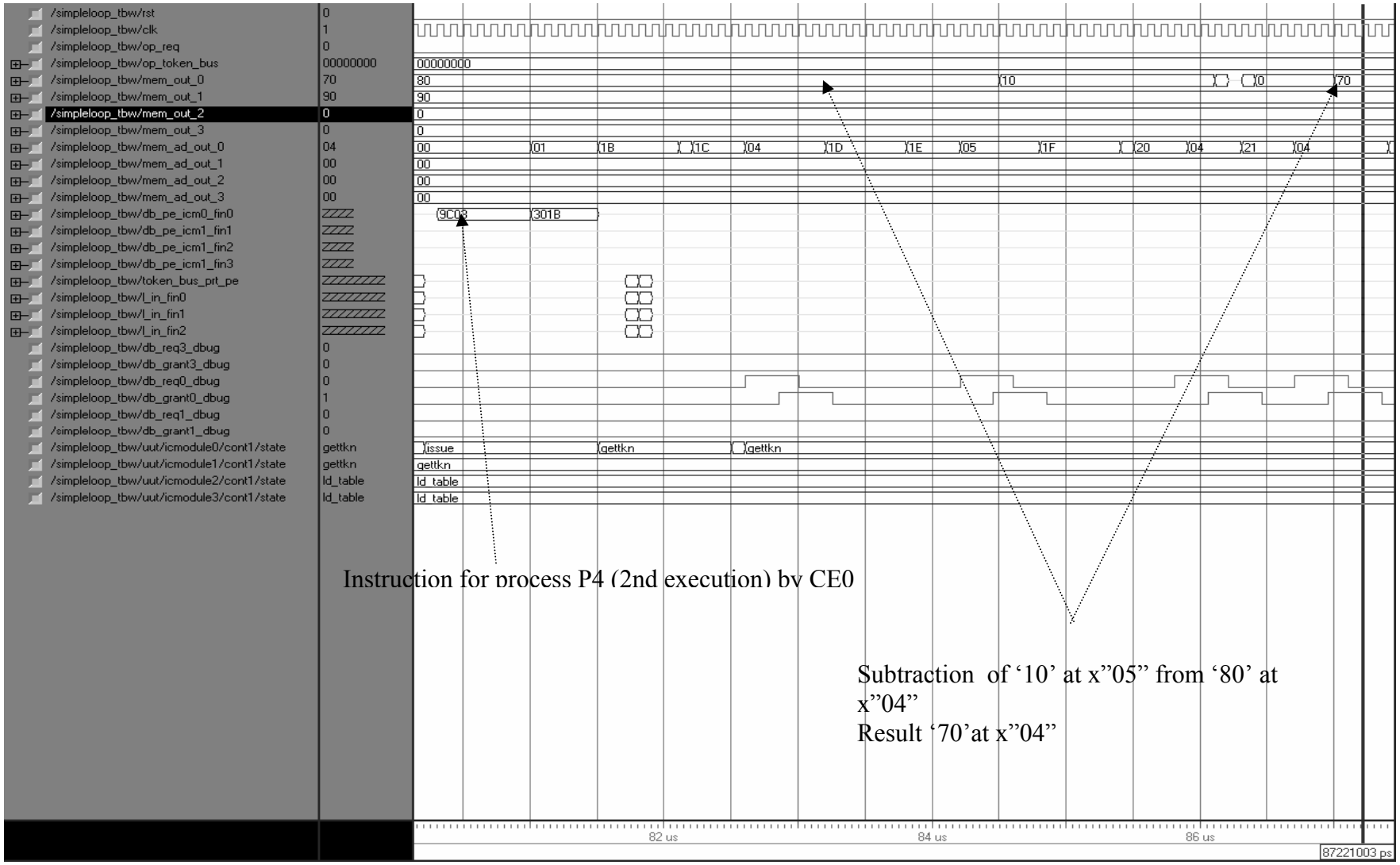

Figure 5.44, Process P4: Third Execution

At the end of process P2 again the comparison operation takes place in process P3.The values compared are unsigned ‘100’ at x”07” and unsigned ‘90’ at x”03”. Since the comparison result is not equal, it loops back again to execute P2. Figure 5.45 shows the execution of process P3 for the third time.

Similarly the process P5 is also executed again for the third time. After the execution of process P4 the updated value is compared with the original. Here new value unsigned ‘70’ at x”03” is compared with the original unsigned’60’ at x”06”. The values are not equal and hence application loops back to execute process P4. The values can be seen at port “mem_out_0” and locations at “mem_ad_out_0” of Figure 5.46.The instruction for the process P2 is also seen to be issued to CE1 by its interface controller. The detail application is shown in Figure 5.47.

The process P2 is executed again after the looping back from process P3. As seen in Figure 5.47 process P2 is executed by CE1 and the results can be viewed at port “mem_out_1” and locations at “mem_ad_out_1”. In this case unsigned value ‘90’ at location x “03”added to value unsigned ‘10’at x”05” to get the result unsigned value ‘100’at x”03”.one can also observe the requests and grants port lines depicting that both CE0 and CE1 are accessing the same memory block ‘0’.

Process P4 is also executed again as a result of looping back from process P3. This process is executed by CE0 as seen from the Figure 5.48. Observing port “mem_out_0” and locations “mem_ad_out_0”, the subtraction operation and the result can be seen. The value unsigned ‘10’ at x”05” is subtracted from updated value at location x”04”, unsigned ‘70’ in this case. The result unsigned ‘60’ is stored at x”04”.

After the execution of process P2 for the fourth time the process P3 is executed again by CE0. The value at x”03” unsigned ‘100’is compared with the original value unsigned ‘100” at x”07”.The result of the comparison is satisfied that is the two values compared are equal. Hence the condition for exiting the loop is satisfied, the application proceeds to execute process P6 which is a join operation as can be seen in Figure 5.33. This is shown in Figure 5.49.

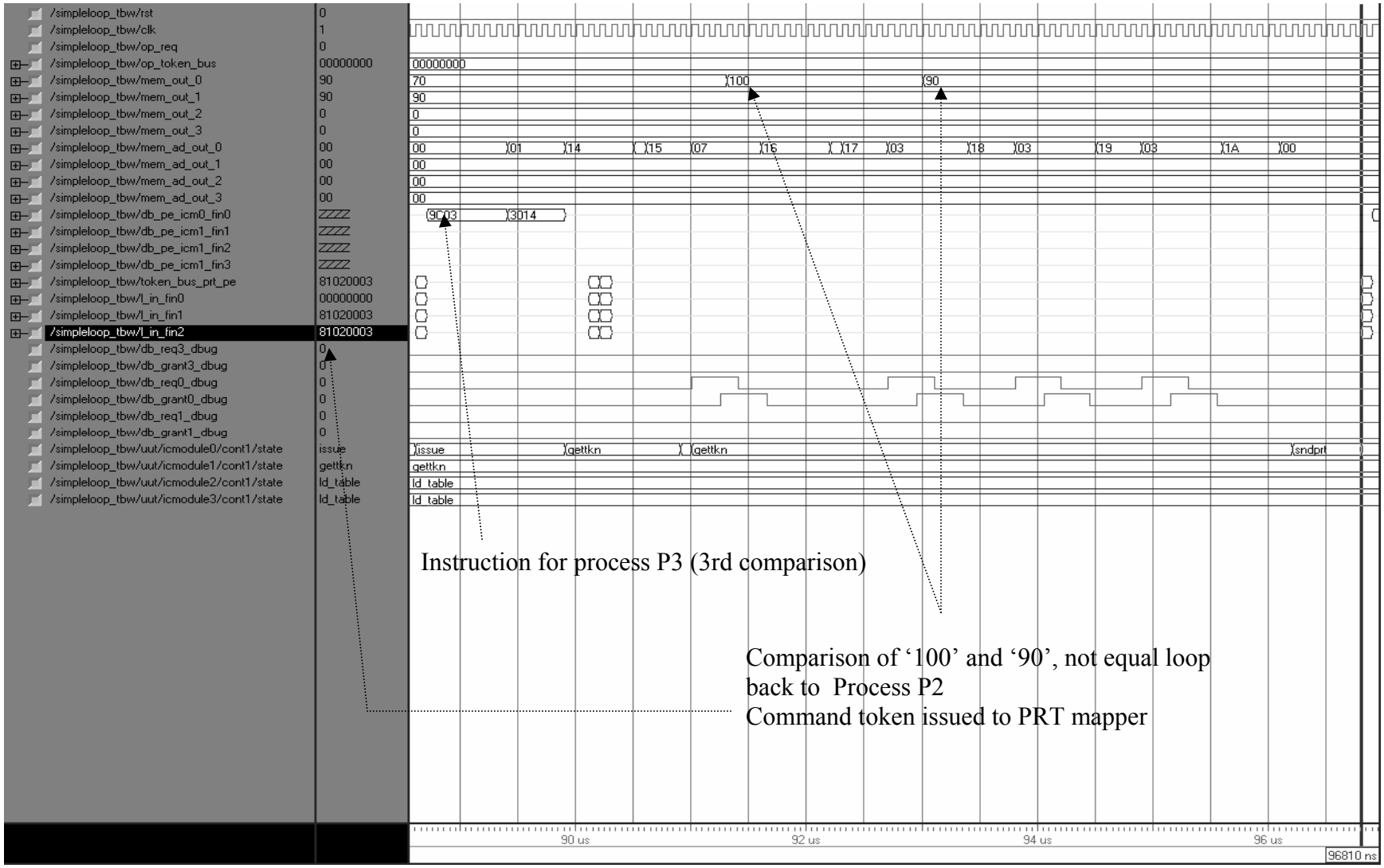

Figure 5.45, Process P3: Third Execution

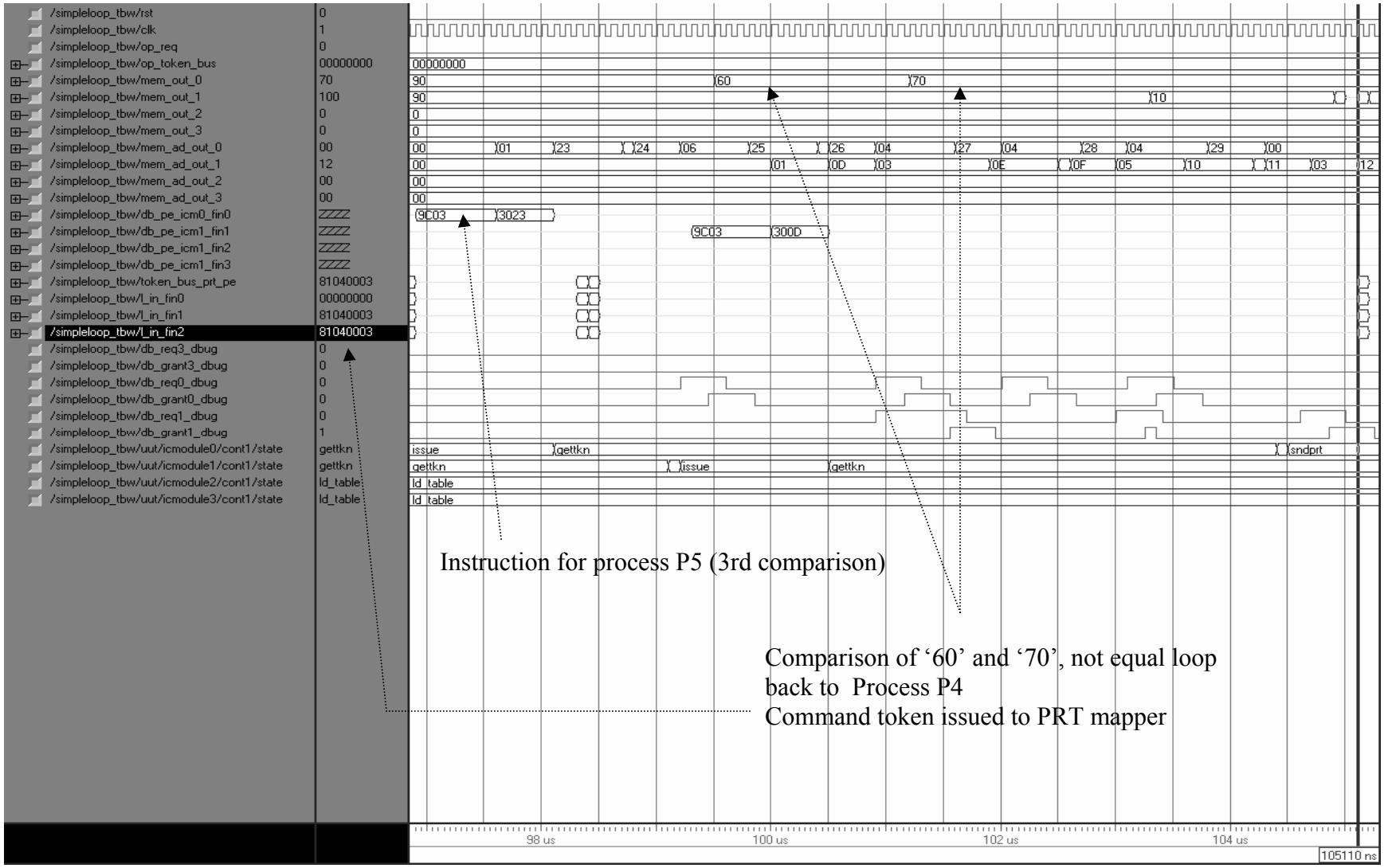

Figure 5.46, Process P5: Third Execution

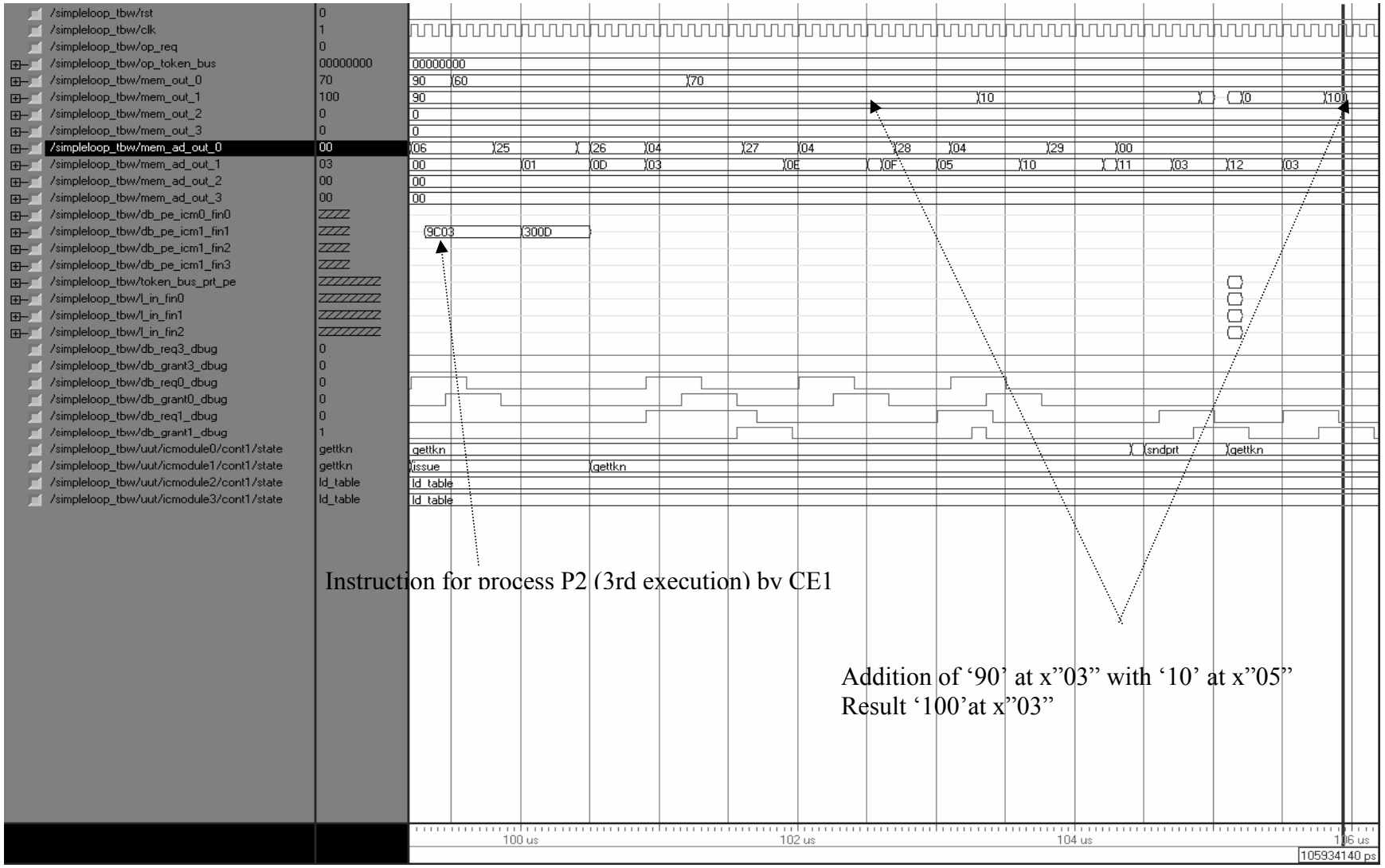

Figure 5.47, Process P2: Fourth Execution

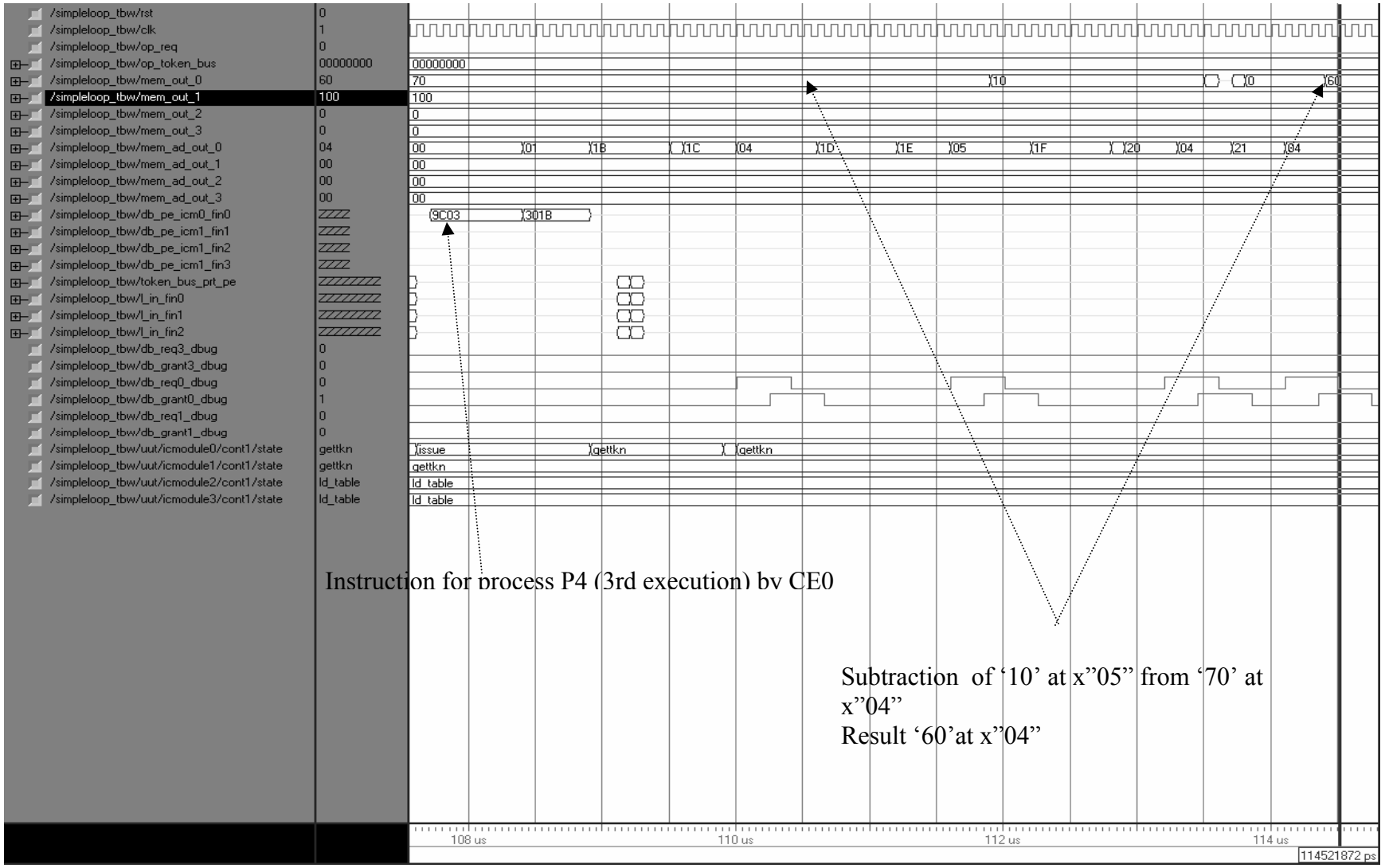

Figure 5.48, Process P4: Fourth Execution

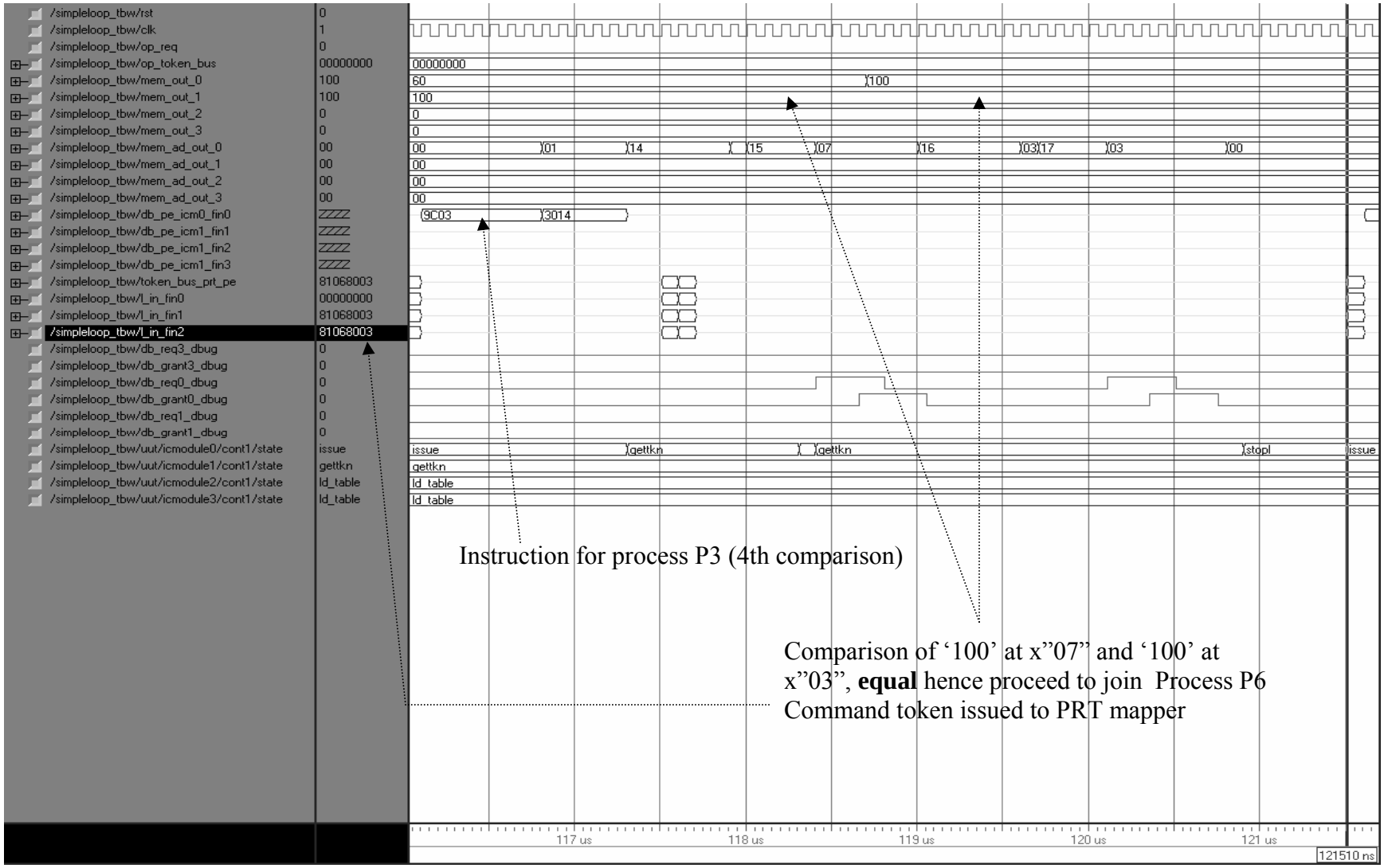

Figure 5.49, Process P3: Fourth Execution, Exit of Loop, Token Issued to PRT Mapper

It can be observed from all the Figures in this application that the signal ‘state’ is taken out from all the interface controller modules as a port.The controller for CE0 goes into the ‘StopL’ state since the Exit Loop condition is satisfied. CE0 issues a command token x”81060003”to the PRT mapper. The mapper waits for other join token which is sent by process P5 after the exit PN condition is satisfied.

The process P5 is executed by CE0 and the instruction is issued by its interface controller, it is shown in Figure 5.50. The value at location x”06”, ‘60’ (unsigned) is compared with updated value unsigned ‘60’ at x”04”. The values are equal and hence the condition for the Exit loop is satisfied. The execution proceeds to process P6 and CE0 issues a command token x” 81068003” similar to process P3 as described earlier. The state of the controller of CE0 goes into the ‘StopL’ state.

The PRT Mapper receives both the tokens of the join process P6. It maps CE1 as the most available CE to execute process P6. The instruction is issued by CE1 interface controller to CE1.The values can be seen at the port “mem_out_1” and locations on “mem_ad_out_1” port. It can be observed that the values ‘100’ (unsigned) which was at x”04” and ‘60’ (unsigned) at x”03” are swapped with respect to the locations. At the end of process P6 the value ‘100’ (unsigned) is at x”03” and value ‘60’ (unsigned) is now at x”04”.This is shown in Figure 5.51.

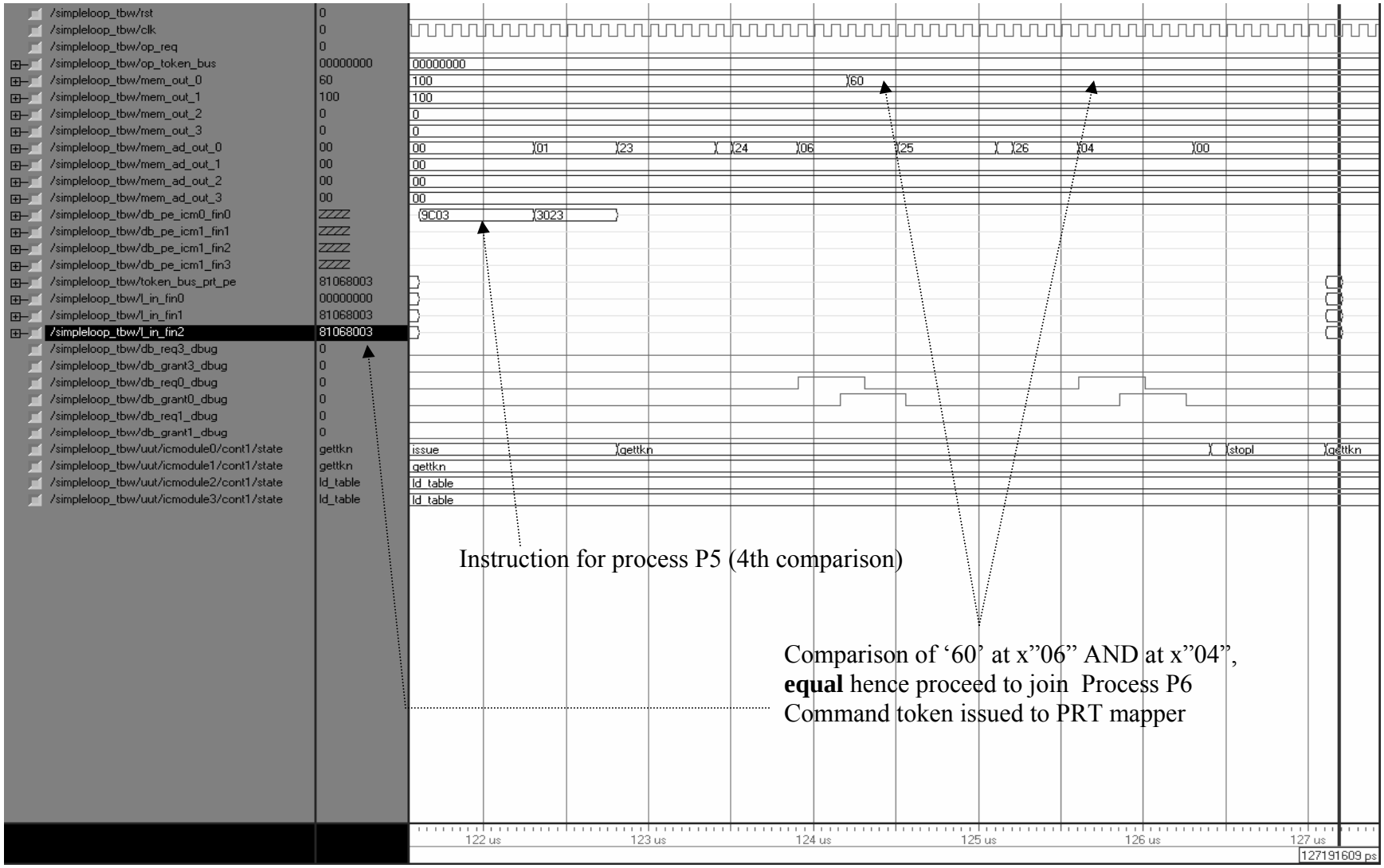

Figure 5.50, Process P5: Fourth Execution, Exit of Loop, Token Issued to PRT Mapper

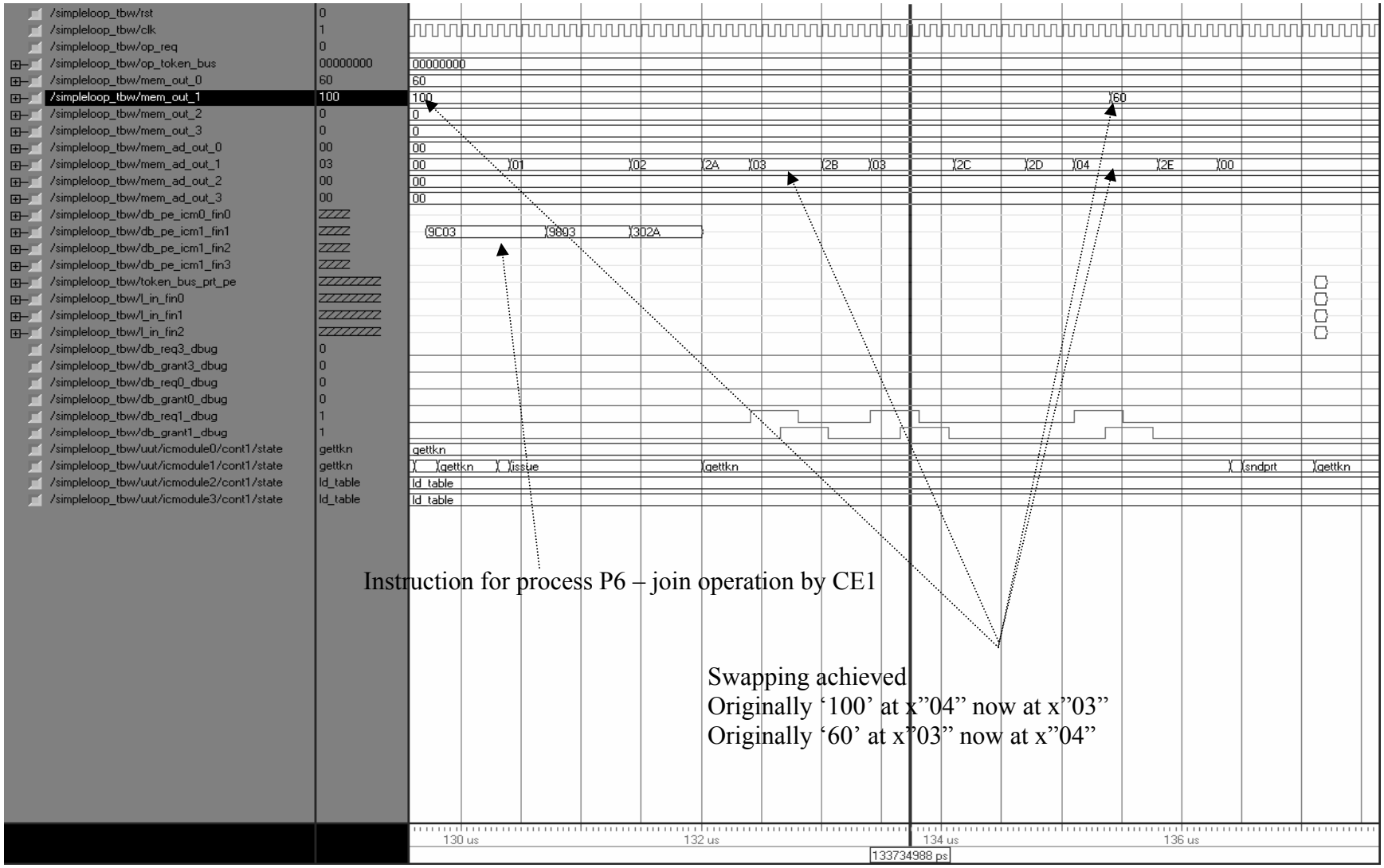

Figure 5.51, Process P6 Join Operation, Final Results, Values Swapped

5.1.8: Latency and Starvation Issues

Latency or delay within a crossbar switch, measured in terms of the amount of the input to output delay, is a constant. [4] The crossbar possesses a constant rate of delay. It is observed that the time required for the grant signal to go high after a request signal is made high by a CE is two and half clock cycles. Once the connection is established between the CE and memory block it takes one clock cycle to read out values from the memory and also one clock cycle to write in data. These delays can be observed from the waveforms shown for the execution of applications described above in section 5.1.3 and 5.1.4.

Chapter 3 discussed choosing the arbitration technique for processors requesting memory access. A variable priority arbitration scheme is used in the design of the crossbar in this case. The priority is based on the value of the queue depth of individual processors [5] [7] [15]. In case there is contention for the same memory block the processor with the highest memory block is granted access. The issue of starvation may arise when the processor, granted access to the memory block continues to request for memory access. In this case the other processors requesting memory access have to wait for an indefinite time until the processor having high priority ceases to request memory access. This can be a serious problem in interconnection networks and has to be given special attention. Fortunately, in this situation where the crossbar switch is being used in a HDCA system, this issue is taken care off. According to the design of the HDCA architecture [5,7,15] the PRT mapper allocates process requests to the individual CEs according to their availability at that point of time during an application run. The availability is determined by tokens lined up in the queue of the processors. Hence in a situation where in processor with highest priority keeps on getting access to the memory block, the PRT Mapper eventually assigns the new processes to the other CEs since the processor with highest priority already has deeper queue depth already. After some time the processor which had been granted access, no longer has the highest (deepest) queue depth and hence the access to the memory ceases. This allows the other processors to access the memory blocks.

However, in other systems where the crossbar could be used in an environment different from the HDCA system, one way to overcome this issue is to make use of the operating system used. The amount of time any processor can be granted access to the memory block could be preset to a certain amount. Once any processor is granted access the operating system should start the timer and at the end of the preset time check if the processor still has access to memory and if yes, then it should be removed at the end of the stipulated time making provision for other processors to gain access to the memory.

Chapter Six

Dynamic Node Level Reconfigurability and Multiple Forking Capability

6.1 Concept of Node Level Reconfigurability and Changes to HDCA

For the applications developed and described in Chapter 5, a static resource allocation algorithm is first executed to statically assign specific application process to specific processors within the HDCA system prior to execution of the application. However, in many cases especially in real time applications, unexpected events may occur. This may lead to a sudden increase of input values exceeding the limits assigned during static allocation. At this time it is required that additional copies of processes, on the fly, be assigned to existing or newly configured processors within the HDCA. This is the concept of node level reconfigurability.

In order to implement this concept in the HDCA system another stand by CE is introduced to the architecture. It triggers when both CE0 and CE1 get overloaded. The maximum queue depth of the processors is set to '8' (unsigned). However the threshold value is selected as '5' (unsigned). The value of the threshold is chosen less than the maximum that is '8' to avoid any loss of tokens that may occur in transition of issuing the token to a stand by CE rather than the overloaded one and also to provide sufficient tolerance to the system. The threshold for each of the CE0 and CE1 is set by feeding in two Load Threshold tokens as shown in Figure 6.1, for a list of all tokens refer [7]. The value of the threshold for both CEs is set to '5' (unsigned) by inputting tokens x"83E80005" for CE0 and x"82E80005" for CE1.

Load Threshold Token

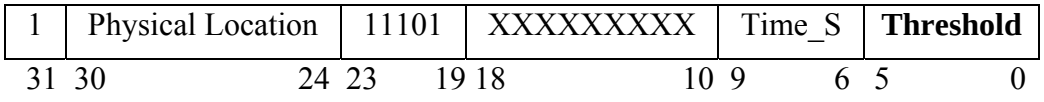

Figure 6.1, Load Threshold Token Fed Into HDCA Setting the Value of Threshold Flag

A design decision to trigger the stand by CE is made on the basis that when both the CEs reach their threshold value of '5' (unsigned). At this point the token instead of being issued to either of the CE0 or CE1 is being given to the stand by CE. The controller

of the PRT Mapper is modified, a check is put if the threshold value of both the CEs is reached and if true the new token is issued to the stand by CE.

The application used to describe and verify the concept of the node level reconfigurability is that explained in 5.1.5. In order to make the system compute intense 8 command tokens are being fed into the system instead of one as in 5.1.5. The waveforms are captured; whenever the threshold for the CE0 and CE1 is reached the stand by CE is triggered is observed.

The command tokens fed in the system are:

x”0101FF03” x”0121FF09” x”0141FF0F” x”0161FF15”
x”0181FF1C x”01A1FF21” x”01C1FF28” x”01E1FF2E”

The command tokens input into the system and the two load threshold tokens are shown in Figure 6.2. The tokens can be seen on “op_token_bus”. The “prog_flag” is set to ‘5’unsigned for CE0 at 6200 ns in the waveform.

Figure 6.3 shows that process 1 is being executed for the first four command tokens. The instructions can be seen at “db_pe_icm0_fin0” for CE0 and at “db_pe_icm1_fin1” for CE1. Also “prog_flag” for both the processors CE0 and CE1 is set to ‘5’ from the load threshold token. The signals “avlsig0”, “avlsig1” and “avlsig5” show the queue depth values of CE0, CE1 and stand by CE respectively. The values of the “avlsig0” and “avlsig1” vary from 0 to 2 as can be seen. The value does not increase beyond that since the process P1 is not complete for all the command tokens and hence have not yet forked to two processes each. As explained in 5.1.5 each process P1 forks to two processes in effect generating two command tokens issued to the PRT Mapper.

Figure 6.4 shows that forking has been done for some of the command tokens as can be seen from the instructions at “db_pe_icm0_fin0” and at “db_pe_icm1_fin1”. Most of the tokens are being issued to CE1 and its queue depth increases as can be seen from the Figure 6.4. It reaches the threshold value of ‘5’ and hence the threshold flag, “th_flag” for CE1 is set to ‘1’around 39.5 us. However, in this case since CE0 is not yet reached its threshold the stand by CE is not triggered

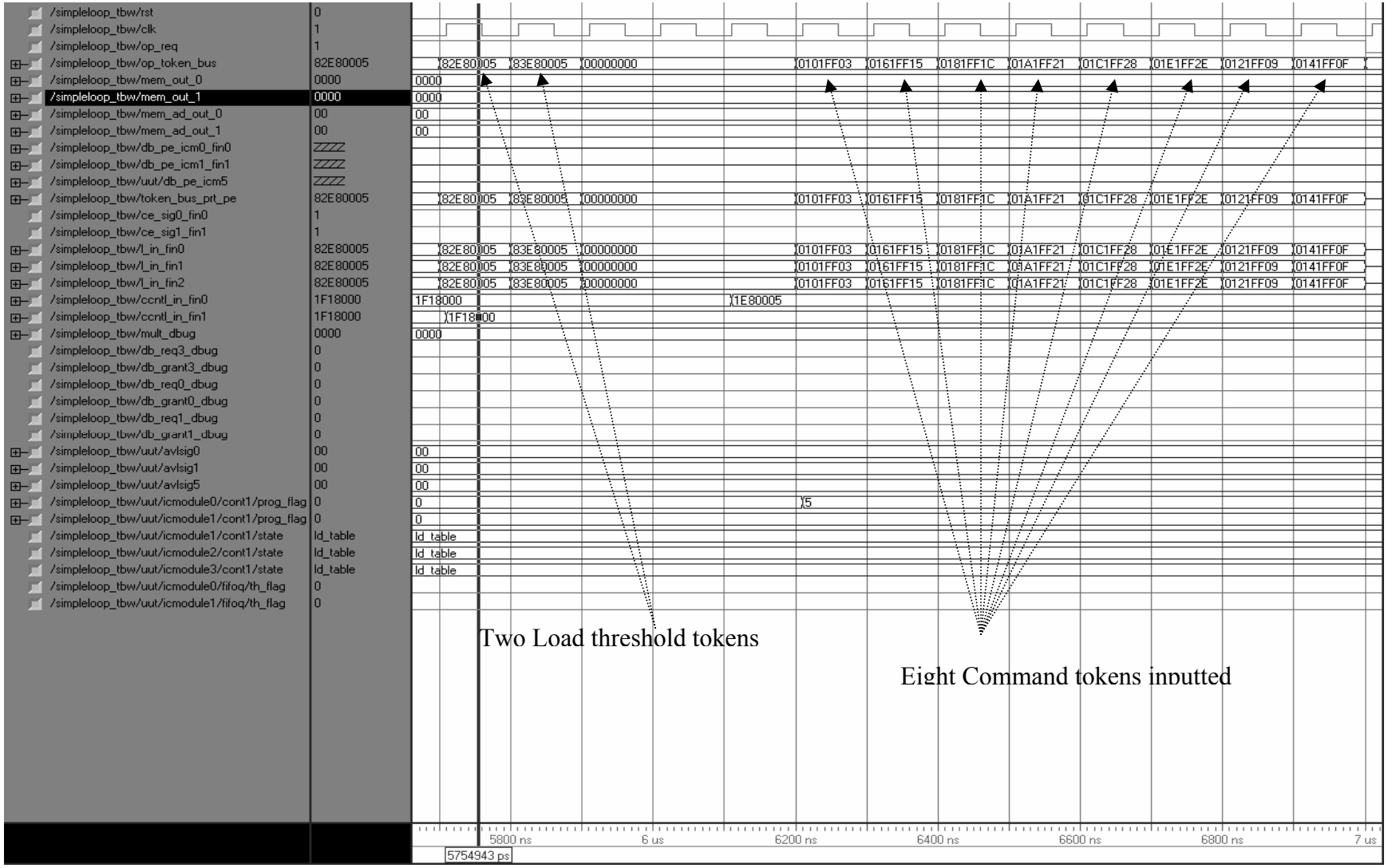

Figure 6.2, Input of two Load Threshold and Eight Command Tokens in the System

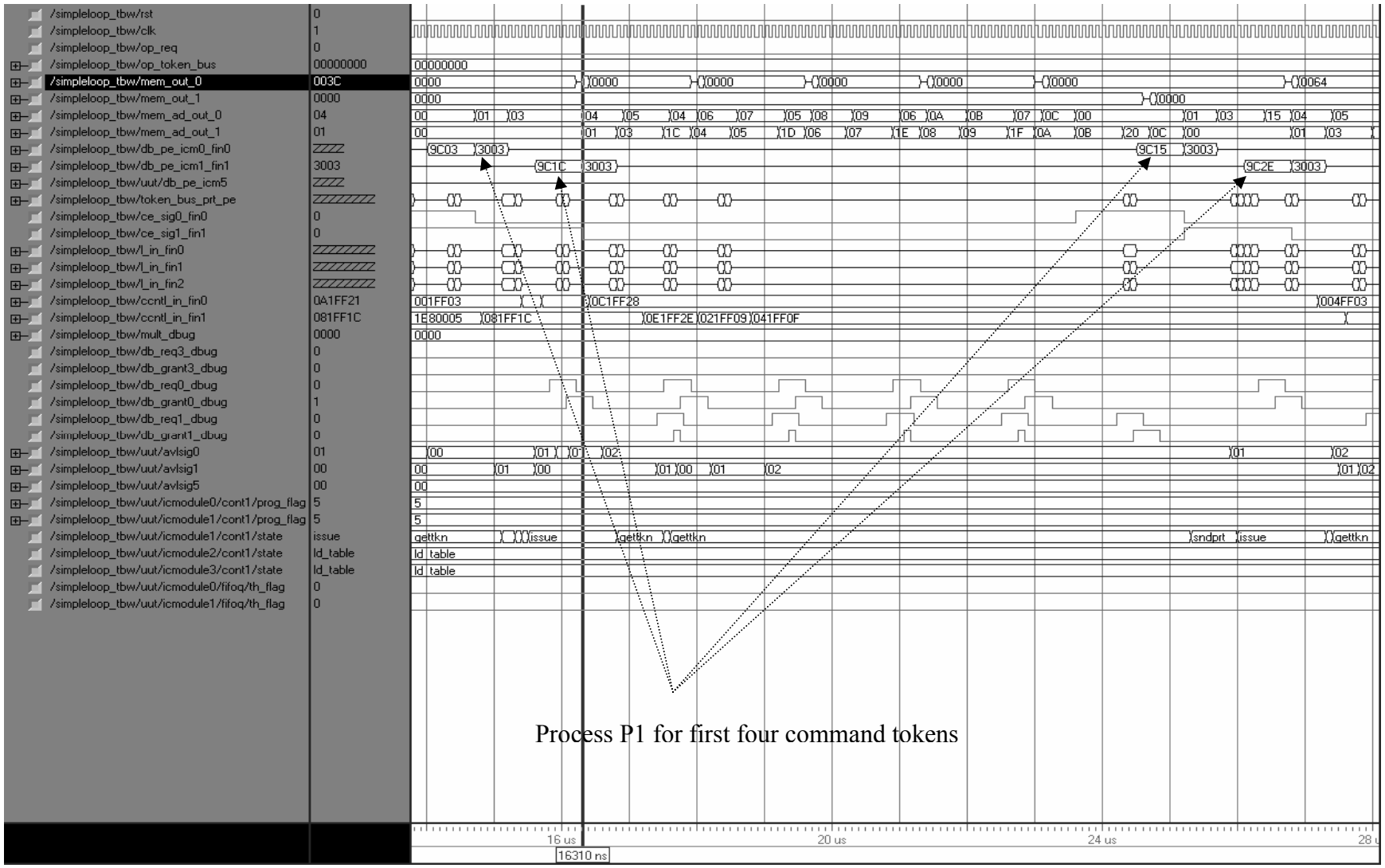

Figure 6.3, Process P1 for First Four Command Tokens

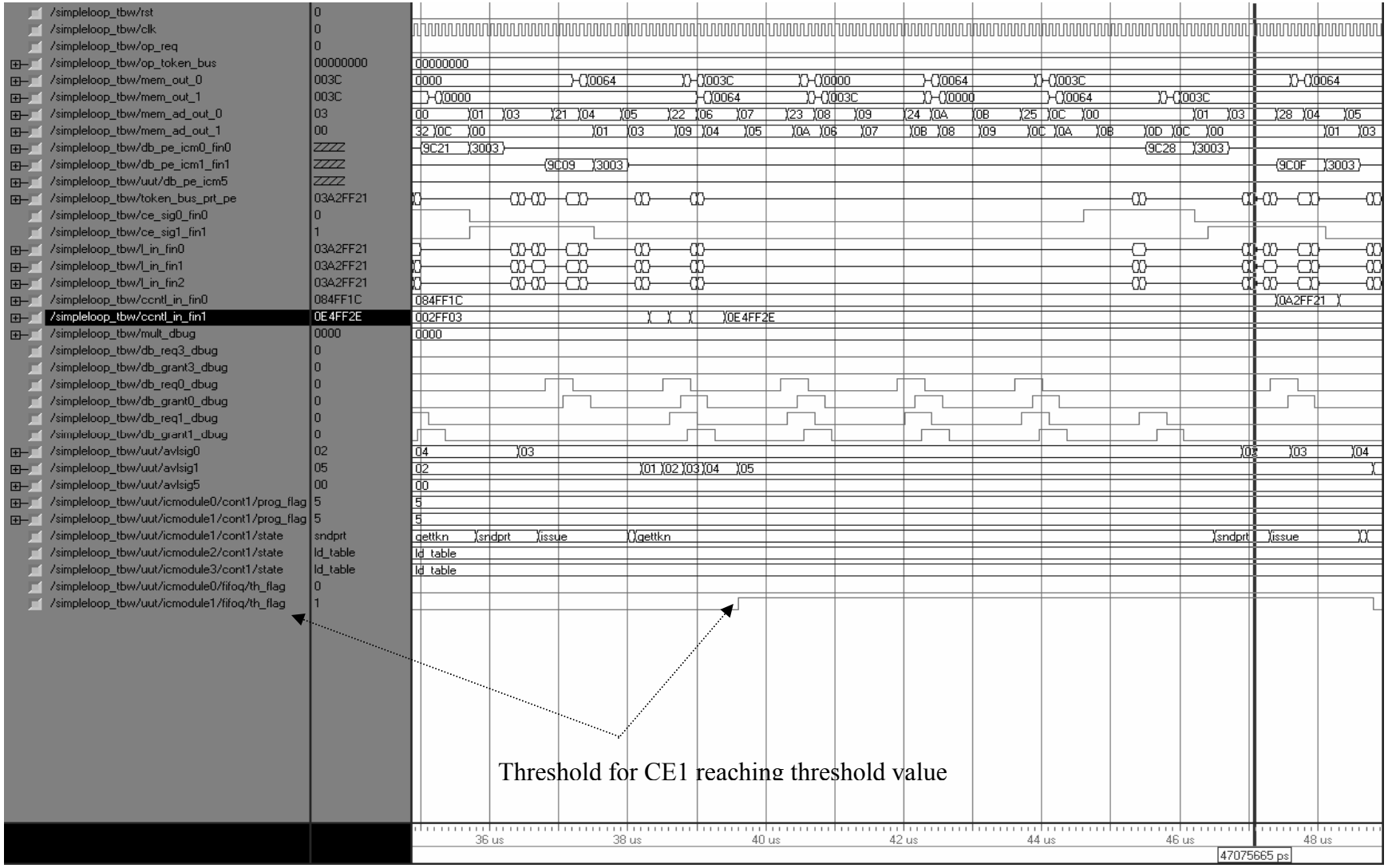

Figure 6.4, Forking of Tokens and Queue Depth of CE1 Reaching Threshold Value

The tokens generated after forking of P1 for all the eight command tokens are more or less distributed and assigned equally to the two CEs, CE0 and CE1. Hence at time 68 us as can be seen in Figure 6.5, both the threshold flags are set to '1', now the stand by CE gets triggered. It can be observed that the following token which is x" 0645FF0F" for process five of the third command token is being issued to stand by CE instead of being issued to CE0 or CE1. In that case the tokens would have been x"0345FF0F" and x"0245FF0F" for CE0 and CE1 respectively. It can also be shown that "avlsig5" goes from "00" to "01" and returns to "00" once the process starts executing on the stand by CE. The token issued to the stand by CE can be observed on "token_bus_prt_pe".

During the course of execution of the application this situation arises many times and stand by CE is triggered to take off the extra load of the CEs. Another such situation is shown in Figure 6.6. The threshold for both the CEs sets to '1' at around 80.5 us and the next following token is being issued to the stand by CE. The token x"0685FF1C" is seen on "token_bus_prt_pe".

Thus it can be concluded from the waveforms that the design modifications to the HDCA in order to incorporate the node level reconfigurability works correctly.

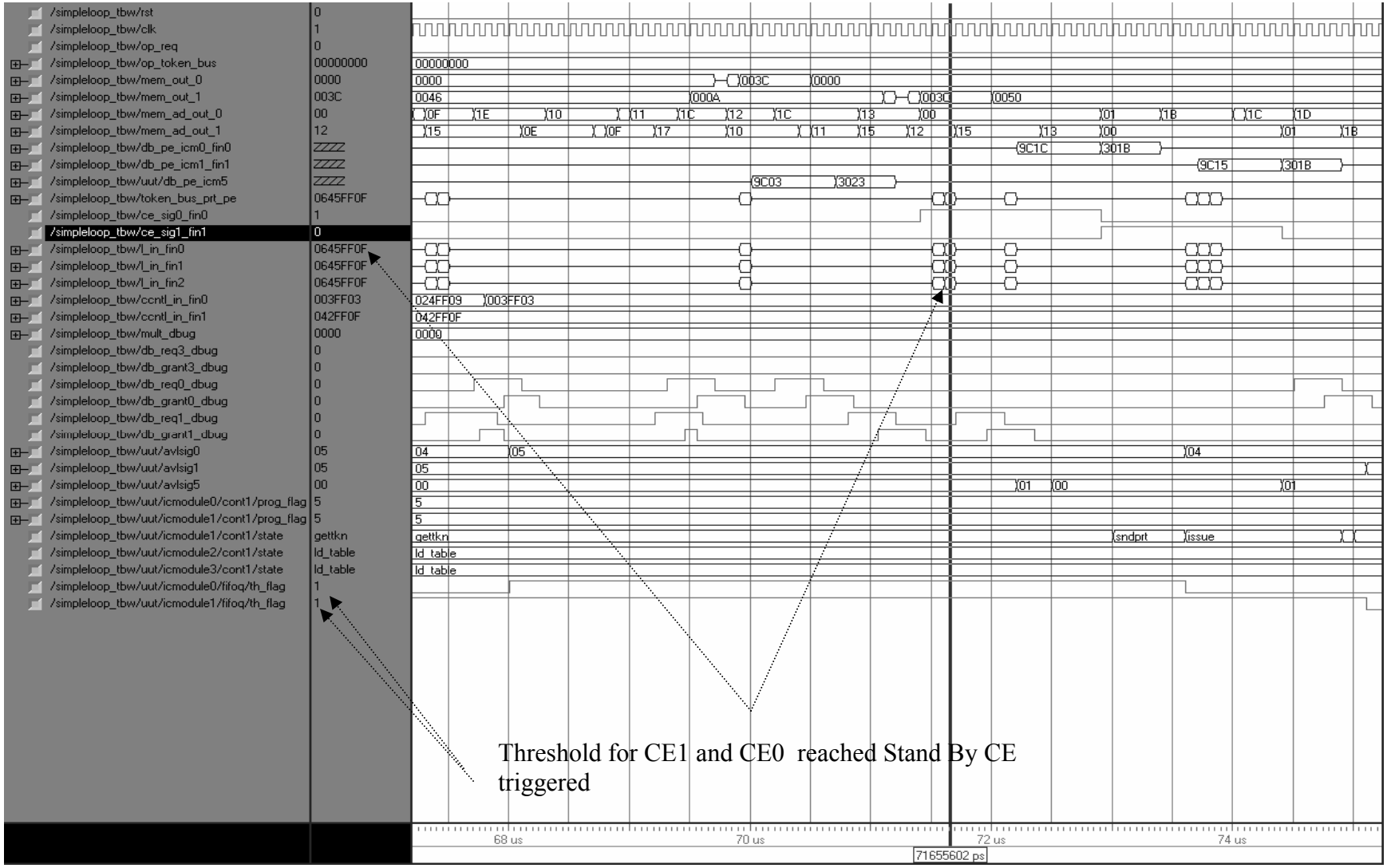

Figure 6.5, Threshold Flag Set for CE0 and CE1, Stand by CE Triggered

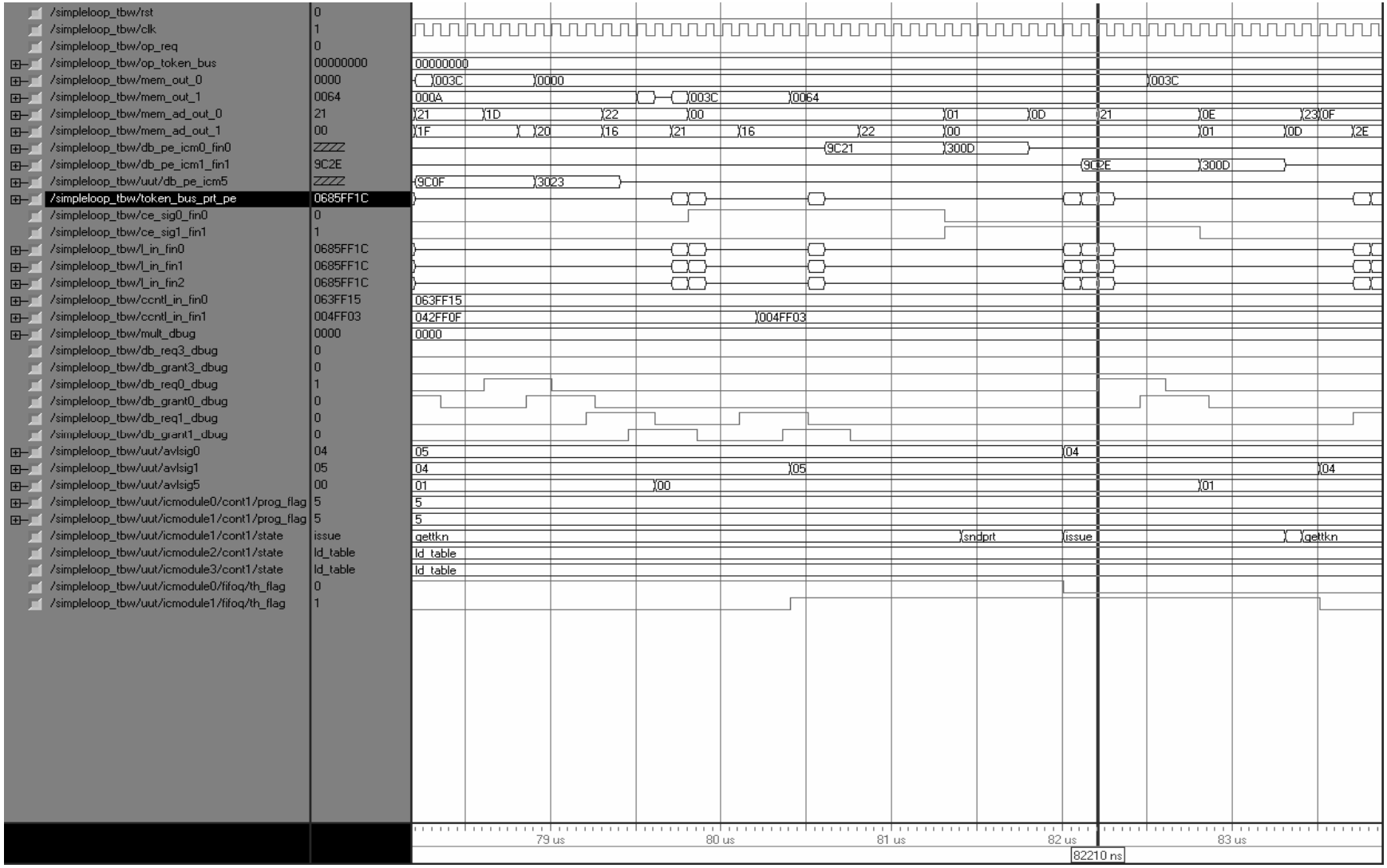

Figure 6.6, Threshold Flag Set 2nd Time for CE0 and CE1, Stand by CE Triggered

6.2 Multiple Forking Capability of the HDCA System:

The applications described thus far as in chapter 5 had a major limitation in terms of parallelism; processes being executed simultaneously. The number of processes that a single process could fork to is limited to two. One of the ways this problem could be overcome is described here.

It can be seen from the Figure 6.6 that a process that forks into three processes is initially divided into two main processes one of which is the dummy process that just helps to fork the processes further into two processes. In the dummy process the processor goes into a "no-op" state causing a delay and forking into two processes. This can be seen as a "cascading effect", with this method a process can be made to fork to 'n' number of processes.

In order to incorporate this concept into the HDCA system, an additional state 'OP13' is added to the controller of the PE. This state as stated earlier is a 'no-op' state that basically forks into two processes. The code for the additional state is included in the Appendix B.

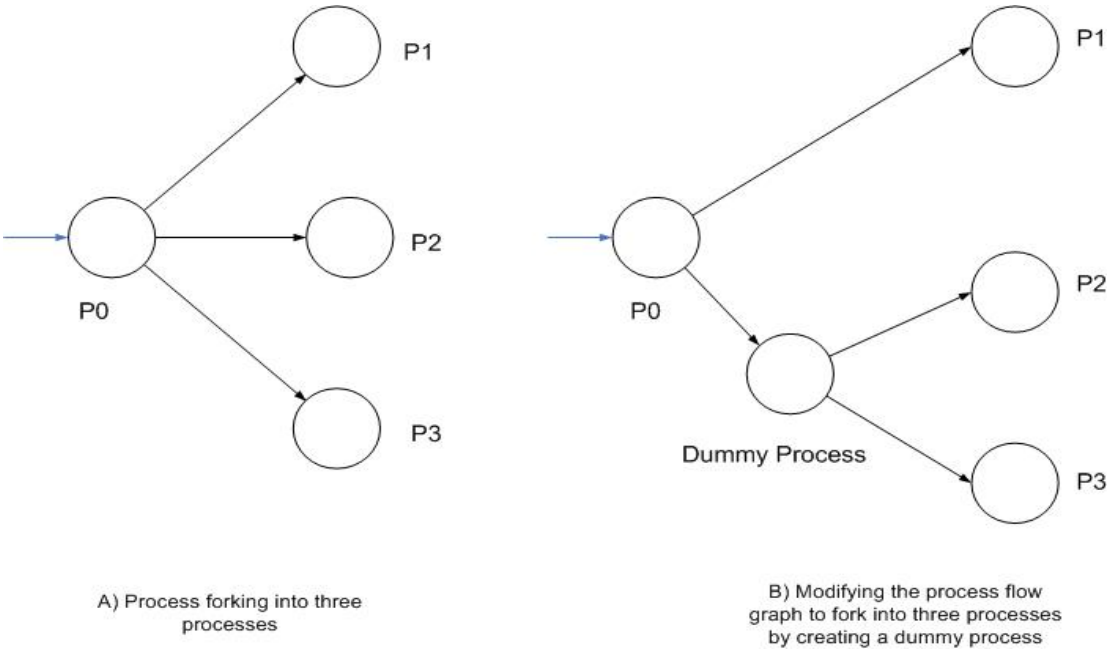

Figure 6.6, Multiple Forking Concept Used in the HDCA System

6.3 Application Describing Multiple Forking in HDCA System:

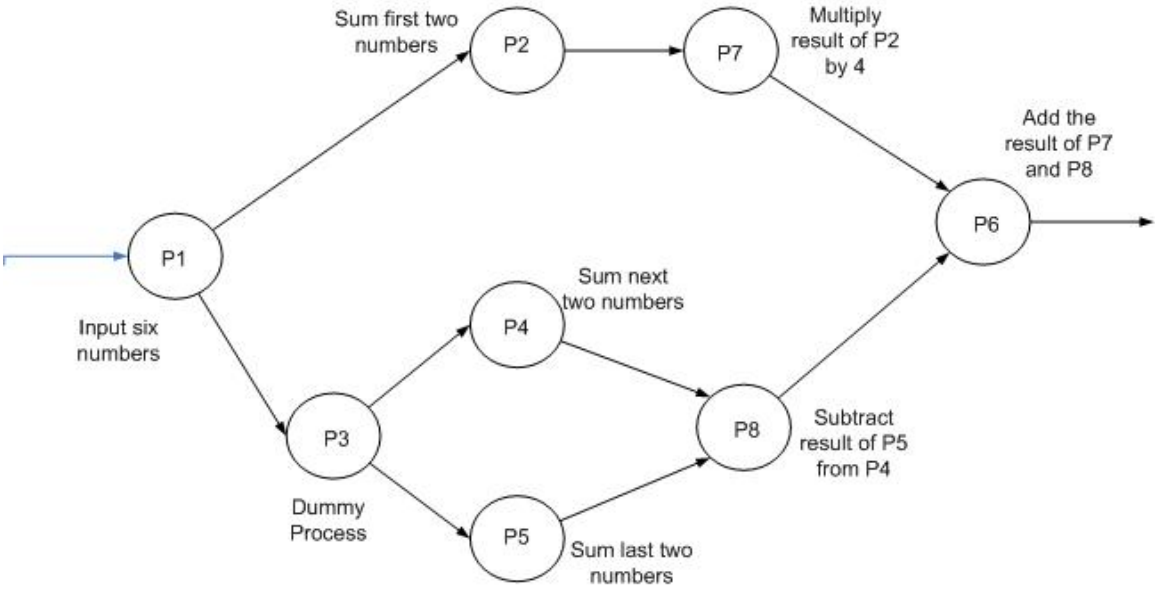

Figure 6.7, Application Describing Multiple Forking in HDCA

Each process shown in the Figure 6.7 is explained in more detail as follows:

P1 – Input 6 numbers into the system.

P2 – Sum the first two numbers inputted.

P3 – Dummy Process that forks into processes P4 and P5.

P4 – Sum of the middle two numbers that is 3rd and 4th numbers inputted.

P5 - Sum of last two numbers inputted.

P8 – Subtract the result of P5 from result of P4.

P7 – Multiply the result of process P2 with '4' (unsigned).

P8 – Add the result of process P7 and P8 and display.

A series of tokens are inputted into the system and finally a command token x"01010003" is fed in, as shown in Figure 6.8

Figure 6.9 shows the instruction for the first process P1 "9C03 3003". Six numbers are inputted into the system all unsigned '2's in this case. The end of process P1 results in three processes, P2, P4 and P5 to achieve this forking as discussed earlier P1 forks to P2 and P3. P3 is a dummy process which facilitates further forking into P4 and

P5 as shown in Figure 6.7. The Figure 6.10 shows the execution of process P2, it is addition of unsigned ‘2’ at locations x”03” and x”04”. The result unsigned ‘4’ is stored at location x”0A”. The Figure also shows the token for process P3 x”02030003”.

Figure 6.11 shows the execution of process P3 which is a “no-op” with instruction “9C03 3000” and also execution and result of process P7. Process P7 is a multiplication process the result of process P2 which is unsigned ‘4’ is multiplied by unsigned ‘4’. The result unsigned ‘16’ is stored at x”0A”. It can be viewed at “mem_out_3” and at location “mem_ad_out_3”.

The execution of dummy process P3 results in two processes P4 and P5. The Figure 6.12 shows the instructions for process P4 x”9C03 3018” and x”9C03 3020” for process P5. The result for both the processes can be observed unsigned ‘4’ (“mem_out_1”) at locations x”14” for process P4 and unsigned ‘4’ (mem_out_1”) at x”1E” for process P5.

Figure 6.13 shows the instruction x”9C03 9803 3028” for the join operation of process P8 as shown in Figure 6.7. It is a subtraction operation, the result of process P4 is subtracted from result of P5. Hence the value of the subtraction operation is unsigned ‘0’ (unsigned ‘4’ at x”14”subtracted from unsigned ‘4’ at x”1E”). The result is ‘0’ (unsigned) is stored at location x” 28”. The Figure also shows the token for the join process P6 being issued to the PRT Mapper by the CE, x”81060003”.

The instruction for final process P6 x”9C03 9803 3030”is shown in Figure 6.14. The process consists of addition operation; the result of P7 is added to the result of P8. The result unsigned ‘16’ (unsigned ‘16’ at location x”0A” added to unsigned ‘0’at x”28”) is stored at location x”3C” as can be seen at “mem_out_1”.

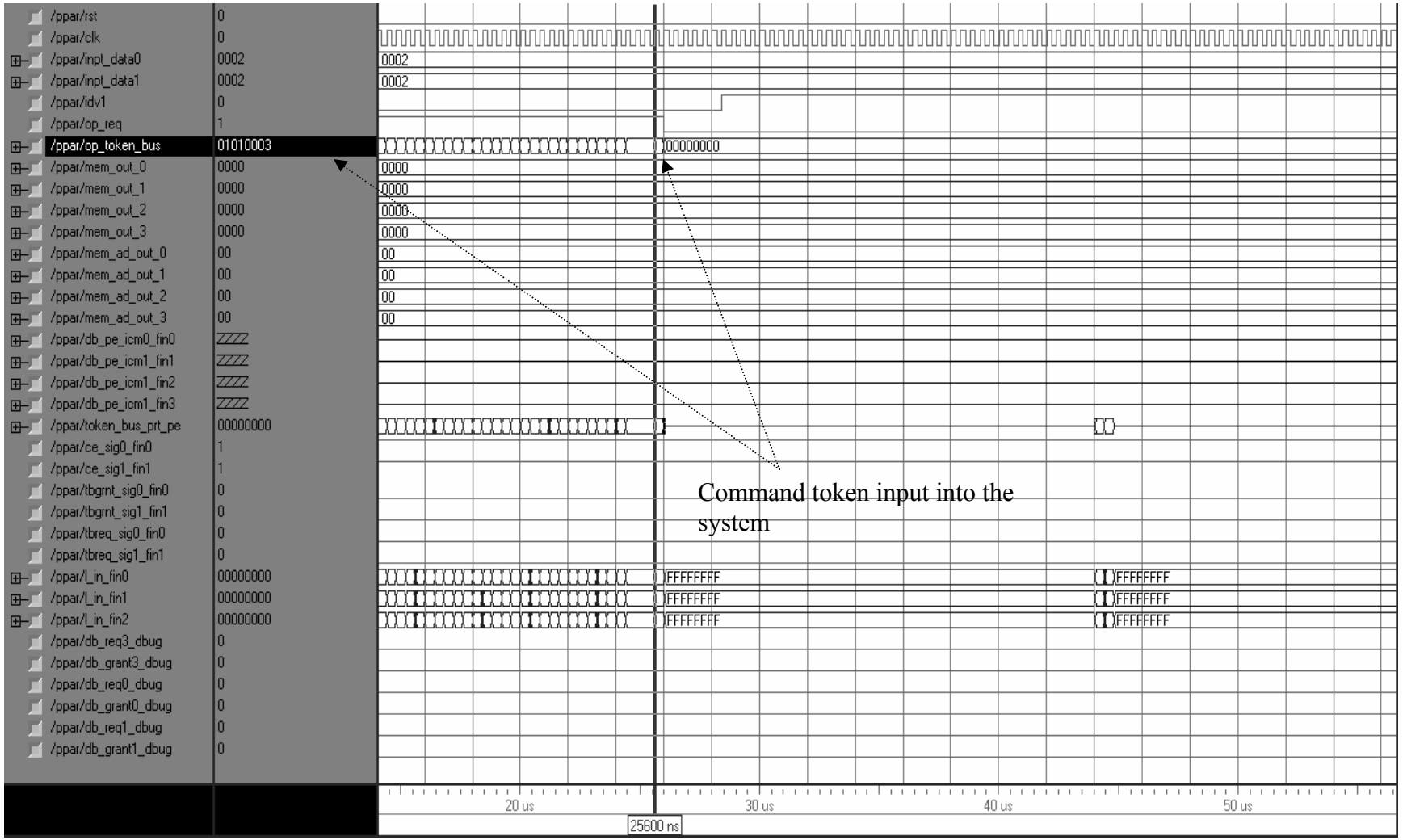

Figure 6.8, Command Token Input into the System

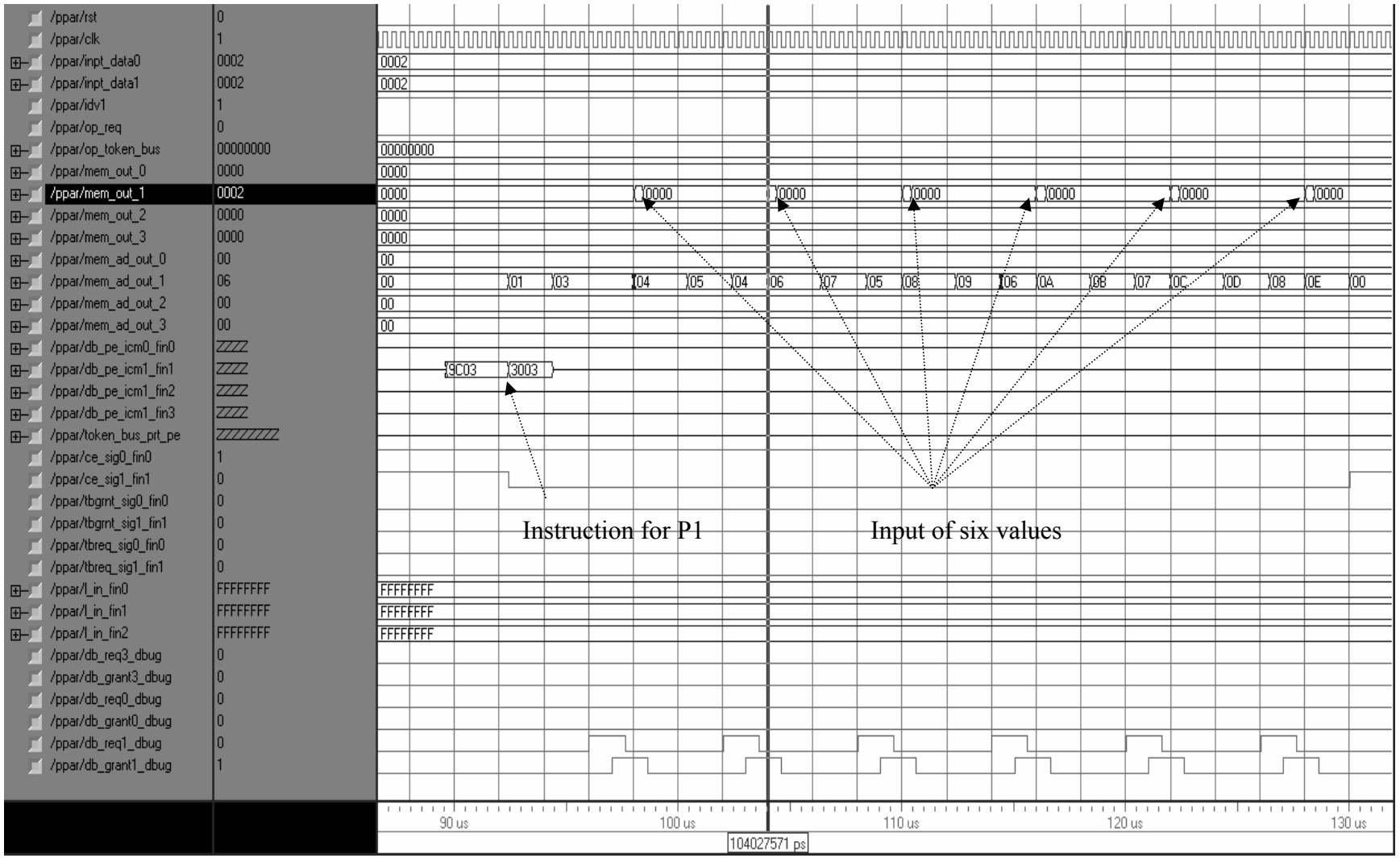

Figure 6.9, Instruction for Process P1

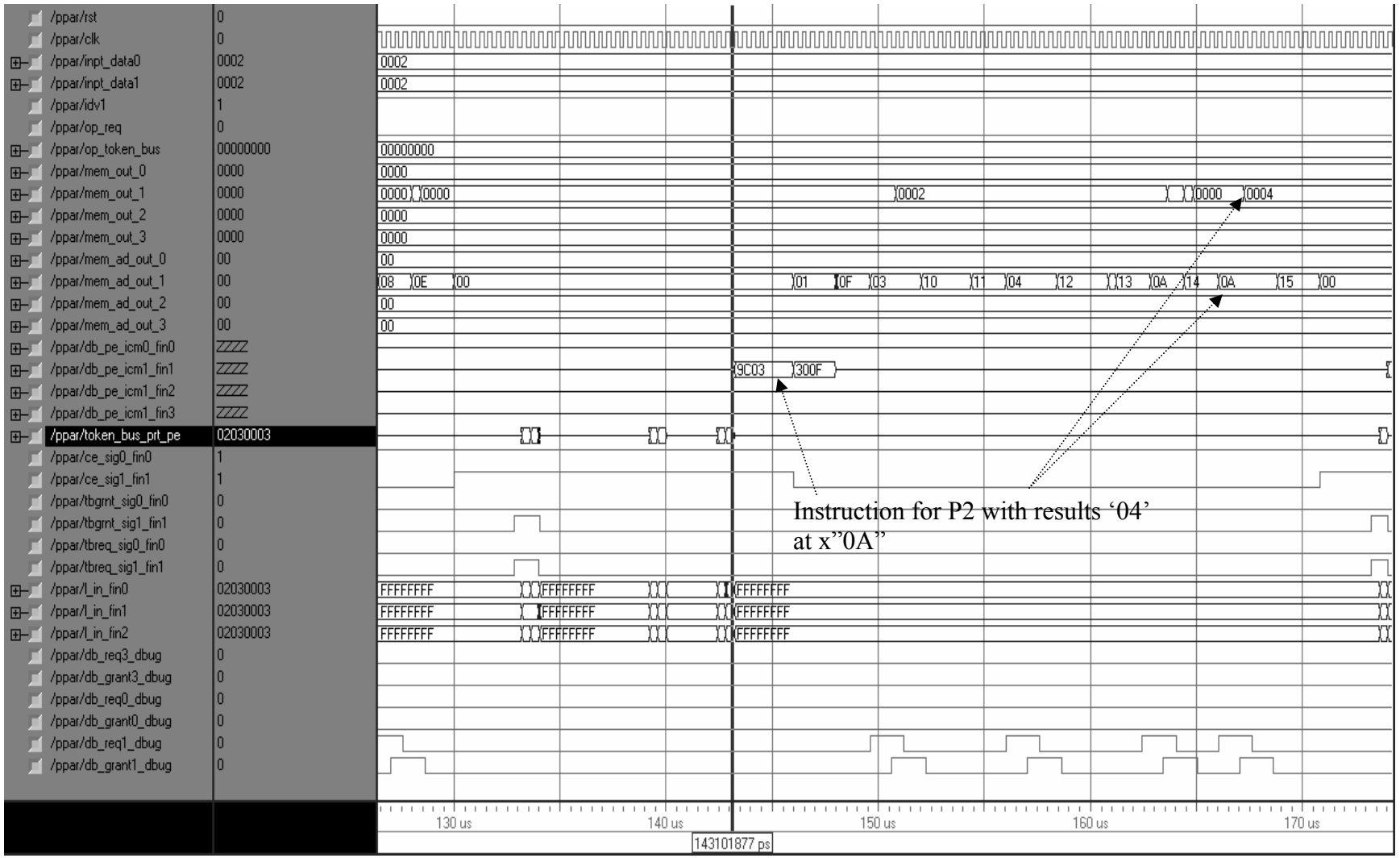

Figure 6.10, Instruction for P2 with Result and Command Token for P3

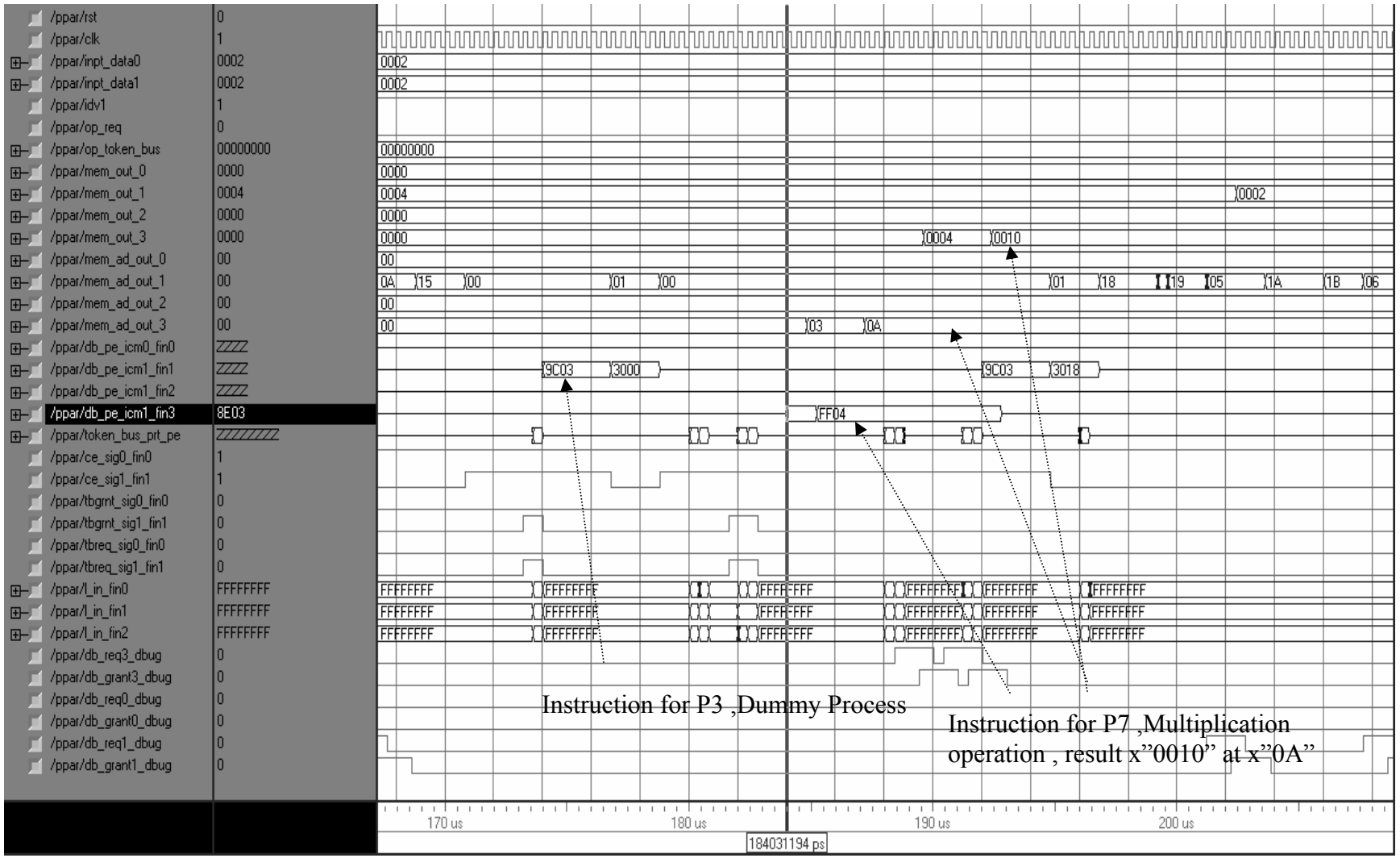

Figure 6.11, Process P3 and Process P7 Execution and Results

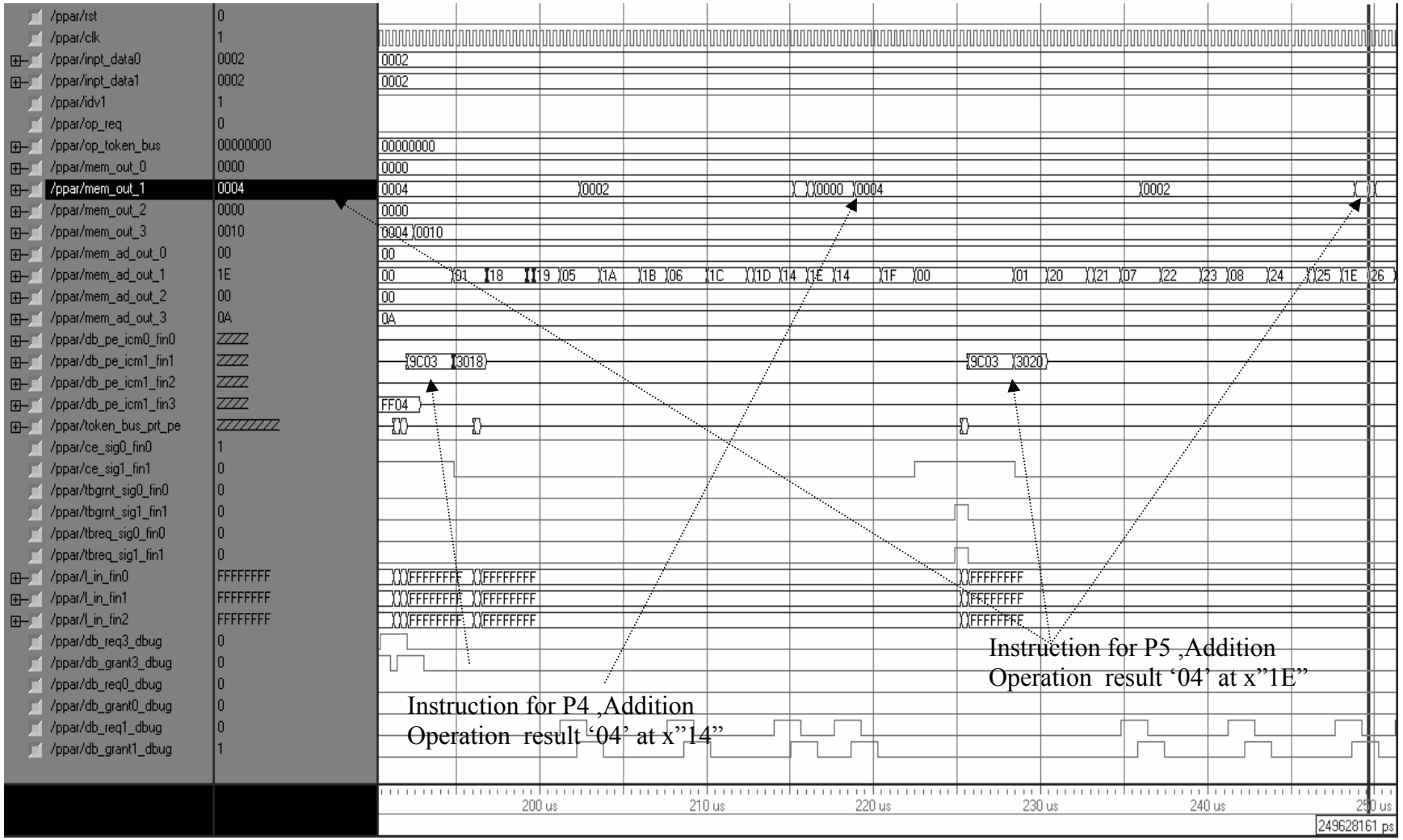

Figure 6.12, Instruction for Process P4 and P5 with Results

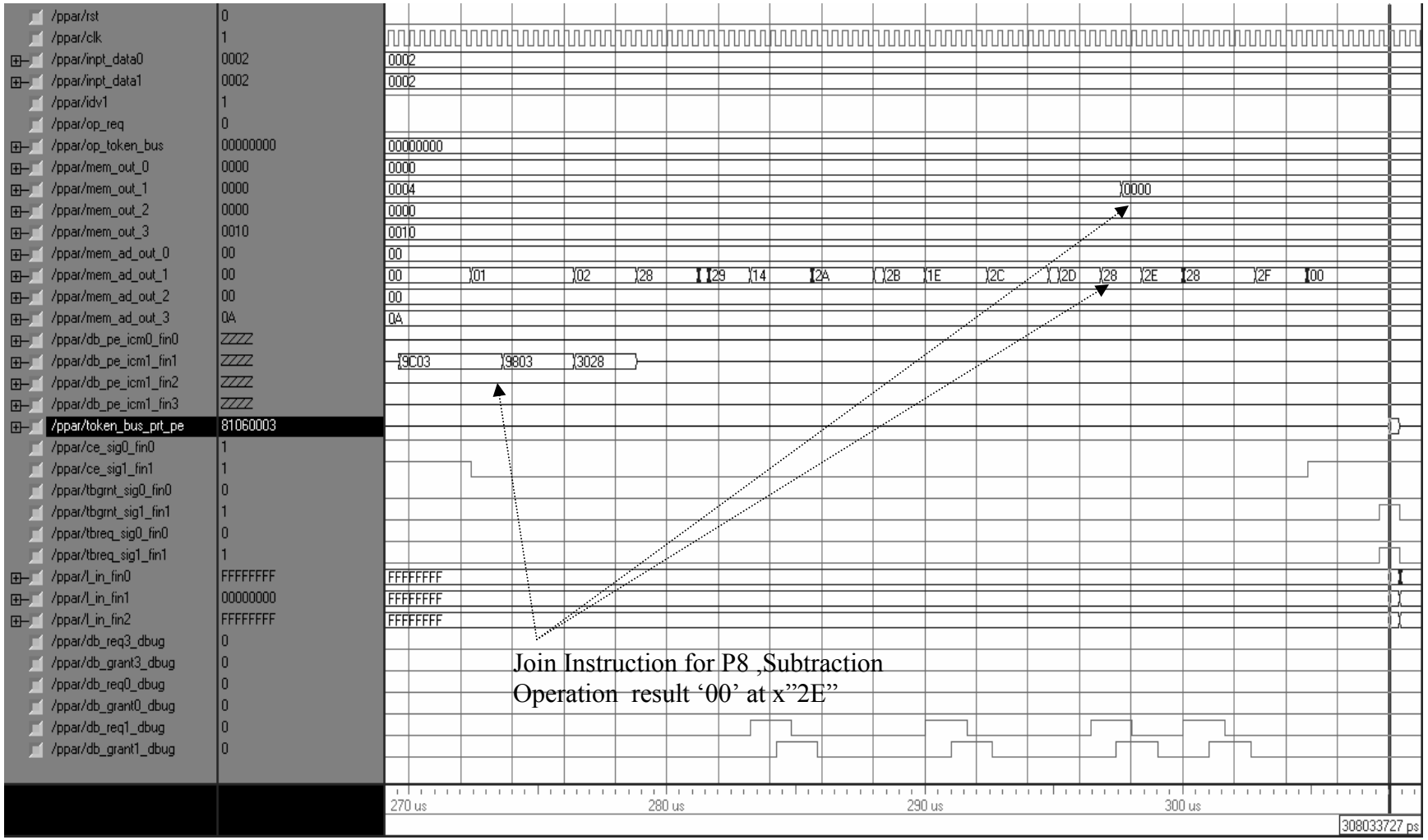

Figure 6.13, Join Instructions for Process P8 with Result

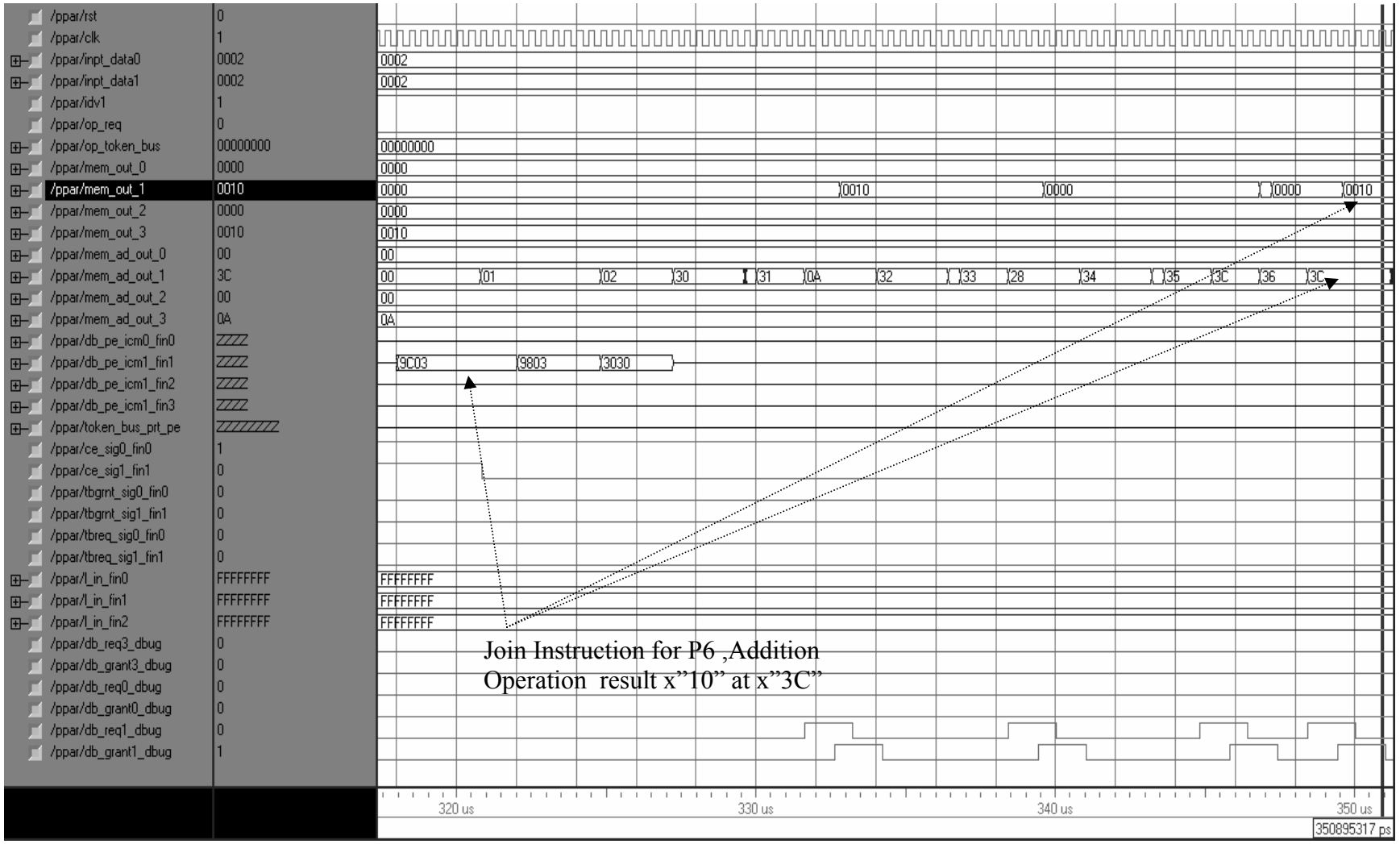

Figure 6.14, Final Process P6 Join Operation and Result

Chapter Seven

FPGA Resources Utilized in HDCA Virtual Prototype Development and Testing Environment

The VHDL coded interconnect network and the entire HDCA system with crossbar switch integrated in it [Appendix A1 and A2] are synthesized and virtually post placed and routed using Xilinx 6.2.3i CAD Tool and Modelsim 5.7g as a simulation tool. The term 'virtual prototype' signifies that the system is validated using the Modelsim Simulator and shows correct results for the 'Simulate Post-Place and Route VHDL Model'. This is the final step before actually downloading the system design onto a prototype board.

The design is tested using Xilinx XC2V8000 from the Virtex II family of chips.It has 8 million gates configuration. The entire design development, testing and validation is done on a system with following parameters: Intel Pentium 4, 3.00 Ghz with 1 GB of RAM. The operating system used is Microsoft Windows XP, service pack 2.

Among the various applications developed with the crossbar switch embedded in the HDCA system, two are tested for the post place and route simulation.

The resource utilization and timing summary for aplication 1: Acyclic Application shown in Figure 5.12 is given below:

Device utilization summary:

Number of External IOBs	717 out of 824	87%
Number of LOCed External IOBs	0 out of 717	0%
Number of MULT18X18s	1 out of 168	1%
Number of RAMB16s	9 out of 168	5%
Number of SLICEs	12429 out of 46592	26%
Number of BUFGMUXs	1 out of 16	6%
Number of TBUFs	908 out of 23296	3%

Total equivalent gate count for design: 874,228

Timing Summary:
Speed Grade: -5

Minimum period: 21.516ns (Maximum Frequency: 46.477MHz)
Minimum input arrival time before clock: 9.415ns

Maximum output required time after clock: 14.407ns
Maximum combinational path delay: 8.562ns

The resource utilization for application 2: Application having Multiple Forking Capability as shown in Figure 6.7 is given below:

Device utilization summary:

Number of External IOBs	727 out of 824	88%
Number of LOCed External IOBs	0 out of 727	0%
Number of MULT18X18s	1 out of 168	1%
Number of RAMB16s	9 out of 168	5%
Number of SLICEs	13912 out of 46592	29%
Number of BUFGMUXs	1 out of 16	6%
Number of TBUFs	908 out of 23296	3%

Total equivalent gate count for design: 895,077

Timing Summary:
Speed Grade: -5

Minimum period: 21.516ns (Maximum Frequency: 46.477MHz)
Minimum input arrival time before clock: 12.957ns
Maximum output required time after clock: 14.407ns
Maximum combinational path delay: 8.562ns

Chapter Eight

Conclusion

A modular and scalable architecture and design for a crossbar interconnect network of a HDCA single chip multiprocessor system is presented. The design capture, synthesis, simulation is done in VHDL using XILINX ISE 6.2.3i and ModelSim 5.7g CAD soft wares. The design is individually validated and integrated in the main HDCA system and validated again against two varied applications. The inclusion of crossbar switch in the HDCA architecture involved major modifications in the HDCA system and some minor changes in the design of the switch. The results show perfect functioning of the crossbar network. Dynamic Node Level reconfigurability feature added to enhance the HDCA capability is also tested against the acyclic application and shows proper functioning. The architecture is limited in terms of a process forking to maximum of two processes; this shortcoming is overcome and is tested by an application and exhibits perfect functioning of the system.

Building up a complete full proof architecture with all the capabilities is an on going process. With the work done in this thesis certain goals are achieved as listed above and described in detail in the earlier chapters; however there is still lot of work to be done before architecture could be used for more serious applications. One of the biggest challenges lies in the development of an Operating system. Some of the important functions that the operating system [3] should do are to perform all initialization operations and monitor the entire system in general and also detect failures, bottlenecks and quickly reconfigure the system to overcome the problem. With the integration of interconnect switch in the system the operating system should keep track of which processors are requesting access to which memory blocks on timely basis so that the condition of data incoherency does not arise.

Appendices

Appendix A1:Post Place and Route VHDL Code For Functional Model of the Interconnect Network

Module Name:main.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity main is

generic ( N : integer := 3;  -- Number of inputs and outputs
          P : integer := 15; -- qdepth value it is actually four times the value of qdepth for each q depth
          K : integer := 15; -- address bus combined (summation ) of all the processors
          Q : integer := 15); -- data bus width input and output

   port ( clk: in std_logic ;
        rst: in std_logic ;
        ctrl: in std_logic_vector(N downto 0) ;
        qdep: in std_logic_vector(P downto 0) ;
        addr_bus: in std_logic_vector(K downto 0) ;
        data_in : in std_logic_vector(Q downto 0) ;
        rw: in std_logic_vector(N downto 0) ;
        flag: inout std_logic_vector(N downto 0) ;
        data_out: out std_logic_vector(Q downto 0) );

end main ;

architecture test_main of main is

type qd is array (N downto 0) of std_logic_vector(N downto 0) ;
type data_array is array (N downto 0) of std_logic_vector(N downto 0) ;
type addr_array is array (N downto 0) of std_logic_vector(N downto 0) ;
type mb is array (N downto 0) of std_logic_vector(1 downto 0) ;
type mem_array is array (K downto 0) of std_logic_vector(N downto 0) ;



-- This function does the priority logic for all the memory blocks
-- This can work for any number of processors and memory blocks
```

```
-- by changing 'i' and 'j' values



function flg (qdep, addr_bus, ctrl:std_logic_vector ) return
std_logic_vector is

variable qdvar: std_logic_vector (N downto 0) ;
variable flag: std_logic_vector(N downto 0) ;
variable qdv : std_logic_vector(N downto 0) ;
variable gnt : std_logic ;
variable a: integer range 0 to N;
variable b: integer ;
variable memaddr : mb ;
variable qd_arr : qd ;


begin


  qd_arr(0) := qdep(3 downto 0) ;
  qd_arr(1) := qdep(7 downto 4) ;
  qd_arr(2) := qdep(11 downto 8) ;
  qd_arr(3) := qdep(15 downto 12) ;

  memaddr(0) := addr_bus(3 downto 2) ;
  memaddr(1) := addr_bus(7 downto 6) ;
  memaddr(2) := addr_bus(11 downto 10) ;
  memaddr(3) := addr_bus(15 downto 14) ;


L1: for i in 0 to N loop

L2: for j in 0 to N loop

     if (ctrl(j) = '0') then
       flag(j) := '0' ;
        qdv(j) := '0' ;
     elsif (memaddr(j) = i) then
          qdv(j) := '1' ;
     else
          qdv(j) := '0' ;
     end if ;
end loop L2 ;

qd_var_loop : for i in 0 to N loop
qdvar(N) := '0';
end loop qd_var_loop;


gnt := '0' ;

L3: for k in 0 to N loop

     if qdv(k) = '1' then
       if qdvar <= qd_arr(k) then
```

```
            qdvar := qd_arr(k) ;
            a := k ;
            gnt := '1' ;
        else
            flag(k) := '0' ;
        end if;
      end if ;
end loop L3 ;


if (gnt = '1') then
flag(a) := '1' ;

end if ;


end loop L1 ;
return (flag) ;

end flg;

signal memory: mem_array ;


begin
P1 : process(ctrl, clk, qdep, addr_bus, rst, data_in) is

begin

if (rst = '1') then
flag_loop : for i in 0 to N loop
flag(N) <= '0';
end loop flag_loop;

data_out_loop : for i in 0 to Q loop
data_out(Q) <= '0';
end loop data_out_loop;

else

flag <= flg(qdep, addr_bus, ctrl) ;

-- Memory transaction
-- The conditional statements make sure that the connection is established
-- before memory transaction
-- This routine is to be repeated for each addition of processor

if (clk 'event and clk = '0') then

if (flag(0) = '1') then

if (rw(0) = '1') then
memory(conv_integer(addr_bus(3 downto 0))) <= data_in(3 downto 0) ;
data_out(3 downto 0) <= (others => 'Z') ;
--data_out(3 downto 0) <= memory(conv_integer(addr_bus(3 downto 0))) ;
else
```

```
data_out(3 downto 0) <= memory(conv_integer(addr_bus(3 downto 0))) ;
end if ;
end if ;

if (flag(1) = '1') then

if (rw(1) = '1') then
memory(conv_integer(addr_bus(7 downto 4))) <= data_in(7 downto 4) ;
data_out(7 downto 4) <= (others => 'Z') ;
else
data_out(7 downto 4) <= memory(conv_integer(addr_bus(7 downto 4))) ;
end if ;
end if ;

if (flag(2) = '1') then

if (rw(2) = '1') then
memory(conv_integer(addr_bus(11 downto 8))) <= data_in(11 downto 8) ;
data_out(11 downto 8) <= (others => 'Z') ;
else
data_out(11 downto 8) <= memory(conv_integer(addr_bus(11 downto 8))) ;
end if ;
end if ;

if (flag(3) = '1') then

if (rw(3) = '1') then
memory(conv_integer(addr_bus(15 downto 12))) <= data_in(15 downto 12) ;
data_out(15 downto 12) <= (others => 'Z') ;
else
data_out(15 downto 12) <= memory(conv_integer(addr_bus(15 downto 12)))
;
end if ;
end if ;

end if;
end if;

end process P1 ;

end test_main ;
```

Appendix A2:Post Place and Route VHDL Code For Acyclic Applications

Module Name:entirenew.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity entiresystry2 is

Port (rst,clk:in std_logic;
      inpt_data0,inpt_data1:in std_logic_vector(15 downto 0);
      idv0,idv1:in std_logic;
      op_req:in std_logic;
      Op_Token_bus: in STD_LOGIC_VECTOR (31 downto 0);
      Mem_out_0,Mem_out_1,Mem_out_2,Mem_out_3: out std_logic_vector ( 15 downto 0);
      Addr_en: in std_logic;
      mem_ad_out_0,mem_ad_out_1,mem_ad_out_2,mem_ad_out_3:out std_logic_vector(6 downto 0);
       R3_out_dbug_fin0,R3_out_dbug_fin1 : out std_logic_vector( 15 downto 0);
       shft_out_dbug_fin0,shft_out_dbug_fin1 : out std_logic_vector( 15 downto 0 );
       dbug_st_pe_fin0,dbug_st_pe_fin1 : out std_logic_vector( 3 downto 0);
       dbus_sig0_fin0,dbus_sig1_fin1,dbus_sig2_fin2 : out std_logic_vector (15 downto 0);
       dataout_lut_fin0,dataout_lut_fin1,dataout_lut_fin2,dataout_lut_fin3:out std_logic_vector(15 downto 0);
       db_pe_icm0_fin0,db_pe_icm1_fin1,db_pe_icm1_fin2,db_pe_icm1_fin3 : out std_logic_vector( 15 downto 0) ;
       R0_out_dbug_fin0,R0_out_dbug_fin1 : out std_logic_vector(15 downto 0);
       token_bus_prt_pe : out std_logic_vector (31 downto 0);
       Wr_out_dbug0_fin0,Wr_out_dbug1_fin1 : out std_logic_vector( 1 downto 0);
       ce_sig0_fin0,ce_sig1_fin1: out std_logic;
       tbgrnt_sig0_fin0,tbgrnt_sig1_fin1 : out std_logic;
       tbreq_sig0_fin0,tbreq_sig1_fin1 : out std_logic;
       i_rdy_icm0_fin0,i_rdy_icm1_fin1 : out std_logic ;
       snd_i_icm0_fin0,snd_i_icm1_fin1 : out std_logic;
        l_in_fin0,l_in_fin1,l_in_fin2 : out std_logic_vector(31 downto 0);
        contrl_0,control_1 : out std_logic_vector( 3 downto 0);
        x_dbug_fin0,x_dbug_fin1,x_dbug_fin3 : out std_logic_vector(6 downto 0);
        dloutfin0,dloutfin1:out std_logic_vector(15 downto 0);
        count_dbug0,count_dbug1,count_dbug3:out std_logic_vector(6 downto 0);
        db_req3_dbug,db_grant3_dbug : out std_logic;
            db_req0_dbug,db_grant0_dbug : out std_logic;
        db_req1_dbug,db_grant1_dbug : out std_logic;
        RLTable0,RLTable1,RLTable2,RLTable3: out std_logic_vector( 1 downto 0);
        dwr0,dwr1,dwr2,dwr3: out std_logic;
        tabin0,tabin1,tabin2,tabin3: out std_logic;
        temp3_ce0,temp3_ce1 :out std_logic_vector(2 downto 0);
        temp2_ce0,temp2_ce1 :out std_logic_vector(1 downto 0);
        temp1_ce0,temp1_ce1 :out std_logic_vector(1 downto 0);
        temp4_ce0,temp4_ce1 :out std_logic_vector(4 downto 0);
            temp5_ce0,temp5_ce1 :out std_logic_vector(3 downto 0);
             count_ce1 : out std_logic_vector (7 downto 0)
```

```
end entiresystry2;

architecture Behavioral of entiresystry2 is
--Begin components used in this module

--PE3/CE0 component

component PE is
  port (  Data_Bus : inout std_logic_vector(15 downto 0);
          R_W : out std_logic;
          Cntl_bus : in std_logic_vector(15 downto 0);
                  RST, ODR, IDV : in std_logic;
                  clk, Bus_grant : in std_logic;
               CInstr_rdy : in std_logic;
                  inpt : in std_logic_vector(15 downto 0);
                  Bus_req, Snd_Instr, Fin : out std_logic;
                Addr : out std_logic_vector(7 downto 0);
                Rq_inpt, Rq_outpt : out std_logic;
                STOPLOOP : out std_logic;
                  -- added for dbugging
               R3_out_dbug : out std_logic_vector( 15 downto 0);
                  shft_out_dbug : out std_logic_vector( 15 downto 0 );
                  dbug_st_pe : out std_logic_vector( 3 downto 0);
                  tmp4_dbug : out std_logic_vector(15 downto 0);
                  m5outdbg: out std_logic_vector(15 downto 0);
                  R0_out_dbug : out std_logic_vector(15 downto 0);
                  tmp3_dbug: out std_logic_vector(2 downto 0);
                  tmp2_dbug: out std_logic_vector(1 downto 0);
                  tmp1_dbug: out std_logic_vector(1 downto 0);
                  tmp44_dbug: out std_logic_vector(4 downto 0)          ;
                  tmp5_dbug: out std_logic_vector(3 downto 0);
                  count_out_pe : out std_logic_vector (7 downto 0)
          ) ;
end component;
--Interface controller component listing

component CONTChip is
   generic (Chip_addr : integer := 3;
          Inst0 : integer := 156;
          Inst1 : integer := 48;
          Inst2 : integer := 152
                  );
   port (
      Data_bus: inout STD_LOGIC_VECTOR (15 downto 0);
      Chip_EN: in STD_LOGIC;
      Snd_i,stoplp: in std_logic;
      Rst: in STD_LOGIC;
      Clk: in STD_LOGIC;
      tbus_grnt: in STD_LOGIC;
      token_bus: inout STD_LOGIC_VECTOR (31 downto 0);
      tbus_req: out STD_LOGIC;
      I_rdy: out std_logic;
      Avail: out STD_LOGIC_VECTOR (4 downto 0);
  x_dbug : out std_logic_vector(6 downto 0);
  count_dbug : out std_logic_vector(6 downto 0);
  Wr_out_dbug  : out std_logic_vector (1 downto 0);
```

```
  R_L_Table_dbug: out STD_LOGIC_VECTOR (1 downto 0);
  Ld_Rd_dbug: out STD_LOGIC;
  ccntl_in_dbug :out std_logic_vector(24 downto 0);
  dataout_lut : out std_logic_vector(15 downto 0);
  outbuf0_dbug: out std_logic_vector(15 downto 0);
  outbuf1_dbug : out std_logic_vector(15 downto 0);
  line_out_dbug: out std_logic_vector(31 downto 0);
  l_in : out std_logic_vector(31 downto 0);
  buf_dbug : out std_logic_vector(24 downto 0);
  cntl_out_fin : out std_logic_vector( 3 downto 0);
  dlout_contchip:out std_logic_vector(15 downto 0);
  dwr_cont: out std_logic;
  tab_in_contchip: out std_logic
   );
end component;

--  Component Listing for Process Req token mapper

component Token_mapr is
   port (
                    token_bus: inout STD_LOGIC_VECTOR (31 downto 0);
                    bus_req: inout STD_LOGIC;
                    clk : in std_logic;
                rst : in std_logic;
                    bus_grnt: in STD_LOGIC;
                Avail3: in STD_LOGIC_VECTOR (4 downto 0);
                    Avail4: in STD_LOGIC_VECTOR (4 downto 0);
                    Avail2: in STD_LOGIC_VECTOR (4 downto 0);
                    Avail5: in STD_LOGIC_VECTOR (4 downto 0);
                    obstemp6_prtdbug,t6_prtdbug: out std_logic_vector(22 downto 0)
            );
end component;
-- Divider PE
component Divpe is
 port (Cntrlr_bus : in std_logic_vector(15 downto 0);
      Snd_I : out std_logic;
      clk : in std_logic;
      rst : in std_logic;
      Instr_rdy : in std_logic;
      Fin : out std_logic;
      Data_bus : inout std_logic_vector(15 downto 0);
      Bus_req : out std_logic;
      Bus_gnt : in std_logic;
      Addr : out std_logic_vector(6 downto 0);
      R_W : buffer std_logic;
      loc_bus_dbug : out std_logic_vector(7 downto 0);
      Iaddr_bus_dbug : out std_logic_vector(7 downto 0);
      Iaddr_dbug :  out std_logic_vector(7 downto 0);
      R2_out_dbug : out std_logic_vector( 7 downto 0);
      Imem_bus_dbug : out std_logic_vector(15 downto 0 )
     );
end component;

component multpe is
   Port ( mcntl_bus : in std_logic_vector(15 downto 0);
               Snd_I : out std_logic;
```

```
                clk : in std_logic;
                rst : in std_logic;
                Instr_rdy : in std_logic;
                Fin : out std_logic;
                mdata_bus : inout std_logic_vector(15 downto 0);
                bus_req : out std_logic;
                bus_gnt : in std_logic;
                multaddr : out std_logic_vector(7 downto 0);--Output address to shared dmem
                r_w : inout std_logic;
                cbusout_dbug : out std_logic_vector(7 downto 0);
                Iaddr_bus_dbug : out std_logic_vector(7 downto 0);
                R2out_dbug : out std_logic_vector( 7 downto 0);
                Imem_bus_dbug : out std_logic_vector(15 downto 0 );
                mux3out_dbg:out std_logic_vector(7 downto 0);
                ms3dbg:out std_logic_vector(1 downto 0);
                ms1dbg : out std_logic;
                ms2dbg : out std_logic;
component multpe is
   Port (    mcntl_bus : in std_logic_vector(15 downto 0);
                Snd_I : out std_logic;
                clk : in std_logic;
                rst : in std_logic;
                Instr_rdy : in std_logic;
                Fin : out std_logic;
                mdata_bus : inout std_logic_vector(15 downto 0);
                bus_req : out std_logic;
                bus_gnt : in std_logic;
                multaddr : out std_logic_vector(7 downto 0);--Output address to shared dmem
                r_w : inout std_logic;
                cbusout_dbug : out std_logic_vector(7 downto 0);
                Iaddr_bus_dbug : out std_logic_vector(7 downto 0);
                R2out_dbug : out std_logic_vector( 7 downto 0);
                Imem_bus_dbug : out std_logic_vector(15 downto 0 );
                mux3out_dbg:out std_logic_vector(7 downto 0);
                ms3dbg:out std_logic_vector(1 downto 0);
                ms1dbg : out std_logic;
                ms2dbg : out std_logic;
                adderout_dbug : out std_logic_vector(7 downto 0);
                ms4dbg : out std_logic;
                lmd_dbg,lmr_dbg : out std_logic;
                ndout : out std_logic;
                multout_fin : out std_logic_vector( 15 downto 0);
                tomultr_dbg:out std_logic_vector(7 downto 0);
                tomultd_dbg:out std_logic_vector(7 downto 0)

                        );
end component;

component gate_ic_a is
   Port ( clk: in std_logic ;
          rst: in std_logic ;
          ctrl: in std_logic_vector(3 downto 0) ;
          qdep: in std_logic_vector(19 downto 0) ;
          addr_bus: in std_logic_vector(27 downto 0) ;
          data_in0,data_in1,data_in2,data_in3 : in std_logic_vector(15 downto 0) ;
          rw: in std_logic_vector(3 downto 0) ;
```

```
        flag: inout std_logic_vector(3 downto 0) ;
        data_out0,data_out1,data_out2,data_out3: out std_logic_vector(15 downto 0)
        );
end component;


--Begin signals used in the system
signal dbus_sig0,dbus_sig1,dbus_sig2,dbus_sig3: std_logic_vector(15 downto 0);
signal rw_sig0,rw_sig1,rw_sig2,rw_sig3: std_logic;
signal db_pe_icm0,db_pe_icm1,db_pe_icm2,db_pe_icm3: std_logic_vector(15 downto 0);
signal db_grant0,db_grant1,db_grant2,db_grant3:std_logic;
signal i_rdy_icm0,i_rdy_icm1,i_rdy_icm2,i_rdy_icm3: std_logic;
signal db_req0,db_req1,db_req2,db_req3: std_logic;
signal snd_i_icm0,snd_i_icm1,snd_i_icm2,snd_i_icm3: std_logic;
signal ce_sig0,ce_sig1,ce_sig2,ce_sig3:std_logic;
signal addr_0,addr_1,addr_2,addr_3:std_logic_vector(7 downto 0);
signal stop_lp_sig0,stop_lp_sig1: std_logic;
signal tbgrnt_sig0,tbgrnt_sig1,tbgrnt_sig2,tbgrnt_sig3:std_logic ;
signal tbreq_sig0,tbreq_sig1,tbreq_sig2,tbreq_sig3 : std_logic;
signal avlsig0,avlsig1,avlsig2,avlsig3 : std_logic_vector( 4 downto 0);
signal op_token_bus_sig : std_logic_vector(31 downto 0);
signal bus_req_prt,bus_grnt_prt : std_logic;
signal mem_ad : std_logic_vector (7 downto 0);
signal mem_di_0,mem_di_1,mem_di_2,mem_di_3 : std_logic_vector( 15 downto 0);
signal mem_do_0,mem_do_1,mem_do_2,mem_do_3 : std_logic_vector( 15 downto 0);
signal m_r_w : std_logic;
signal optmp_req : std_logic;
signal op_gnt:std_logic; -- This was earlier set to buffer resulting in elaboration error in post-translate
simulation
signal odr0,odr1: std_logic;
signal Rq_OPT0 :  std_logic;
signal Rq_OPT1 :  std_logic;
signal rq_ipt0,rq_ipt1 : std_logic;


begin
--Port Mapping for components
PE3_CE0: pe port map( Data_Bus=>dbus_sig0,
                      R_W => rw_sig0,
                      Cntl_bus=>db_pe_icm0,
                      RST=>rst,
                  ODR=>odr0,
                  IDV=>idv0,
                      clk=>clk,
                  Bus_grant=>db_grant0,
                  CInstr_rdy=>I_rdy_icm0,
                      inpt =>inpt_data0,
                      Bus_req=>db_req0,
                      Snd_Instr=>snd_i_icm0,
                      Fin=>ce_sig0,
              Addr =>addr_0,
              Rq_inpt=>Rq_IPT0,
              Rq_outpt=>Rq_OPT0,
              STOPLOOP =>Stop_lp_sig0,
              -- added for dbugging
              R3_out_dbug=>R3_out_dbug_fin0,
```

```
                shft_out_dbug=>shft_out_dbug_fin0,
                dbug_st_pe => dbug_st_pe_fin0,
                R0_out_dbug => R0_out_dbug_fin0,
                        tmp3_dbug => temp3_ce0,
                        tmp2_dbug => temp2_ce0,
                        tmp1_dbug => temp1_ce0,
                        tmp44_dbug => temp4_ce0,
                        tmp5_dbug => temp5_ce0,
                        count_out_pe => open

                      );
PE2_CE1: pe port map( Data_Bus=>dbus_sig1,
                R_W => rw_sig1,
                Cntl_bus=>db_pe_icm1,
                RST=>rst,
                ODR=>odr1,
                IDV=> idv1,
                clk=>clk,
                        Bus_grant=>db_grant1,
                        CInstr_rdy=>I_rdy_icm1,
                        inpt =>inpt_data1,
                        Bus_req=>db_req1,
                        Snd_Instr=>snd_i_icm1,
                        Fin=>ce_sig1,
                        Addr =>addr_1,
                        Rq_inpt=>Rq_IPT1,
                        Rq_outpt=>Rq_OPT1,
                        STOPLOOP =>Stop_lp_sig1,
        -- added for dbugging
                R3_out_dbug=>R3_out_dbug_fin1,
                shft_out_dbug=>shft_out_dbug_fin1,
                dbug_st_pe => dbug_st_pe_fin1,
                R0_out_dbug => R0_out_dbug_fin1,
                     tmp3_dbug => temp3_ce1,
                        tmp2_dbug => temp2_ce1,
                        tmp1_dbug => temp1_ce1,
                        tmp44_dbug => temp4_ce1,
                        tmp5_dbug => temp5_ce1,
                        count_out_pe => count_ce1
      );
Icmodule0: contchip port map( Data_bus => db_pe_icm0,
                                Chip_EN => ce_sig0,
                                Snd_i => snd_i_icm0,
                                stoplp => stop_lp_sig0,
                                Rst => rst,
                        Clk =>clk,
                             tbus_grnt =>tbgrnt_sig0,
                                  token_bus =>op_token_bus_sig,
                        tbus_req =>tbreq_sig0,
                             I_rdy =>I_rdy_icm0,
                                  Avail =>avlsig0,
                        x_dbug  =>x_dbug_fin0,
                             count_dbug =>count_dbug0,
                        Wr_out_dbug  =>Wr_out_dbug0_fin0,
                        R_L_Table_dbug =>RLTable0,
                        Ld_Rd_dbug =>open,
```

```
                        dataout_lut =>dataout_lut_fin0,
                        outbuf0_dbug =>open,
                        outbuf1_dbug =>open,
                                line_out_dbug =>open,
                                l_in =>l_in_fin0,
                                buf_dbug => open       ,
                            ccntl_in_dbug => open,
                                cntl_out_fin => control_0,
                                dlout_contchip=>dloutfin0,
                                dwr_cont=>dwr0,
                                tab_in_contchip => tabin0

                            );
Icmodule1: contchip   Generic map (chip_addr =>2,
                                    Inst0=> 156,
                                    Inst1=> 48,
                                    Inst2=> 152)
    port map(       Data_bus => db_pe_icm1,
            Chip_EN => ce_sig1,
            Snd_i => snd_i_icm1,
                stoplp => stop_lp_sig1,
            Rst => rst,
            Clk =>clk,
            tbus_grnt =>tbgrnt_sig1,
            token_bus =>op_token_bus_sig,
            tbus_req =>tbreq_sig1,
            I_rdy =>I_rdy_icm1,
            Avail =>avlsig1,
            x_dbug  =>x_dbug_fin1,
            count_dbug =>count_dbug1,
            Wr_out_dbug  =>Wr_out_dbug1_fin1,
            R_L_Table_dbug =>RLTable1,
            Ld_Rd_dbug =>open,
            dataout_lut =>dataout_lut_fin1,
            outbuf0_dbug =>open,
            outbuf1_dbug =>open,
                    line_out_dbug =>open,
                    l_in =>l_in_fin1 ,
                    buf_dbug => open,
                    ccntl_in_dbug => open,
                    cntl_out_fin => control_1,
                    dlout_contchip=>dloutfin1,
                    dwr_cont=>dwr1,
                    tab_in_contchip => tabin1
              );

-- port mapping for interface controller module for div chip
Icmodule2: contchip   Generic map (chip_addr => 4,
                                    Inst0=> 142,
                                    Inst1=> 255,
                                    Inst2=> 142)
    port map(   Data_bus => db_pe_icm2,
            Chip_EN => ce_sig2,
            Snd_i => snd_i_icm2,
            stoplp => '0',
            Rst => rst,
```

```
            Clk =>clk,
            tbus_grnt =>tbgrnt_sig2,
            token_bus =>op_token_bus_sig,
            tbus_req =>tbreq_sig2,
            I_rdy =>I_rdy_icm2,
            Avail =>avlsig2,
            x_dbug  =>open,
            count_dbug =>open,
            Wr_out_dbug  =>open,
            R_L_Table_dbug =>RLTable2,
            Ld_Rd_dbug =>open,
            dataout_lut =>dataout_lut_fin2,
            outbuf0_dbug =>open,
            outbuf1_dbug =>open,
                line_out_dbug =>open,
                l_in =>l_in_fin2 ,
                buf_dbug => open,
                ccntl_in_dbug => open,
                    dwr_cont=>dwr2,
                    tab_in_contchip => tabin2
             );
Icmodule3: contchip   Generic map (chip_addr => 5,
                                   Inst0=> 142,
                             Inst1=> 255,
                                   Inst2=> 142)
    port map(   Data_bus => db_pe_icm3,
            Chip_EN => ce_sig3,
            Snd_i => snd_i_icm3,
            stoplp => '0',
            Rst => rst,
            Clk =>clk,
            tbus_grnt =>tbgrnt_sig3,
            token_bus =>op_token_bus_sig,
            tbus_req =>tbreq_sig3,
            I_rdy =>I_rdy_icm3,
            Avail =>avlsig3,
            x_dbug  =>x_dbug_fin3,
            count_dbug =>count_dbug3,
            Wr_out_dbug  =>open,
            R_L_Table_dbug =>RLTable3,
            Ld_Rd_dbug =>open,
            dataout_lut =>dataout_lut_fin3,
            outbuf0_dbug =>open,
            outbuf1_dbug =>open,
                    line_out_dbug =>open,
                    l_in =>open,
                    buf_dbug => open,
                    ccntl_in_dbug => open,
                    dwr_cont=>dwr3,
                    tab_in_contchip => tabin3
   );

prtmapper: token_mapr port map( token_bus =>Op_token_bus_sig,
            bus_req=>bus_req_prt,
            clk =>clk,
            rst =>rst,
```

```
            bus_grnt =>bus_grnt_prt,
            Avail3 =>avlsig0,
            Avail4 => avlsig2,
            Avail2 =>avlsig1,
                    Avail5 => avlsig3,
                temp6_prtdbug=>open,
                    t6_prtdbug=>open

        );
DIV1 : divpe port map(Cntrlr_bus=>db_pe_icm2,
    Snd_I=> snd_i_icm2,
    clk => clk,
    rst => rst,
    Instr_rdy => I_rdy_icm2,
    Fin => ce_sig2,
    Data_bus => dbus_sig2,
    Bus_req => db_req2,
    Bus_gnt => db_grant2,
    Addr => addr_2(6 downto 0),
    R_W => rw_sig2,
    loc_bus_dbug => open,
    Iaddr_bus_dbug => open,
    Iaddr_dbug => open,
    R2_out_dbug => open,
    Imem_bus_dbug => open
    );


multpemap: multpe port map


    (   mcntl_bus => db_pe_icm3,
            Snd_I => snd_i_icm3,
            clk =>clk,
            rst =>rst,
            Instr_rdy =>i_rdy_icm3,
            Fin =>ce_sig3,
            mdata_bus =>dbus_sig3,
            bus_req =>db_req3,
            bus_gnt =>db_grant3,
            multaddr =>addr_3,
                r_w =>rw_sig3,
                cbusout_dbug => open,
                Iaddr_bus_dbug => open,
                R2out_dbug => open,
                Imem_bus_dbug =>open,
                mux3out_dbg=> open,
                ms3dbg=> open,
                ms1dbg => open,
                ms2dbg => open ,
                adderout_dbug => open,
                ms4dbg => open,
                lmd_dbg=> open,
                lmr_dbg => open,
                ndout => open,
                multout_fin => open,
```

```
                    tomultr_dbg=> open,
                    tomultd_dbg=> open

                    );

IC_gate: gate_ic_a Port map  ( clk => clk,
                                rst => rst,
                            ctrl(0) => db_req0,
                                ctrl(1) => db_req1,
                                ctrl(2) => db_req2,
                                ctrl(3) => db_req3,
                                qdep(4 downto 0)  => avlsig0,
                                qdep(9 downto 5)  => avlsig1,
                                qdep(14 downto 10)=> avlsig2,
                                qdep(19 downto 15)=> avlsig3,
                                addr_bus(6 downto 0)   => addr_0(6 downto 0),
                                addr_bus(13 downto 7)  => addr_1(6 downto 0),
                          addr_bus(20 downto 14) => addr_2(6 downto 0),
                                addr_bus(27 downto 21) => addr_3(6 downto 0),
                                data_in0  => mem_di_0,
                                data_in1  => mem_di_1,
                                data_in2  => mem_di_2,
                                data_in3  => mem_di_3,
                                rw(0) => rw_sig0,
                                rw(1) => rw_sig1,
                                rw(2) => rw_sig2,
                                rw(3) => rw_sig3,
                                flag(0) => db_grant0,
                                flag(1) => db_grant1,
                                flag(2) => db_grant2,
                                flag(3) => db_grant3,
                                data_out0 => mem_do_0,
                                data_out1 => mem_do_1,
                                data_out2 => mem_do_2,
                                data_out3 => mem_do_3
                             );
-- signals taken out for dbugging
dbus_sig0_fin0 <= dbus_sig0;
dbus_sig1_fin1 <= dbus_sig1;
dbus_sig2_fin2 <= dbus_sig2;
db_pe_icm0_fin0 <= db_pe_icm0;
db_pe_icm1_fin1 <= db_pe_icm1;
db_pe_icm1_fin2 <= db_pe_icm2;
db_pe_icm1_fin3 <= db_pe_icm3;
token_bus_prt_pe <= Op_token_bus_sig;
ce_sig1_fin1 <= ce_sig1;
ce_sig0_fin0 <= ce_sig0;
tbgrnt_sig0_fin0 <= tbgrnt_sig0;
tbgrnt_sig1_fin1 <=          tbgrnt_sig1;
tbreq_sig0_fin0 <= tbreq_sig0;
tbreq_sig1_fin1 <= tbreq_sig1;
i_rdy_icm0_fin0<= i_rdy_icm0;
i_rdy_icm1_fin1<= i_rdy_icm1;
snd_i_icm0_fin0 <= snd_i_icm0;
snd_i_icm1_fin1 <= snd_i_icm1;
db_req3_dbug<= db_req3;
```

```
db_grant3_dbug <= db_grant3;
db_req1_dbug<= db_req1;
db_grant1_dbug <= db_grant1;
db_req0_dbug<= db_req0;
db_grant0_dbug <= db_grant0;


-- changes made with the addition of IC switch
-- Address ports taken out --
     mem_ad_out_0<=addr_0(6 downto 0);
                    mem_ad_out_1<=addr_1(6 downto 0);
                    mem_ad_out_2<=addr_2(6 downto 0);
                    mem_ad_out_3<=addr_3(6 downto 0);
-- Memory contents to be viewed --
                    Mem_out_0 <= mem_do_0;
                    Mem_out_1 <= mem_do_1;
                    Mem_out_2 <= mem_do_2;
                    Mem_out_3 <= mem_do_3;
-- addition of  process 1 for the inputting of values into the data memory
input_2_mem : process(db_grant0,db_grant1,db_grant2,db_grant3,clk,rst)

begin
          if(rst ='1') then
            mem_di_0 <= x"0000";
            mem_di_1 <= x"0000";
            mem_di_2 <= x"0000";
            mem_di_3 <= x"0000";

          else

          if(clk'event and clk='0') then
             if(db_grant0 ='1' ) then

                              mem_di_0 <= dbus_sig0;
                    else mem_di_0 <=(others =>'0');
                    end if;

                    if(db_grant1 ='1' ) then

                      mem_di_1 <= dbus_sig1;
                    else mem_di_1 <=(others =>'0');
                    end if;

                    if(db_grant2 ='1' ) then

                      mem_di_2 <= dbus_sig2;
                    else mem_di_2 <=(others =>'0');
                    end if;

                    if(db_grant3 ='1' ) then

                      mem_di_3 <= dbus_sig3;
                    else mem_di_3 <=(others =>'0');
                    end if;
           end if;
 end if;
```

```
end process input_2_mem;


-- process 2 for outputting the values from data memory
output_from_mem : process(db_grant0,db_grant1,db_grant2,db_grant3,rw_sig0,rw_sig1,rw_sig2,
                    rw_sig3,clk,rst)

begin

if(rst='1') then
  dbus_sig0 <= x"0000";
  dbus_sig1 <= x"0000";
  dbus_sig2 <= x"0000";
  dbus_sig3 <= x"0000";
  else

            if(clk'event and clk='0') then
               if(db_grant0 ='1' and rw_sig0 ='0') then

                                  dbus_sig0 <= mem_do_0;
                        else dbus_sig0 <=(others =>'Z');
                        end if;

                        if(db_grant1 ='1' and rw_sig1 ='0') then

                          dbus_sig1 <= mem_do_1;
                        else dbus_sig1 <=(others =>'Z');
                        end if;

                        if(db_grant2 ='1' and rw_sig2 ='0') then

                          dbus_sig2 <= mem_do_2;
                        else dbus_sig2 <=(others =>'Z');
                        end if;

                        if(db_grant3 ='1' and rw_sig3 ='0') then

                          dbus_sig3 <= mem_do_3;
                        else dbus_sig3 <=(others =>'Z');
                        end if;
             end if;
  end if;
end process output_from_mem;



-- end of process 2

-- Token bus logic
optmp_req <= Op_req;
Tknbuslg : process (tbreq_sig0,tbgrnt_sig0,bus_req_prt,bus_grnt_prt,tbreq_sig1,
      tbgrnt_sig1,tbreq_sig2,tbgrnt_sig2,tbgrnt_sig3,tbreq_sig3,Optmp_req,Op_gnt, rst)
 begin
  if rst = '1' then
   tbgrnt_sig0 <= '0';
   bus_grnt_prt <= '0';
```

```
 tbgrnt_sig1 <= '0';
 tbgrnt_sig2 <= '0';
 tbgrnt_sig3 <= '0';
 Op_gnt <= '0';
elsif (bus_req_prt ='1')and (tbgrnt_sig0='0')  and(tbgrnt_sig1='0') and
     (tbgrnt_sig2='0')and(Op_gnt='0') and (tbgrnt_sig3='0') then
 tbgrnt_sig0 <= '0';
 bus_grnt_prt <= '1';
 tbgrnt_sig2 <= '0';
 tbgrnt_sig1 <= '0';
 tbgrnt_sig3 <= '0';
 Op_gnt <= '0';
elsif (Optmp_req ='1') and (bus_grnt_prt ='0') and (tbgrnt_sig0='0') and
   (tbgrnt_sig1='0') and (tbgrnt_sig2='0') and (tbgrnt_sig3 ='0')then
 tbgrnt_sig0 <= '0';
 bus_grnt_prt <= '0';
 tbgrnt_sig1 <= '0';
 tbgrnt_sig2 <= '0';
 tbgrnt_sig3 <= '0';
 Op_gnt <= '1';
elsif (tbreq_sig0 = '1') and (bus_grnt_prt='0') and (Op_gnt='0') and
 (tbgrnt_sig2='0')and (tbgrnt_sig1='0') and (tbgrnt_sig3 ='0') then
 tbgrnt_sig0 <= '1';
 bus_grnt_prt <= '0';
 tbgrnt_sig2 <= '0';
 tbgrnt_sig1 <= '0';
 tbgrnt_sig3 <= '0';
 Op_gnt <= '0';
elsif (tbreq_sig2='1') and (bus_grnt_prt='0') and (Op_gnt='0') and
(tbgrnt_sig0='0') and (tbgrnt_sig1='0') and (tbgrnt_sig3 ='0')  then
 tbgrnt_sig0 <= '0';
 bus_grnt_prt <= '0';
 tbgrnt_sig2 <='1';
 tbgrnt_sig1 <= '0';
 tbgrnt_sig3 <= '0';
 Op_gnt <= '0';
 elsif (tbreq_sig1='1') and (bus_grnt_prt='0') and (Op_gnt='0') and
  (tbgrnt_sig0='0') and (tbgrnt_sig2='0')and (tbgrnt_sig3='0') then
  tbgrnt_sig0 <= '0';
 bus_grnt_prt <= '0';
tbgrnt_sig2<='0';
 tbgrnt_sig1 <= '1';
 tbgrnt_sig3 <= '0';
 Op_gnt <= '0';
 elsif (tbreq_sig3='1') and (bus_grnt_prt='0') and (Op_gnt='0') and
  (tbgrnt_sig0='0') and (tbgrnt_sig2='0')and (tbgrnt_sig1='0') then
          tbgrnt_sig0 <= '0';
   bus_grnt_prt <= '0';
   tbgrnt_sig2<='0';
   tbgrnt_sig1 <= '0';
   tbgrnt_sig3 <= '1';
 Op_gnt <= '0';
end if;
if (bus_req_prt = '0') then bus_grnt_prt <= '0';
end if;
if (Optmp_req = '0') then Op_gnt <= '0';
```

```
 end if;
 if (tbreq_sig0 = '0') then tbgrnt_sig0 <= '0';
 end if;
 if (tbreq_sig2 = '0') then tbgrnt_sig2 <= '0';
 end if;
 if (tbreq_sig1 = '0') then tbgrnt_sig1 <= '0';
 end if;
 if (tbreq_sig3 = '0') then tbgrnt_sig3 <= '0';
 end if;
end process;

arbiter_logic: process(clk,rst)
begin
if rst = '1' then
        odr0<='0';
        odr1<='0';

elsif (clk'event and clk='1') then
        case rq_opt0 is
                when '1' => odr0 <= '1';
                when '0' => odr0 <= '0';
                when others =>
        end case;

        case rq_opt1 is
                when '1' => odr1 <= '1';
                when '0' => odr1 <= '0';
                when others =>
        end case;

end if;
end process arbiter_logic;

Op_token_bus_sig <= Op_token_bus when Op_gnt = '1' else
    (others=>'Z');


end Behavioral;

                adderout_dbug : out std_logic_vector(7 downto 0);
                ms4dbg : out std_logic;
                lmd_dbg,lmr_dbg : out std_logic;
                ndout : out std_logic;
                multout_fin : out std_logic_vector( 15 downto 0);
                tomultr_dbg:out std_logic_vector(7 downto 0);
                tomultd_dbg:out std_logic_vector(7 downto 0)

                );
end component;

component gate_ic_a is
   Port ( clk: in std_logic ;
          rst: in std_logic ;
          ctrl: in std_logic_vector(3 downto 0) ;
          qdep: in std_logic_vector(19 downto 0) ;
          addr_bus: in std_logic_vector(27 downto 0) ;
```

```
        data_in0,data_in1,data_in2,data_in3 : in std_logic_vector(15 downto 0) ;
        rw: in std_logic_vector(3 downto 0) ;
        flag: inout std_logic_vector(3 downto 0) ;
        data_out0,data_out1,data_out2,data_out3: out std_logic_vector(15 downto 0)
          --          f_s_out0,f_s_out1,f_s_out2,f_s_out3 : out std_logic_vector(3 downto 0);
--                    dco_out0,dco_out1,dco_out2,dco_out3 : out std_logic_vector(3 downto 0)
        );
end component;


 --
--Begin signals used in the system
signal dbus_sig0,dbus_sig1,dbus_sig2,dbus_sig3: std_logic_vector(15 downto 0);
signal rw_sig0,rw_sig1,rw_sig2,rw_sig3: std_logic;
signal db_pe_icm0,db_pe_icm1,db_pe_icm2,db_pe_icm3: std_logic_vector(15 downto 0);
signal db_grant0,db_grant1,db_grant2,db_grant3:std_logic;
signal i_rdy_icm0,i_rdy_icm1,i_rdy_icm2,i_rdy_icm3: std_logic;
signal db_req0,db_req1,db_req2,db_req3: std_logic;
signal snd_i_icm0,snd_i_icm1,snd_i_icm2,snd_i_icm3: std_logic;
signal ce_sig0,ce_sig1,ce_sig2,ce_sig3:std_logic;
signal addr_0,addr_1,addr_2,addr_3:std_logic_vector(7 downto 0);
signal stop_lp_sig0,stop_lp_sig1: std_logic;
signal tbgrnt_sig0,tbgrnt_sig1,tbgrnt_sig2,tbgrnt_sig3:std_logic ;
signal tbreq_sig0,tbreq_sig1,tbreq_sig2,tbreq_sig3 : std_logic;
signal avlsig0,avlsig1,avlsig2,avlsig3 : std_logic_vector( 4 downto 0);
signal op_token_bus_sig : std_logic_vector(31 downto 0);
signal bus_req_prt,bus_grnt_prt : std_logic;
signal mem_ad : std_logic_vector (7 downto 0);
signal mem_di_0,mem_di_1,mem_di_2,mem_di_3 : std_logic_vector( 15 downto 0);
signal mem_do_0,mem_do_1,mem_do_2,mem_do_3 : std_logic_vector( 15 downto 0);
signal m_r_w : std_logic;
signal optmp_req : std_logic;
signal op_gnt:std_logic; -- This was earlier set to buffer resulting in elaboration error in post-translate
simulation
signal odr0,odr1: std_logic;
signal Rq_OPT0 :  std_logic;
signal Rq_OPT1 :  std_logic;
signal rq_ipt0,rq_ipt1 : std_logic;
--signal idv0, idv1 : std_logic;


--signal token_bus_prt_pe_sig :std_logic_vector(31 downto 0);
begin
--Port Mapping for components
PE3_CE0: pe port map( Data_Bus=>dbus_sig0,
              R_W => rw_sig0,
            Cntl_bus=>db_pe_icm0,
            RST=>rst,
           ODR=>odr0,
          IDV=>idv0,
            clk=>clk,
           Bus_grant=>db_grant0,
            CInstr_rdy=>I_rdy_icm0,
           inpt =>inpt_data0,
            Bus_req=>db_req0,
           Snd_Instr=>snd_i_icm0,
```

```
          Fin=>ce_sig0,
            Addr =>addr_0,
            Rq_inpt=>Rq_IPT0,
          Rq_outpt=>Rq_OPT0,
          STOPLOOP =>Stop_lp_sig0,
         -- added for dbugging
            R3_out_dbug=>R3_out_dbug_fin0,
            shft_out_dbug=>shft_out_dbug_fin0,
           dbug_st_pe => dbug_st_pe_fin0,
          R0_out_dbug => R0_out_dbug_fin0,
                                        tmp3_dbug => temp3_ce0,
                        tmp2_dbug => temp2_ce0,
                        tmp1_dbug => temp1_ce0       ,
                                        tmp44_dbug => temp4_ce0,
                                        tmp5_dbug => temp5_ce0            ,
                                        count_out_pe => open
                      --  tmp6_dbug => temp6_ce0

      );
PE2_CE1: pe port map( Data_Bus=>dbus_sig1,
            R_W => rw_sig1,
              Cntl_bus=>db_pe_icm1,
            RST=>rst,
          ODR=>odr1,
          IDV=> idv1,
            clk=>clk,
         Bus_grant=>db_grant1,
           CInstr_rdy=>I_rdy_icm1,
           inpt =>inpt_data1,
           Bus_req=>db_req1,
          Snd_Instr=>snd_i_icm1,
          Fin=>ce_sig1,
           Addr =>addr_1,
           Rq_inpt=>Rq_IPT1,
         Rq_outpt=>Rq_OPT1,
         STOPLOOP =>Stop_lp_sig1,
         -- added for dbugging
           R3_out_dbug=>R3_out_dbug_fin1,
          shft_out_dbug=>shft_out_dbug_fin1,
           dbug_st_pe => dbug_st_pe_fin1,
          R0_out_dbug => R0_out_dbug_fin1,
                                        tmp3_dbug => temp3_ce1,
                        tmp2_dbug => temp2_ce1,
                        tmp1_dbug => temp1_ce1,
                                        tmp44_dbug => temp4_ce1,
                                        tmp5_dbug => temp5_ce1            ,
                                        count_out_pe => count_ce1
                      --   tmp6_dbug => temp6_ce1
      );
Icmodule0: contchip port map(  Data_bus => db_pe_icm0,
                  Chip_EN => ce_sig0,
                 Snd_i => snd_i_icm0,
               stoplp => stop_lp_sig0,
               Rst => rst,
               Clk =>clk,
               tbus_grnt =>tbgrnt_sig0,
```

```
                token_bus =>op_token_bus_sig,
                tbus_req =>tbreq_sig0,
                I_rdy =>I_rdy_icm0,
                Avail =>avlsig0,
               x_dbug  =>x_dbug_fin0,
               count_dbug =>count_dbug0,
               Wr_out_dbug  =>Wr_out_dbug0_fin0,
               R_L_Table_dbug =>RLTable0,
               Ld_Rd_dbug =>open,
               dataout_lut =>dataout_lut_fin0,
               outbuf0_dbug =>open,
               outbuf1_dbug =>open,
               --line_out_dbug =>line_out_dbug_fin0,
                                                    line_out_dbug =>open,
                              l_in =>l_in_fin0,
                              --buf_dbug => buf_dbug_fin0     ,
                                      buf_dbug => open          ,
                              --ccntl_in_dbug => ccntl_in_fin0,
                                      ccntl_in_dbug => open,
                              cntl_out_fin => control_0,
                              dlout_contchip=>dloutfin0,
                                      dwr_cont=>dwr0,
                                      tab_in_contchip => tabin0

        );
Icmodule1: contchip   Generic map (chip_addr =>2,Inst0=> 156,
          Inst1=> 48, Inst2=> 152)
        port map(  Data_bus => db_pe_icm1,
              Chip_EN => ce_sig1,
              Snd_i => snd_i_icm1,
            stoplp => stop_lp_sig1,
              Rst => rst,
              Clk =>clk,
              tbus_grnt =>tbgrnt_sig1,
              token_bus =>op_token_bus_sig,
              tbus_req =>tbreq_sig1,
              I_rdy =>I_rdy_icm1,
              Avail =>avlsig1,
             x_dbug  =>x_dbug_fin1,
             count_dbug =>count_dbug1,
             Wr_out_dbug  =>Wr_out_dbug1_fin1,
             R_L_Table_dbug =>RLTable1,
             Ld_Rd_dbug =>open,
             dataout_lut =>dataout_lut_fin1,
             outbuf0_dbug =>open,
            outbuf1_dbug =>open,
            --line_out_dbug =>line_out_dbug_fin1,
                                                 line_out_dbug =>open,
                        l_in =>l_in_fin1 ,
                        --buf_dbug => buf_dbug_fin1,
                                buf_dbug => open,
                        --ccntl_in_dbug => ccntl_in_fin1,
                                ccntl_in_dbug => open,
                        cntl_out_fin => control_1,
                        dlout_contchip=>dloutfin1,
                                dwr_cont=>dwr1,
```

```
                                tab_in_contchip => tabin1
                            --Statedbg_fin =>St_fin0
    );

-- port mappinh for interface controller module for div chip
Icmodule2: contchip   Generic map (chip_addr => 4,Inst0=> 142,
          Inst1=> 255, Inst2=> 142)
     port map(  Data_bus => db_pe_icm2,
                Chip_EN => ce_sig2,
                Snd_i => snd_i_icm2,
              stoplp => '0',
                Rst => rst,
                Clk =>clk,
                tbus_grnt =>tbgrnt_sig2,
                token_bus =>op_token_bus_sig,
                tbus_req =>tbreq_sig2,
                I_rdy =>I_rdy_icm2,
                Avail =>avlsig2,
               x_dbug  =>open,
               count_dbug =>open,
               Wr_out_dbug  =>open,
               R_L_Table_dbug =>RLTable2,
               Ld_Rd_dbug =>open,
               dataout_lut =>dataout_lut_fin2,
               outbuf0_dbug =>open,
              outbuf1_dbug =>open,
              --line_out_dbug =>line_out_dbug_fin2,
                                        line_out_dbug =>open,
                            l_in =>l_in_fin2 ,
                            buf_dbug => open,
                            ccntl_in_dbug => open,
                                    dwr_cont=>dwr2,
                                    tab_in_contchip => tabin2
    );
--
Icmodule3: contchip   Generic map (chip_addr => 5,Inst0=> 142,
          Inst1=> 255, Inst2=> 142)
     port map(  Data_bus => db_pe_icm3,
                Chip_EN => ce_sig3,
                Snd_i => snd_i_icm3,
              stoplp => '0',
                Rst => rst,
                Clk =>clk,
                tbus_grnt =>tbgrnt_sig3,
                token_bus =>op_token_bus_sig,
                tbus_req =>tbreq_sig3,
                I_rdy =>I_rdy_icm3,
                Avail =>avlsig3,
               x_dbug  =>x_dbug_fin3,
               count_dbug =>count_dbug3,
               Wr_out_dbug  =>open,
               R_L_Table_dbug =>RLTable3,
               Ld_Rd_dbug =>open,
               dataout_lut =>dataout_lut_fin3,
               outbuf0_dbug =>open,
              outbuf1_dbug =>open,
```

```
          --line_out_dbug =>line_out_dbug_fin3,
                                        line_out_dbug =>open,
                    l_in =>open,
                    buf_dbug => open,
                    ccntl_in_dbug => open,
                          dwr_cont=>dwr3,
                          tab_in_contchip => tabin3
   );

prtmapper: token_mapr port map( token_bus =>Op_token_bus_sig,
            bus_req=>bus_req_prt,
            clk =>clk,
            rst =>rst,
            bus_grnt =>bus_grnt_prt,
            Avail3 =>avlsig0,
            Avail4 => avlsig2,
            Avail2 =>avlsig1,
                              Avail5 => avlsig3,
            --obstemp6_prtdbug=>obstemp6_prtdbug_fin,
                                        obstemp6_prtdbug=>open,
            --t6_prtdbug=>t6_prtdbug_fin
                                        t6_prtdbug=>open

         );
-- Port map to the shared core generated Data Memory.
--datamem : proc_dmem port map (addr => Mem_ad(4 downto 0),clk => clk,din => Mem_di,
  --   dout => Mem_do, we => M_R_W);
-- port map to the divider and interface controller module
DIV1 : divpe port map(Cntrlr_bus=>db_pe_icm2,
      Snd_I=> snd_i_icm2,
      clk => clk,
  rst => rst,
      Instr_rdy => I_rdy_icm2,
      Fin => ce_sig2,
      Data_bus => dbus_sig2,
      Bus_req => db_req2,
      Bus_gnt => db_grant2,
      Addr => addr_2(6 downto 0),
      R_W => rw_sig2,
  loc_bus_dbug => open,
  Iaddr_bus_dbug => open,
  Iaddr_dbug => open,
  R2_out_dbug => open,
  Imem_bus_dbug => open

        );


multpemap: multpe port map


       ( mcntl_bus => db_pe_icm3,
                Snd_I => snd_i_icm3,
                clk =>clk,
                rst =>rst,
                Instr_rdy =>i_rdy_icm3,
```

```
        Fin =>ce_sig3,
        mdata_bus =>dbus_sig3,
        bus_req =>db_req3,
        bus_gnt =>db_grant3,
        multaddr =>addr_3,
                r_w =>rw_sig3,
                cbusout_dbug => open,
                Iaddr_bus_dbug => open,
                --Iaddr_dbug :  out std_logic_vector(7 downto 0);
                R2out_dbug => open,
                Imem_bus_dbug =>open,

                mux3out_dbg=> open,
                ms3dbg=> open,
                ms1dbg => open,
                ms2dbg => open ,
                adderout_dbug => open,
                ms4dbg => open,
                lmd_dbg=> open,
                lmr_dbg => open,
                ndout => open,
                --multout_fin => mult_dbug,
                multout_fin => open,
                tomultr_dbg=> open,
                tomultd_dbg=> open

                );

IC_gate: gate_ic_a
   Port map  ( clk => clk,
            rst => rst,
           ctrl(0) => db_req0,
                                                ctrl(1) => db_req1,
                                                ctrl(2) => db_req2,
                                                ctrl(3) => db_req3,
                                                qdep(4 downto 0)  => avlsig0,
                                          qdep(9 downto 5)  => avlsig1,
                                        qdep(14 downto 10)=> avlsig2,
                                          qdep(19 downto 15)=> avlsig3,
                     addr_bus(6 downto 0)   => addr_0(6 downto 0),
                                addr_bus(13 downto 7)  => addr_1(6 downto 0),
              addr_bus(20 downto 14) => addr_2(6 downto 0),
                                addr_bus(27 downto 21) => addr_3(6 downto 0),
           data_in0  => mem_di_0,
                           data_in1  => mem_di_1,
                           data_in2  => mem_di_2,
                           data_in3  => mem_di_3,
                     rw(0) => rw_sig0,
                              rw(1) => rw_sig1,
                                        rw(2) => rw_sig2,
                                        rw(3) => rw_sig3,
                     flag(0) => db_grant0,
                                        flag(1) => db_grant1,
                                        flag(2) => db_grant2,
                                        flag(3) => db_grant3,
                     data_out0 => mem_do_0,
```

```
                        data_out1 => mem_do_1,
                        data_out2 => mem_do_2,
                        data_out3 => mem_do_3
        );
-- signals taken out for dbugging
dbus_sig0_fin0 <= dbus_sig0;
dbus_sig1_fin1 <= dbus_sig1;
dbus_sig2_fin2 <= dbus_sig2;
db_pe_icm0_fin0 <= db_pe_icm0;
db_pe_icm1_fin1 <= db_pe_icm1;
db_pe_icm1_fin2 <= db_pe_icm2;
db_pe_icm1_fin3 <= db_pe_icm3;
token_bus_prt_pe <= Op_token_bus_sig;
--Addr_0_fin0 <=Addr_0;
--Addr_1_fin1<=Addr_1;
ce_sig1_fin1 <= ce_sig1;
ce_sig0_fin0 <= ce_sig0;
tbgrnt_sig0_fin0 <= tbgrnt_sig0;
tbgrnt_sig1_fin1 <=          tbgrnt_sig1;
tbreq_sig0_fin0 <= tbreq_sig0;
tbreq_sig1_fin1 <= tbreq_sig1;
i_rdy_icm0_fin0<= i_rdy_icm0;
i_rdy_icm1_fin1<= i_rdy_icm1;
snd_i_icm0_fin0 <= snd_i_icm0;
snd_i_icm1_fin1 <= snd_i_icm1;
db_req3_dbug<= db_req3;
db_grant3_dbug <= db_grant3;
db_req1_dbug<= db_req1;
db_grant1_dbug <= db_grant1;
db_req0_dbug<= db_req0;
db_grant0_dbug <= db_grant0;

--

-- changes made with the addition of IC switch
-- Address ports taken out --
    mem_ad_out_0<=addr_0(6 downto 0);
                    mem_ad_out_1<=addr_1(6 downto 0);
                    mem_ad_out_2<=addr_2(6 downto 0);
                    mem_ad_out_3<=addr_3(6 downto 0);

-- Memory contents to be viewed --

                    Mem_out_0 <= mem_do_0;
                    Mem_out_1 <= mem_do_1;
                    Mem_out_2 <= mem_do_2;
                    Mem_out_3 <= mem_do_3;


-- addition of  process 1 for the inputting of values into the data memory
input_2_mem : process(db_grant0,db_grant1,db_grant2,db_grant3,clk,rst)

begin
          if(rst ='1') then
            mem_di_0 <= x"0000";
            mem_di_1 <= x"0000";
```

```
            mem_di_2 <= x"0000";
            mem_di_3 <= x"0000";

          else

          if(clk'event and clk='0') then
            if(db_grant0 ='1' ) then

                        mem_di_0 <= dbus_sig0;
                else mem_di_0 <=(others =>'0');
                end if;

                if(db_grant1 ='1' ) then

                  mem_di_1 <= dbus_sig1;
                else mem_di_1 <=(others =>'0');
                end if;

                if(db_grant2 ='1' ) then

                  mem_di_2 <= dbus_sig2;
                else mem_di_2 <=(others =>'0');
                end if;

                if(db_grant3 ='1' ) then

                  mem_di_3 <= dbus_sig3;
                else mem_di_3 <=(others =>'0');
                end if;
           end if;
  end if;
end process input_2_mem;

          -- end of process 1


-- end of changes made ----

-- process 2 for outputting the values from data memory
output_from_mem : process(db_grant0,db_grant1,db_grant2,db_grant3,rw_sig0,rw_sig1,rw_sig2,
                  rw_sig3,clk,rst)

begin

if(rst='1') then
  dbus_sig0 <= x"0000";
  dbus_sig1 <= x"0000";
  dbus_sig2 <= x"0000";
  dbus_sig3 <= x"0000";
  else

          if(clk'event and clk='0') then
            if(db_grant0 ='1' and rw_sig0 ='0') then

                        dbus_sig0 <= mem_do_0;
                else dbus_sig0 <=(others =>'Z');
```

```
                    end if;

                    if(db_grant1 ='1' and rw_sig1 ='0') then

                      dbus_sig1 <= mem_do_1;
                    else dbus_sig1 <=(others =>'Z');
                    end if;

                    if(db_grant2 ='1' and rw_sig2 ='0') then

                      dbus_sig2 <= mem_do_2;
                    else dbus_sig2 <=(others =>'Z');
                    end if;

                    if(db_grant3 ='1' and rw_sig3 ='0') then

                      dbus_sig3 <= mem_do_3;
                    else dbus_sig3 <=(others =>'Z');
                    end if;
            end if;
  end if;
end process output_from_mem;



-- end of process 2

-- Token bus logic
optmp_req <= Op_req;
Tknbuslg : process (tbreq_sig0,tbgrnt_sig0,bus_req_prt,bus_grnt_prt,tbreq_sig1,
     tbgrnt_sig1,tbreq_sig2,tbgrnt_sig2,tbgrnt_sig3,tbreq_sig3,Optmp_req,Op_gnt, rst)
 begin
  if rst = '1' then
   tbgrnt_sig0 <= '0';
   bus_grnt_prt <= '0';
   --Tbs4_gnt <= '0';
   tbgrnt_sig1 <= '0';
   tbgrnt_sig2 <= '0';
   tbgrnt_sig3 <= '0';
   Op_gnt <= '0';
  elsif (bus_req_prt ='1')and (tbgrnt_sig0='0')  and(tbgrnt_sig1='0') and
       (tbgrnt_sig2='0')and(Op_gnt='0') and (tbgrnt_sig3='0') then
   tbgrnt_sig0 <= '0';
   bus_grnt_prt <= '1';
   --Tbs4_gnt <= '0';
   tbgrnt_sig2 <= '0';
   tbgrnt_sig1 <= '0';
   tbgrnt_sig3 <= '0';
   Op_gnt <= '0';
  elsif (Optmp_req ='1') and (bus_grnt_prt ='0') and (tbgrnt_sig0='0') and
    (tbgrnt_sig1='0') and (tbgrnt_sig2='0') and (tbgrnt_sig3 ='0')then
   tbgrnt_sig0 <= '0';
   bus_grnt_prt <= '0';
   --Tbs4_gnt <= '0';
   tbgrnt_sig1 <= '0';
   tbgrnt_sig2 <= '0';
```

```
 tbgrnt_sig3 <= '0';
 Op_gnt <= '1';
elsif (tbreq_sig0 = '1') and (bus_grnt_prt='0') and (Op_gnt='0') and
 (tbgrnt_sig2='0')and (tbgrnt_sig1='0') and (tbgrnt_sig3 ='0') then
 tbgrnt_sig0 <= '1';
 bus_grnt_prt <= '0';
 --Tbs4_gnt <= '0';
 tbgrnt_sig2 <= '0';
 tbgrnt_sig1 <= '0';
 tbgrnt_sig3 <= '0';
 Op_gnt <= '0';
elsif (tbreq_sig2='1') and (bus_grnt_prt='0') and (Op_gnt='0') and
(tbgrnt_sig0='0') and (tbgrnt_sig1='0') and (tbgrnt_sig3 ='0')  then
 tbgrnt_sig0 <= '0';
 bus_grnt_prt <= '0';
 --Tbs4_gnt <= '1';
 tbgrnt_sig2 <='1';
 tbgrnt_sig1 <= '0';
 tbgrnt_sig3 <= '0';
 Op_gnt <= '0';
 elsif (tbreq_sig1='1') and (bus_grnt_prt='0') and (Op_gnt='0') and
  (tbgrnt_sig0='0') and (tbgrnt_sig2='0')and (tbgrnt_sig3='0') then
  tbgrnt_sig0 <= '0';
 bus_grnt_prt <= '0';
-- Tbs4_gnt <= '0';
tbgrnt_sig2<='0';
 tbgrnt_sig1 <= '1';
 tbgrnt_sig3 <= '0';
 Op_gnt <= '0';
 elsif (tbreq_sig3='1') and (bus_grnt_prt='0') and (Op_gnt='0') and
  (tbgrnt_sig0='0') and (tbgrnt_sig2='0')and (tbgrnt_sig1='0') then
          tbgrnt_sig0 <= '0';
   bus_grnt_prt <= '0';
-- Tbs4_gnt <= '0';
   tbgrnt_sig2<='0';
   tbgrnt_sig1 <= '0';
   tbgrnt_sig3 <= '1';
 Op_gnt <= '0';
end if;
if (bus_req_prt = '0') then bus_grnt_prt <= '0';
end if;
if (Optmp_req = '0') then Op_gnt <= '0';
end if;
if (tbreq_sig0 = '0') then tbgrnt_sig0 <= '0';
end if;
if (tbreq_sig2 = '0') then tbgrnt_sig2 <= '0';
end if;
if (tbreq_sig1 = '0') then tbgrnt_sig1 <= '0';
end if;
if (tbreq_sig3 = '0') then tbgrnt_sig3 <= '0';
end if;
end process;

arbiter_logic: process(clk,rst)
begin
if rst = '1' then
```

```
            odr0<='0';
            odr1<='0';

elsif (clk'event and clk='1') then
            case rq_opt0 is
                        when '1' => odr0 <= '1';
                        when '0' => odr0 <= '0';
                        when others =>
            end case;

            case rq_opt1 is
                        when '1' => odr1 <= '1';
                        when '0' => odr1 <= '0';
                        when others =>
            end case;

end if;
end process arbiter_logic;

Op_token_bus_sig <= Op_token_bus when Op_gnt = '1' else
     (others=>'Z');


end Behavioral;
```

Module Name: contchip.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity CONTChip is
   generic (Chip_addr : integer := 3;
               Inst0 : integer := 156;
               Inst1 : integer := 48;
               Inst2 : integer := 152);
   port (
      Data_bus: inout STD_LOGIC_VECTOR (15 downto 0);
      Chip_EN: in STD_LOGIC;
      Snd_i,stoplp: in std_logic;
      Rst: in STD_LOGIC;
      Clk: in STD_LOGIC;
      tbus_grnt: in STD_LOGIC;
      token_bus: inout STD_LOGIC_VECTOR (31 downto 0);
      tbus_req: out STD_LOGIC;
      I_rdy: out std_logic;
      Avail: out STD_LOGIC_VECTOR (4 downto 0);
            --x_dbug : out std_logic_vector(9 downto 0);
            x_dbug : out std_logic_vector(6 downto 0);
            --count_dbug : out std_logic_vector(9 downto 0);
            count_dbug : out std_logic_vector(6 downto 0);
            Wr_out_dbug  : out std_logic_vector (1 downto 0);
            R_L_Table_dbug: out STD_LOGIC_VECTOR (1 downto 0);
            Ld_Rd_dbug: out STD_LOGIC;
            --tab_1ntry : out std_logic_vector (4 downto 0);
            --tab_addntry : out std_logic_vector ( 7 downto 0);
            --tab_exitpn_ntry : out std_logic_vector( 3 downto 0);
            ccntl_in_dbug :out std_logic_vector(24 downto 0);
```

```
            --QData_dbug : out std_logic_vector (17 downto 0);
            dataout_lut : out std_logic_vector(15 downto 0);
            outbuf0_dbug: out std_logic_vector(15 downto 0);
            outbuf1_dbug : out std_logic_vector(15 downto 0);
            line_out_dbug: out std_logic_vector(31 downto 0);
            l_in : out std_logic_vector(31 downto 0);
            buf_dbug : out std_logic_vector(24 downto 0);
           -- Statedbg_fin :out string(1 to 10):="          "
           cntl_out_fin : out std_logic_vector( 3 downto 0);
           dlout_contchip:out std_logic_vector(15 downto 0);
           dwr_cont: out std_logic;
           tab_in_contchip: out std_logic
            );
end CONTChip;

architecture CONTChip_arch of CONTChip is

component queue is                                  --FIFO  Queue code
          port ( clk, enw, rst_f,rst_r,enr,s:in std_logic;
               time_s: in std_logic_vector(3 downto 0);
               din: in std_logic_vector(17 downto 0);
               ram_add: in std_logic_vector(5 downto 0);
               prog_flag: in std_logic_vector(5 downto 0);
               error: inout std_logic;
               sign: out std_logic;
               ITRC: out std_logic_vector(3 downto 0);
               th_flag: out std_logic;
               count_token:inout std_logic_vector(5 downto 0);
               dout: out std_logic_vector(17 downto 0));
end component;

component LUT is
   generic ( Instr0 : integer := 156;
               Instr1 : integer := 48;
               Instr2 : integer := 152);
   port (
      R_L_Table: in STD_LOGIC_VECTOR (1 downto 0);
      Ld_Rd: in STD_LOGIC;
      Data: inout STD_LOGIC_VECTOR (15 downto 0);
      rst: in STD_LOGIC;
      clk : in STD_LOGIC;
      Wr_out  : in std_logic_vector (1 downto 0);
      W_en : out std_logic;
      addr: in STD_LOGIC_VECTOR (4 downto 0);
      time_stmp : in STD_LOGIC_VECTOR(2 downto 0);
      Proc_Num: in STD_LOGIC_VECTOR (4 downto 0);
      data_loc: in STD_LOGIC_VECTOR (7 downto 0);
      join_flg: buffer std_logic;
      Instr_out: out STD_LOGIC_VECTOR (15 downto 0);
            --tab_1ntry : out std_logic_vector (4 downto 0);
            --tab_addntry : out std_logic_vector ( 7 downto 0);
            --tab_exitpn_ntry : out std_logic_vector( 3 downto 0)
                     tab_in_dbg: out std_logic
   );
end component;
```

```
component Cntl_Logic is
   generic (Chip_addr : integer := 3;
         Inst0 : integer := 156;
               Inst1 : integer := 48;
               Inst2 : integer := 152);
   port (
      rst: in STD_LOGIC;
      clk: in STD_LOGIC;
      tkn_bus: inout STD_LOGIC_VECTOR (31 downto 0);
      Cnt_token: in STD_LOGIC_VECTOR (5 downto 0);
      thl_flag: in STD_LOGIC;
      ITRC: in STD_LOGIC_VECTOR (3 downto 0);
      sign: in STD_LOGIC;
      Join_flg: in STD_LOGIC;
      data: inout STD_LOGIC_VECTOR (15 downto 0);
      En_W: out STD_LOGIC;
      En_R: out STD_LOGIC;
      rst_f: out STD_LOGIC;
      rst_r: out STD_LOGIC;
      s: out STD_LOGIC;
      bus_grant : in std_logic;
      bus_rqst  : out std_logic;
      time_s: out STD_LOGIC_VECTOR (3 downto 0);
      ram_addr: out STD_LOGIC_VECTOR (5 downto 0);
      D_out: out STD_LOGIC_VECTOR (17 downto 0);
      Prog_flag: out STD_LOGIC_VECTOR (5 downto 0);
      wr_out: buffer STD_LOGIC_VECTOR (1 downto 0);
      LT_addr: out STD_LOGIC_VECTOR (4 downto 0);
      rst_LT: out STD_LOGIC;
      R_L_table: buffer STD_LOGIC_VECTOR (1 downto 0);
      Ld_Rd: out STD_LOGIC;
      Instr_Rdy: out STD_LOGIC;
      Snd_instr : in std_logic;
      finished, stoploop: in STD_LOGIC;
            -- x_dbug : out std_logic_vector(9 downto 0);
            x_dbug : out std_logic_vector(6 downto 0);
             --count_dbug : out std_logic_vector( 9 downto 0);
             count_dbug : out std_logic_vector( 6 downto 0);
             outbuf0_dbug : out std_logic_vector(15 downto 0);
             outbuf1_dbug : out std_logic_vector(15 downto 0);
             line_out_dbug: out std_logic_vector(31 downto 0);
             line_in_dbug : out std_logic_vector(31 downto 0);
             buf_in_dbug : out std_logic_vector(24 downto 0);
             cntl_in_dbug : out std_logic_vector (24 downto 0);
             cntl_out : out std_logic_vector( 3 downto 0);
              dlout:out std_logic_vector(15 downto 0);
                     dwr_op: out std_logic
            --Statedbg:out string(1 to 10):="          "

   );
end component;

signal Instr_out : std_logic_vector(15 downto 0);       --LUT output
signal WEN : std_logic;                                 --chip output enable
signal QData : std_logic_vector(17 downto 0);           --FIFO output
signal rst_lut, rst_f, rst_r : std_logic;
```

```
signal R_L_Table, WR_Out : std_logic_vector(1 downto 0);
signal Read_Load : std_logic;
signal jn_flag : std_logic;
signal LData : std_logic_vector(15 downto 0);                   --I/O for LUT
signal Addr  : std_logic_vector(4 downto 0);            --LUT address lines
signal tok_cnt : std_logic_vector(5 downto 0);                  --FIFO count
signal Thres_flag : std_logic;                                  --Threshold flag
signal ITRC : std_logic_vector(3 downto 0);
signal sign, s : std_logic;
signal en_Wr, en_Rd : std_logic;                        --FIFO read/write
signal time_S : std_logic_vector(3 downto 0);                   --FIFO time setting
signal Ram_addr : std_logic_vector(5 downto 0);                 --FIFO address lines
signal FData : std_logic_vector(17 downto 0);                   --FIFO input lines
signal Prog_flag : std_logic_vector(5 downto 0);        --FIFO threshold set lines
-- added for dbugging

begin

  Cont1 : Cntl_logic generic map (Chip_addr,Inst0, Inst1, Inst2)
                          port
map(rst=>Rst,clk=>Clk,tkn_bus=>token_bus,Cnt_token=>tok_cnt,thl_flag=>Thres_flag,

ITRC=>ITRC,sign=>sign,join_flg=>jn_flag,data=>LData,En_W=>en_Wr,En_R=>en_Rd,rst_f=>rst_f,rst_
r=>rst_r,
                s=>s,bus_grant=>tbus_grnt,bus_rqst=>tbus_req,time_s=>time_S,ram_addr=>Ram_addr,
                D_out=>FData,Prog_flag=>Prog_flag,wr_out=>WR_Out,LT_addr=>Addr,rst_LT=>rst_lut,
                R_L_table=>R_L_Table,Ld_Rd=>Read_Load,Instr_Rdy=>I_rdy,Snd_instr=>Snd_i,
                                        finished=>Chip_EN,
stoploop=>stoplp,x_dbug=>x_dbug,count_dbug=>count_dbug,
                                        outbuf0_dbug=>outbuf0_dbug,outbuf1_dbug=>outbuf1_dbug,
                                        line_out_dbug=>line_out_dbug,line_in_dbug =>
l_in,buf_in_dbug=>buf_dbug,

cntl_in_dbug=>ccntl_in_dbug,cntl_out=>cntl_out_fin,dlout=>dlout_contchip,dwr_op=> dwr_cont);

  LUT1  : LUT generic map(Inst0, Inst1, Inst2)
          port map(R_L_Table=>R_L_Table,Ld_Rd=>Read_Load,Data=>LData,rst=>rst_lut,clk=>clk,
          Wr_out=>WR_Out,W_en=>WEN,addr=>Addr,time_stmp=>QData(17 downto
15),Proc_Num=>QData(14 downto 10),
          data_loc=>QData(7 downto 0),join_flg=>jn_flag,Instr_out=>Instr_out,tab_in_dbg =>
tab_in_contchip
                          );

  FIFOQ : queue port
map(clk=>clk,enw=>en_Wr,rst_f=>rst_f,rst_r=>rst_r,enr=>en_Rd,s=>s,time_s=>time_S,

din=>FData,ram_add=>Ram_addr,prog_flag=>Prog_flag,error=>open,sign=>sign,ITRC=>ITRC,
            th_flag=>Thres_flag,count_token=>tok_cnt,dout=>QData);

 -- added for checking the changes

 Wr_out_dbug <= wr_out;
 R_L_Table_dbug<= R_L_Table;
 Ld_Rd_dbug <= Read_Load ;
-- QData_dbug<=QData;
dataout_lut<= Ldata;
```

```
 Data_bus <= Instr_out when WEN = '1' else (others=>'Z');
   Avail <= Tok_cnt(4 downto 0);
 end CONTChip_arch;
```

Module Name: cntl_logic.vhd

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use std.textio.all;

entity Cntl_Logic is
    generic (Chip_addr : integer := 3;
          Inst0 : integer := 156;
                Inst1 : integer := 48;
                Inst2 : integer := 152);
    port (
       rst: in STD_LOGIC;
       clk: in STD_LOGIC;
       tkn_bus: inout STD_LOGIC_VECTOR (31 downto 0);
       Cnt_token: in STD_LOGIC_VECTOR (5 downto 0);
       thl_flag: in STD_LOGIC;
       ITRC: in STD_LOGIC_VECTOR (3 downto 0);
       sign: in STD_LOGIC;
       Join_flg: in STD_LOGIC;
       data: inout STD_LOGIC_VECTOR (15 downto 0);
       En_W: out STD_LOGIC;
       En_R: out STD_LOGIC;
       rst_f: out STD_LOGIC;
       rst_r: out STD_LOGIC;
       s: out STD_LOGIC;
       bus_grant : in std_logic;
       bus_rqst  : out std_logic;
       time_s: out STD_LOGIC_VECTOR (3 downto 0);
       ram_addr: out STD_LOGIC_VECTOR (5 downto 0);
       D_out: out STD_LOGIC_VECTOR (17 downto 0);
       Prog_flag: out STD_LOGIC_VECTOR (5 downto 0);
       wr_out: buffer STD_LOGIC_VECTOR (1 downto 0);
       LT_addr: out STD_LOGIC_VECTOR (4 downto 0);
       rst_LT: out STD_LOGIC;
       R_L_table: buffer STD_LOGIC_VECTOR (1 downto 0);
       Ld_Rd: out STD_LOGIC;
       Instr_Rdy: out STD_LOGIC;
       Snd_instr : in std_logic;
       finished, stoploop: in STD_LOGIC;
             --x_dbug :  out std_logic_vector(9 downto 0);
               x_dbug :  out std_logic_vector(6 downto 0);
             --count_dbug :  out std_logic_vector(9 downto 0);
             count_dbug :  out std_logic_vector(6 downto 0);
             cntl_in_dbug : out std_logic_vector(24 downto 0);
             outbuf0_dbug : out std_logic_vector( 15 downto 0);
             outbuf1_dbug : out std_logic_vector( 15 downto 0);
             line_out_dbug: out std_logic_vector(31 downto 0);
             line_in_dbug : out std_logic_vector(31 downto 0);
```

```
            buf_in_dbug : out std_logic_vector(24 downto 0);
           --  Statedbg:out string(1 to 10):="          "
            cntl_out : out std_logic_vector( 3 downto 0);
            dlout:out std_logic_vector(15 downto 0);
                    dwr_op: out std_logic
   );
end Cntl_Logic;

architecture Cntl_Logic_arch of Cntl_Logic is

component mapbuf
        port (
        din: IN std_logic_VECTOR(24 downto 0);
        clk: IN std_logic;
        wr_en: IN std_logic;
        rd_en: IN std_logic;
        ainit: IN std_logic;
        dout: OUT std_logic_VECTOR(24 downto 0);
        full: OUT std_logic;
        empty: OUT std_logic);
end component;
--signal stout:string(1 to 10):="State     ";
signal nxt_lded : std_logic;
signal wr_en, ld_t : std_logic;
signal line_in, line_out : std_logic_vector(31 downto 0);
constant Load_Table : std_Logic_vector := "111111";  --tkn opcode
constant Load_Thres : std_logic_vector := "111101";  --tkn opcode
constant Table_input: std_logic_vector := "111110";  --tkn opcode
constant Status     : std_logic_vector := "111100";  --tkn opcode
constant Switch     : std_logic_vector := "111011";  --tkn opcode
constant tken           : std_logic_vector := "00----";  --tkn value
constant PRT_addr   : std_logic_vector := "0000001";   --PRT addr
constant PRT_stat   : std_logic_vector := "0000000111100";  --snd status to PRT
signal lcl_addr : std_logic_vector(6 downto 0);
type State_Type is (Sysrst,Ld_table,GetTkn,StopL, DeQ,Issue,Dummy,SndPRT,ChkStat,PRam);
signal State: State_Type;
-- entry is data structure for loading LUT
type entry is record
    entry0, entry1: std_logic_vector(15 downto 0);
end record;
--*************************Make changes here for different apps************************
type entry_tbl is array(6 downto 0) of entry;
--*************************************************************************************
signal tbl_entry : entry_tbl;
signal outbuf0, outbuf1 : std_logic_vector(15 downto 0);
signal buf_in, temp3 : std_logic_vector(24 downto 0);
signal dline_in, dline_out : std_logic_vector(15 downto 0);
signal dwr : std_logic;
signal re, we, empty, full : std_logic;
signal cntl_in, last_cntl_in : std_logic_vector(24 downto 0);
--signal count, x : std_logic_vector(9 downto 0);
signal count, x : std_logic_vector(6 downto 0);

begin
dlout<=dline_out;
x_dbug <= x;
```

```
count_dbug<= count;
cntl_in_dbug <= cntl_in;
lcl_addr <= conv_std_logic_vector(Chip_addr, 7);
outbuf0_dbug<=outbuf0;
outbuf1_dbug<=outbuf1;
line_out_dbug<= line_out;
line_in_dbug <= line_in;
buf_in_dbug <= buf_in;
dwr_op <= dwr;
-- define tri-state logic for token bus
with (wr_en) select
        line_in <=  tkn_bus  when '1',
                    (others=>'0') when others;

tkn_bus <= line_out when wr_en = '0' else
           (others=>'Z');
-- define tri-state logic for data bus
dline_in <= data when dwr = '1' else
           (others=> 'Z');
data <= dline_out when dwr = '0' else
           (others=> 'Z');

INFifo : mapbuf port map (din => buf_in,clk =>clk,wr_en => we,rd_en => re,
                          ainit => rst, dout => cntl_in,
                          full => full,empty => empty);

getdata : process (clk, full, line_in, rst)
        begin
                if rst = '1' then
                        we <= '0';
                        buf_in <= (others=>'0');
                elsif (clk'event and clk='1') then
                        if (line_in(30 downto 24) = lcl_addr and full ='0') then
                                buf_in <= line_in(31)&line_in(23 downto 0);
                                we <= '1';
                        else
                                buf_in <= (others=>'0');
                                we <= '0';
                        end if;
                end if;
end process;

-- Initialize the Table with entry0 and entry1 asynchronously at reset.

--init_table: process(rst)
--begin
--if rst = '1' then
-- for i in 0 to 4 loop
--        tbl_entry(i).entry0(15 downto 0)<=x"0000";
--        tbl_entry(i).entry1(15 downto 0)<=x"0000";
-- end loop;
--end if;
--end process init_table;
```

```
CntlSt: process (clk,rst)

  variable ind, ind2 : integer;
  variable done, comp, running, stopflag, Snd_done, in_delay, buf_delay : Boolean;
  variable delay, iter, fin_join, first_val, in_delay2 : Boolean;
  variable iss_delay, is2_delay : Boolean;

  begin
   if rst = '1' then
              State <= Sysrst;
          elsif (clk'event and clk='1') then

            case State is
                    when Sysrst =>
                            cntl_out <="0000";

--                          stout<="Reset     ";
                            --count <= "0000000001";  done := False; x <= "0000000001";
                    --      count <= "00001";  done := False; x <= "00001";
                            count <= "0000001";  done := False; x <= "0000001";
                            Snd_done := False; comp := False; running := False;
                            bus_rqst <= '0'; first_val := true; in_delay2 := False;
                            dwr <= '1'; iss_delay := False; in_delay := false; stopflag:=false;
                            rst_f <= '1';                   --reset Queue
                            rst_r <= '1'; buf_delay := false;
                            rst_LT <= '1';                  --reset LUT
                            R_L_Table <= "00"; is2_delay := false;
                            Ld_RD <= '0';
                            nxt_lded <='0';                 --block PE from getting tkn
                            wr_en <= '1';                     --enable bus snoop
                            State <= Ld_Table;
                            Instr_rdy <= '0';
                            fin_join := false;
                            prog_flag <= "000000";
                            LT_addr <= "00000";
                            wr_out <= "00";
                            en_W <= '0'; en_R <= '0';
                            time_s <= "0000"; s <= '0';
                            ram_addr <= "000000";
                            D_out <= "0000000000000000";
                            re <= '0';
                            delay := false; iter := false;
                            temp3 <= (others=>'0');
                            last_cntl_in <= (others=>'0');

                    when Ld_Table =>
              cntl_out <="0001";
--                          stout<="Load Table";
```

```
                wr_en <= '1';
            Ld_Rd <= '0';
            rst_f <= '0';
            rst_r <= '0';
            rst_LT <= '0';
            en_W <= '0'; en_R <= '0';
                s <= '0';
                ram_addr <= "000000";
                D_out <= "00000000000000000000";
                bus_rqst <= '0';
                wr_out <= "00";
            if (done = false) then  --get table tokens
              case count is
                when "0000001" => ind := 0;
                when "0000010" => ind := 1;
                when "0000100" => ind := 2;
                when "0001000" => ind := 3;
                when "0010000" => ind := 4;
                when "0100000" => ind := 5;
                when "1000000" => ind := 6;
                when others => null;
               end case;
              if (empty = '0' and in_delay = false) then
                Re <='1';                                   --get token from queue
                in_delay := true;
                Count <= count;
                State <= Ld_table;
              elsif (in_delay = true and in_delay2 = False) then
                in_delay2 := true;  re <= '0';
                Count <= Count;
                State <= Ld_table;
              elsif (in_delay2 = true) then             --parse token
                if (cntl_in(24 downto 19))=Load_Table then
                        tbl_entry(ind).entry1(7 downto 0) <= cntl_in(7 downto 0);       --data
addr
                    tbl_entry(ind).entry0(0) <= cntl_in(8);    --hold field
                    tbl_entry(ind).entry1(8) <= cntl_in(9);    --Join field
                    Count <= Count;
                   elsif (cntl_in(24 downto 19))=Table_Input then
            tbl_entry(ind).entry0(15 downto 11)<=cntl_in(18 downto 14); --PN
            tbl_entry(ind).entry0(10 downto 6) <=cntl_in(13 downto 9);  --Next PN
            tbl_entry(ind).entry0(5 downto 1) <=cntl_in(8 downto 4);    --Next PN1
            tbl_entry(ind).entry1(12 downto 9) <=cntl_in(3 downto 0);   --Exit PN
                                        tbl_entry(ind).entry1(15 downto 13) <="000"; --
ununsed bits init to 0
            --count <= count(8 downto 0)&count(9);
                        count <= count(5 downto 0)&count(6);
            --if count < "1000000000" then
                          if count < "1000000" then
              done := false;
            else
              done := True;
            end if;
                    end if;
                    in_delay := false;
                    in_delay2 := false;
```

```
                    Re <= '0';
                end if;
                State <= Ld_Table;
              elsif done = True then                    -- load LUT
                re <= '0';
                                                        case x is
                        when "0000001" => LT_addr <= "00000"; ind2 := 0;
                        when "0000010" => LT_addr <= "00001"; ind2 := 1;
                        when "0000100" => LT_addr <= "00010"; ind2 := 2;
                        when "0001000" => LT_addr <= "00011"; ind2 := 3;
                        when "0010000" => LT_addr <= "00100"; ind2 := 4;
                        when "0100000" => LT_addr <= "00101"; ind2 := 5;
                        when "1000000" => LT_addr <= "00110"; ind2 := 6;
                        when others => null;

                end case;
                case R_L_Table is
                   when "00" => dwr <= '0';                     --enable write to LUT
                                dline_out  <= tbl_entry(ind2).entry0;
                                        R_L_Table <="01";
                                        State <= Ld_Table;
                             when "01" => dwr <= '0';
                                        dline_out <= tbl_entry(ind2).entry1;
                                        R_L_Table <= "10";
                                        State <= Ld_Table;
                                      --          when "10" => R_L_Table <= "00";
                                   when "10" => R_L_Table <= "00";
                                        dwr <= '0';                     --enable write to LUT
                     dline_out  <= tbl_entry(ind2).entry0;


                                                --if x < "1000000000" then
                                                        if x < "1000000" then
                                              --          x <= x(8 downto 0)&x(9);
                                                           x <= x(5 downto 0)&x(6);
                                        State <= Ld_table;
                                                                else
                                        done := False;
                                        --x <= x(8 downto 0)&x(9);
                                                                 x <= x(5 downto 0)&x(6);
                                        dwr <= '1';
                                        State <= GetTkn;
                                       end if;
                   when others => R_L_Table <= "00";
                                   --x<= "0000000001";  done := False; dwr <= '1';
                                                        x<= "0000001";  done := False; dwr <= '1';
                                      State <= GetTkn;
                 end case;
                end if;

            when GetTkn =>
                        cntl_out <="0010";
--                      stout<="Get Token ";
                        en_W <= '0';
                        bus_rqst <= '0';
                        wr_en <= '1';
```

```
      R_L_Table <= "00";
      en_R <= '0';
      LT_addr <= "00000";
      if join_flg = '0' then
        wr_out <= "00";
      else
        wr_out <= wr_out;
      end if;
      R_L_Table <= "00";
      Ld_RD <= '0';
      s <= '0';
      ram_addr <= "000000";
      rst_f <= '0';
rst_r <= '0';
rst_LT <= '0';
if (stoploop = '1') then
  stopflag := true;
  state <= GetTkn;  -- break out the process loop
elsif ((finished = '1') and (nxt_lded='0') and (running=True)) then
       running := false;
       State <= Dummy;                    --handle finished proc
      elsif ((stopflag=true) and (finished = '1') and (nxt_lded='1')) then
       State <= StopL;
      elsif ((stopflag=false) and (finished = '1') and (nxt_lded='1')) then
       State <= SndPRT;                 --handle finished proc
      elsif (nxt_lded='0' and Cnt_token > "000000") then --Dequeue for processing
  State <= DeQ;
elsif (empty = '0' and in_delay = false) then
       re <= '1';                      --get token
       in_delay := true;
       Count <= Count;
       State <= GetTkn;
elsif (in_delay = true and buf_delay = false) then
  re <= '0';
  buf_delay := true;
  count <= Count;
  State <= GetTkn;
elsif (buf_delay = true) then
       if (cntl_in(24 downto 19))= Status then
         last_cntl_in <= cntl_in;
         State <= ChkStat;
  elsif (cntl_in(24 downto 19)) = Load_Table then
    last_cntl_in <= cntl_in;
    State <= Ld_Table;
  elsif (cntl_in(24 downto 19)) = Load_Thres then
    prog_flag <= cntl_in(5 downto 0); --ld threshold value
    time_s <= cntl_in(9 downto 6);         --ld sample time
    last_cntl_in <= cntl_in;
    State <= GetTkn;
  elsif (cntl_in(24 downto 19)) = Switch then
    temp3 <= cntl_in;
    last_cntl_in <= cntl_in;
    State <= PRam;                         --enter psuedo-RAM funct.
  elsif (cntl_in(24) = '0') then    --token rcvd
    if (Cnt_token /= "111111") then        --enque token
      en_W <= '1';
```

```
                    D_out(17 downto 10) <= cntl_in(23 downto 16);
                    D_out(9 downto 0) <= cntl_in(9 downto 0);
                    last_cntl_in <= cntl_in;
                    State <= GetTkn;
                   end if;
                  else
                   State <= GetTkn;                --invalid token read
                  end if;
                  buf_delay := false;
                  in_delay := false;
                     else
                        re <= '0';
                        State <= GetTkn;                    --repeat
                     end if;


               when StopL =>
                     cntl_out <="0011";
--                   stout<="Stop Loop ";
                          en_R <= '0'; en_W <='0';
                     s <= '0';
                     ram_addr <= "000000";
                     rst_f <= '0';
                  rst_r <= '0';
                  rst_LT <= '0';
                  re <= '0';
                  LT_addr <= "00000";
                       D_out <= "0000000000000000";
                       stopflag :=false;
                       if Snd_done = False then
                        Ld_Rd <= '1';
                        dwr <= '1';         -- enable write from LUT to controller
                        if first_val = true then
                         case R_L_Table is
                          when "00" => R_L_Table <= "10";
                                       State <= StopL;
                          when "01" => R_L_Table <= "10";
                                       State <= StopL;
                          when "10" => R_L_Table <= "11";
                                       State <= StopL;
                          when "11" => R_L_Table <= "11";
                                       outbuf0 <= dline_in;
                                       first_val := false;
                                       State <= StopL;
                          when others => R_L_Table <= "00";
                         end case;
                        else
                         R_L_Table <= "00";
                         outbuf1 <= Dline_in;
                         Ld_Rd <= '0';
                         Snd_done := True;
                         first_val := true;
                         State <= StopL;
                        end if;
                       else
                         bus_rqst <= '1';
```

```
                                Ld_Rd <='0';
                                R_L_Table <= "00";
                                if bus_grant = '1' then
                                  wr_en <= '0';
                                  line_out(20 downto 0) <= ('0'&outbuf1(11 downto
8)&"00000000"&outbuf1(7 downto 0));
                                  line_out(30 downto 24) <= PRT_addr;
                                  line_out(23 downto 21) <= outbuf0(13 downto 11); --time stamp
                                  line_out(31) <= '0';                  --hold field
                                  Snd_done := false;
                                  if nxt_lded = '1' then
                                    State <= Issue;
                                  else
                                    State <= GetTkn;
                                  end if;
                                else
                                   State <= StopL;                              --wait for bus
                                end if;
                              end if;



                        when DeQ =>
                              cntl_out <="0100";
--                            stout<="De-Queue  ";
                                    en_W <= '0';
                              s <= '0';
                                   ram_addr <= "000000";
                                   rst_f <= '0';
                              rst_r <= '0';
                              rst_LT <= '0';
                              bus_rqst <= '0';
                              LT_addr <= "00000";
                                   temp3 <= (others=>'0');
                                   D_out <= "00000000000000000000";
                                   en_R <= '1';
                                   LD_RD <= '1';
                                   nxt_lded <= '1';
                                   R_L_Table <= "01";
                                   re <= '0';
                                   if Join_flg = '1' then
                                     fin_join := true;
                                     wr_out <= wr_out;
                                   else
                                     fin_join := false;
                                     wr_out <= "00";
                                   end if;
                                   if (finished = '1') then
                                     State <= Issue;
                                   elsif (finished = '0') then
                                     State <= GetTkn;
                              end if;

                        when Issue =>
                            cntl_out <="0101";
--                                  stout<=" Issue    ";
```

```
        en_R <= '0'; en_W <= '0';
        wr_en <= '1';
        bus_rqst <= '0';
        nxt_lded <= '0';
        R_L_Table <= "00";
        s <= '0';
        ram_addr <= "000000";
        rst_f <= '0';
    rst_r <= '0';
    rst_LT <= '0';
    re <= '0';
    bus_rqst <= '0';
    LT_addr <= "00000";
        D_out <= "00000000000000000";
        if (join_flg='1' and cnt_token > "000000" and fin_join = false) then
          Instr_Rdy <= '0';
          State <= DeQ;                    --Issue another token
        elsif (join_flg='1' and cnt_token = "000000" and fin_join = false) then
          State <= GetTkn;                          --Other join tkn not available
          nxt_lded <= '0';
          Instr_Rdy <= '0';
        elsif ((join_flg = '0') or (join_flg='1' and fin_join = true)) then
         case (wr_out) is
                when "00" => Wr_out <= "01";  --snd 1st instr
                             Instr_Rdy <= '1';
                             State <= Issue;
                when "01" => if (snd_instr = '0' or iss_delay = False or is2_delay =
false) then
                               state <= Issue;
                               Wr_out <= Wr_out;
                               if iss_delay = true then
                                  is2_delay := true;    --2nd delay cycle
                               end if;
                               iss_delay := true;    --delay to allow PE to read instr.
                             else
                               if fin_join=true then --snd 2nd/3rd instrs
                            Wr_out <= "11";
                            Instr_Rdy <= '1';
                           else
                            Wr_out <= "10";
                            Instr_Rdy <= '1';
                           end if;
                           iss_delay := false;   --reset delay var.
                                is2_delay := false;
                                State <= Issue;
                               end if;
                when "10" => if (snd_instr = '0' or iss_delay = False or is2_delay =
false) then
                               Instr_Rdy <= '0';
                               Wr_out <= Wr_out;
                               if iss_delay = true then
                                  is2_delay := true;
                               end if;
                               iss_delay := true;
                               STATE <= Issue;
                             else
```

```
                                        Wr_out <= "00";
                                        iss_delay := false;
                                        is2_delay := false;
                                        running := True;
                                        fin_join := false;
                                        Instr_Rdy <= '0';
                                        if (Cnt_token = "000000") then
                                                State <= GetTkn;
                                        else
                                                State <= DeQ;
                                        end if;
                                    end if;
                                when "11" => if (snd_instr = '0' or iss_delay = False or is2_delay =
false) then
                                                state <= issue;
                                                Wr_out <= Wr_out;
                                                Instr_Rdy <= '0';
                                                if iss_delay = true then
                                                  is2_delay := true;
                                                end if;
                                                iss_delay := true;
                                            else
                                              Wr_out <= "10";
                                              Instr_Rdy <= '1';
                                              iss_delay := false;
                                              is2_delay := false;
                                           State <= Issue;
                                         end if;
                                when others => Wr_out <= "00";
                                               State <= GetTkn;
                            end case;
                          end if;

                when Dummy =>
                    cntl_out <="0110";
--                          stout<="  Dummy   ";
                          en_R <= '0'; en_W <='0';
                          wr_en <= '1';
                          bus_rqst <= '0';
                          s <= '0';
                          ram_addr <= "000000";
                          rst_f <= '0';
                      rst_r <= '0';
                      rst_LT <= '0';
                      LT_addr <= "00000";
                          D_out <= "0000000000000000";
                          Ld_Rd <= '1';
                          R_L_Table <= "01";
                          if stopflag = true then State <= StopL;
                          else State <= SndPRT;
                          end if;

                when SndPRT =>
                             cntl_out <="0111";
--                      stout<="Send PRT  ";
                             en_R <= '0'; en_W <='0';
```

```
                    s <= '0';
                    ram_addr <= "000000";
                    rst_f <= '0';
                rst_r <= '0';
                rst_LT <= '0';
                re <= '0';
                LT_addr <= "00000";
                    D_out <= "0000000000000000";
                    if Snd_done = False then
                     Ld_Rd <= '1';
                     dwr <= '1';          -- enable write from LUT to controller
                     if first_val = true then
                      case R_L_Table is
                       when "00" => R_L_Table <= "10";
                                    State <= SndPRT;
                       when "01" => R_L_Table <= "10";
                                    State <= SndPRT;
                       when "10" => R_L_Table <= "11";
                                    State <= SndPRt;
                       when "11" => R_L_Table <= "11";
                                    outbuf0 <= dline_in;
                                    first_val := false;
                                    State <= SndPRT;
                       when others => R_L_Table <= "00";
                      end case;
                     else
                      R_L_Table <= "00";
                      outbuf1 <= Dline_in;
                      Ld_Rd <= '0';
                      Snd_done := True;
                      first_val := true;
                      State <= SndPRT;
                     end if;
                    else
                      bus_rqst <= '1';
                      Ld_Rd <='0';
                      R_L_Table <= "00";
                      if bus_grant = '1' then
                       wr_en <= '0';
                       if comp = False then
                         --line_out(20 downto 0) <= (outbuf0(9 downto 5)&"00000000"&outbuf1(7
downto 0));
                                line_out(20 downto 0) <= (outbuf0(9 downto
5)&"00000000"&cntl_in(7 downto 0));
                          line_out(30 downto 24) <= PRT_addr;
                          line_out(23 downto 21) <= outbuf0(13 downto 11); --time stamp
                          line_out(31) <= outbuf0(10);              --hold field
                          if outbuf0(4 downto 0) = "00000" then  --check for 2nd token
                           comp := false;                          --only one tkn to snd
                                Snd_done := false;
                                if nxt_lded = '1' then
                                  State <= Issue;
                           else
                                  State <= GetTkn;
                           end if;
                          else
```

```
                                    State <= SndPRT;
                                  comp := True;
                                end if;
                              else
                                --line_out(20 downto 0) <= (outbuf0(4 downto 0)&"00000000"&outbuf1(7
downto 0));
                                     line_out(20 downto 0) <= (outbuf0(4 downto
0)&"00000000"&cntl_in(7 downto 0));
                                line_out(30 downto 24) <= PRT_addr;
                                line_out(23 downto 21) <= outbuf0(13 downto 11); --time stamp
                                line_out(31) <= outbuf0(10);
                                comp := false;
                                Snd_done := false;
                                if nxt_lded = '1' then
                                      State <= Issue;
                                else
                                      State <= GetTkn;
                                end if;
                              end if;
                            else
                              State <= SndPRT;                          --wait for bus
                            end if;
                          end if;

                when ChkStat =>
                    cntl_out <="1000";
--                            stout<="Check Stat";
                      re <= '0';
                            line_out(31) <= '0';
                            line_out(30 downto 24) <= PRT_addr;
                            line_out(23) <= '0';
                            line_out(22) <= sign;
                            line_out(21 downto 18) <= ITRC;
                            line_out(17) <= thl_flag;
                            line_out(16 downto 11) <= Cnt_token(5 downto 0);
                            line_out(10 downto 0) <= (others=>'0');
                            bus_rqst <= '1';
                            if bus_grant = '1' then
                              wr_en <= '0';
                              State <= GetTkn;
                            else
                              State <= ChkStat;
                            end if;

                when PRam =>
                    cntl_out <="1001";
--                            stout<=" PRam     ";
                            if (iter = false and delay = false) then
                              S <= '1'; re <='0';
                              ram_addr <= temp3(5 downto 0);
                              iter := true;
                              State <= PRam;
                            elsif (iter = true and delay = false) then
                              S <= '1';
                              ram_addr <= temp3(11 downto 6);
                              iter := false; delay := true;
```

```
                                State <= PRam;
                              elsif (delay = true) then
                                S <= '0';
                                temp3 <= (others=>'0');
                                delay := false;
                                State <= GetTkn;
                              end if;
            end case;
   end if;

end process;

end Cntl_Logic_arch;
```

Module Name: mapbuf.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mapbuf is
    port (din: in std_logic_vector(24 downto 0);
            clk: in std_logic;
            wr_en: in std_logic;
            rd_en: in std_logic;
        ainit: in std_logic;
        dout: out std_logic_vector(24 downto 0);
        full: out std_logic;
        empty: out std_logic);
end mapbuf;

architecture buf_body of mapbuf is
--depth should be atleast 2 times the CE having the most no. of processes.For eg:
--if CE0 has 10 processes and multiplier CE has 8 processes, then the depth should be atleast 10x2=20 or
19 downto 0
constant deep: integer := 50; --changed to 31 for app2 mat mult
type fifo_array is array(deep downto 0) of std_logic_vector(24 downto 0);
signal mem: fifo_array;
signal f1,e1 : std_logic;

begin
full<=f1;
empty<=e1;
process (clk, ainit)
variable startptr, endptr: natural range 0 to deep+1;
begin

 if clk'event and clk = '1' then
  if ainit='1' then
   startptr:=0;
   endptr:=0;
   f1<='0';
   e1<='1';
  end if;
  if wr_en = '1' then
```

```
  if f1 /='1' then
   mem(endptr) <= din;
   e1<='0';
   endptr:=endptr+1;
   if endptr>deep then endptr:=0;
   end if;
   if endptr=startptr then
    f1<='1';
   end if;
  end if;
 end if;

 if rd_en ='1' then
  if e1 /= '1' then
   dout <= mem(startptr);
   f1<='0';
   startptr:=startptr+1;
   if startptr > deep then startptr:=0;
   end if;
   if startptr=endptr then
    e1<='1';
   end if;
  end if;
 end if;
end if;
end process;
end buf_body;
```

Module Name: lut.vhd

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity LUT is
   generic ( Instr0 : integer := 156;
             Instr1 : integer := 48;
             Instr2 : integer := 152);
   port (
      R_L_Table: in STD_LOGIC_VECTOR (1 downto 0);
      Ld_Rd: in STD_LOGIC;
      Data: inout STD_LOGIC_VECTOR (15 downto 0);
      rst: in STD_LOGIC;
      clk : in STD_LOGIC;
      Wr_out  : in std_logic_vector (1 downto 0);
      W_en : out std_logic;
      addr: in STD_LOGIC_VECTOR (4 downto 0);
      time_stmp : in STD_LOGIC_VECTOR(2 downto 0);
      Proc_Num: in STD_LOGIC_VECTOR (4 downto 0);
      data_loc: in STD_LOGIC_VECTOR (7 downto 0); -- coming from the Q
      join_flg: buffer std_logic;
      Instr_out: out STD_LOGIC_VECTOR (15 downto 0);
            --  tab_1ntry : out std_logic_vector(4 downto 0);
            --  tab_addntry : out std_logic_vector ( 7 downto 0);
```

```
            --  tab_exitpn_ntry : out std_logic_vector( 3 downto 0);
                    tab_in_dbg : out std_logic
    );
end LUT;

architecture LUT_arch of LUT is

signal Last_Proc : std_logic_vector(7 downto 0);   --hold last data loc issued
signal Last_PN   : std_logic_vector(4 downto 0);   --hold last PN #
signal Snd_buf_PN: std_logic_vector(4 downto 0);   --hold PN# of outbuffer
type Entry is record
            H_fld: std_logic;                       --Hold bit of entry
            J_fld: std_logic;                       --Proc is a join op
            PN : std_logic_vector(4 downto 0);    --Process Number
            Inst_addr : std_logic_vector(7 downto 0);  --address of 1st instr.
            Nxt_PN0 : std_logic_vector(4 downto 0);       --Next PN
            Nxt_PN1 : std_logic_vector(4 downto 0);       --PN used if a fork
            Exit_PN : std_logic_vector(3 downto 0);       --PN after exit the process loop
end record;
type table is array(23 downto 0) of entry;
signal L_table : table;
-- changing to just one entry for dbugging

--signal L_table : entry;
--variable L_table : table;
signal tab_out, tab_in : std_logic;
signal temp_data  : std_logic_vector(15 downto 0);

-- ADDED TO DBUG
signal temp_data_in1,temp_data_in2 :std_logic_vector (15 downto 0);
-------------

--constant Ldreg_data : std_logic_vector(31 downto 10):= "11111111000011110000000";
--constant LdPC : std_logic_vector(31 downto 10):= "11110000111111111000000";
signal Snd_buf_Inst0, Snd_buf_Inst1 : std_logic_vector(15 downto 0);
signal last_time_stmp, Snd_buf_tmstp : std_logic_vector(2 downto 0);
signal Snd_buf_Inst2 : std_logic_vector(15 downto 0);
signal Ldreg_data, LdPC,Ldreg2_data : std_logic_vector(15 downto 8);
--signal l1, l2, l0 : unsigned(15 downto 8);

-- signals added for dbugging
--signal tab_1ntry : std_logic_vector(4 downto 0);

begin
--l0 <= CONV_unsigned(Instr0, 8);
--l1 <= Conv_unsigned(Instr1, 8);
--l2 <= Conv_unsigned(Instr2, 8);
--Ldreg_data <= Conv_std_logic_vector(l0, 8);
--LdPC <= Conv_std_logic_vector(l1, 8);
--Ldreg2_data <= Conv_std_logic_vector(l2, 8);

 -- added for dbugging
--tab_1ntry <=L_table(0).PN;
--tab_addntry <= L_table(0).Inst_addr;
--tab_exitpn_ntry <= L_table(0).Exit_PN;
------------
```

```
Ldreg_data <= Conv_std_logic_vector(Instr0, 8);
LdPC <= Conv_std_logic_vector(Instr1, 8);
Ldreg2_data <= Conv_std_logic_vector(Instr2, 8);

Snd_buf_Inst0(15 downto 8) <= Ldreg_data;
Snd_buf_Inst1(15 downto 8) <= LdPC;
Snd_buf_Inst2(15 downto 8) <= Ldreg2_data;

read: process (clk, R_L_Table, Ld_Rd, rst)              --decode queue tokens
          begin                                                 --and send nxt tkn to cntrlr
   if rst = '1' then
                Snd_buf_Inst1(7 downto 0) <= (others=>'0');
                Snd_buf_Inst0(7 downto 0) <= (others=>'0');
                Snd_buf_Inst2(7 downto 0) <= (others=>'0');
                Join_flg <= '0';
                Snd_buf_tmstp <= (others=> '0');
                Last_Proc <= (others=>'0');
                last_PN <= (others=>'0');
                last_time_stmp <= (others=> '0');
                Snd_buf_PN <= (others=>'0');
                temp_data <= (others=>'0');
          elsif (clk'event and clk='1') then

            if Ld_Rd = '1' then
                    case (R_L_Table) is
                            when "01"=>
                                                            --Issue to PE
                                    if join_flg = '0' then
                                        Last_Proc <= Snd_buf_Inst0(7 downto 0);
                                        last_PN <= Snd_buf_PN;
                                        last_time_stmp <= Snd_buf_tmstp;
                                        Snd_buf_Inst0(7 downto 0) <= data_loc;
                                        Snd_buf_PN <= Proc_num;
                                        Snd_buf_tmstp <= time_stmp;
                                    end if;
                                    --for x in 0 to 9 loop
                                    for x in 0 to 22 loop
                                    -- some changes for dbugging
                                            if Proc_Num = L_table(x).PN then
                                            --if Proc_Num = L_table.PN then
                                         if join_flg = '0' then
                                            Snd_buf_Inst1(7 downto 0) <= L_table(x).Inst_addr;
                                                    --Snd_buf_Inst1(7 downto 0) <= L_table.Inst_addr;
                                                    if L_table(x).J_fld = '1' then
                                                    --if L_table.J_fld = '1' then
                                                join_flg <= '1';
                                               else
                                                 join_flg <= '0';
                                         end if;
                                         else                       --join op, issue another data loc
                                          Snd_buf_Inst2(7 downto 0) <= data_loc;
                                          join_flg <= '0';
                                         end if;
                                        end if;
                                    end loop;
```

```
                        when "10"=>
                                Join_flg <='0';
                                --for z in 0 to 9 loop
                                for z in 0 to 22 loop                   --send to cntrlr
                                  if Last_PN = L_table(z).PN then
                                                                        --next token PN's
                                    temp_data(4 downto 0) <= L_table(z).Nxt_PN1;
                                    temp_data(9 downto 5)<= L_table(z).Nxt_PN0;
                                    temp_data(10) <= L_table(z).H_fld;
                                    temp_data(13 downto 11) <= last_time_stmp;
                                    temp_data(15 downto 14) <= "00";
                                  end if;
                                end loop;
                                --for z in 0 to 9 loop                  --send to cntrlr
                                --  if Last_PN = L_table.PN then
                                                                        --next token PN's
                                  --  temp_data(4 downto 0) <= L_table.Nxt_PN1;
                                   -- temp_data(9 downto 5)<= L_table.Nxt_PN0;
                                    --temp_data(10) <= L_table.H_fld;
                                    --temp_data(13 downto 11) <= last_time_stmp;
                                    --temp_data(15 downto 14) <= "00";
                                  --end if;
                        --          end loop;
                        when "11"=>
                                join_flg <= '0';

                        when others =>  --for y in 0 to 9 loop                  --send to cntrlr
                                          for y in 0 to 22 loop
                                  if Last_PN = L_table(y).PN then
                                    temp_data(15 downto 12) <= "0000";
                                    temp_data(11 downto 8) <= L_table(y).Exit_PN;
                                    temp_data(7 downto 0) <= Last_Proc;  --data location
                                  end if;
                                        end loop;
                                --for y in 0 to 9 loop                  --send to cntrlr
                                  --if Last_PN = L_table.PN then
                                    --temp_data(15 downto 12) <= "0000";
                                    --temp_data(11 downto 8) <= L_table.Exit_PN;
                                    --temp_data(7 downto 0) <= Last_Proc;  --data location
                                  --end if;
                                --end loop;
                                temp_data <= temp_data;
                                --join_flg <= '0';
                        end case;
                    end if;
                end if;
end process;

-- control for tab_out tri-state
tab_out <= '1' when (Ld_Rd ='1' and (R_L_table = "10" or R_L_table = "11")) else
                '0';

--data_load : process (tab_out, tab_in, data, temp_data)            --trnfr data to/from cntrlr
            --begin
            --          if tab_in = '1'then
            --            if R_L_table ="01"
```

```
        --          then temp_data_in1 <= data;
        --          elsif R_L_table ="10"
        --              then temp_data_in2 <= data;
        --              --end if;--else data <= (others=> 'Z');
        --          end if;
        --      elsif tab_out ='1' then data <= temp_data;
        --      else data <= (others=> 'Z');
        --      end if;
--end process;
data_load : process (clk,tab_out, tab_in, data, temp_data)      --trnfr data to/from cntrlr
        begin
   if(clk'event and clk='0') then
                if tab_in = '1'then
                  if R_L_table ="01" then
                        temp_data_in1 <= data;
                  elsif R_L_table ="10" then
                    temp_data_in2 <= data;
                      --end if;--else data <= (others=> 'Z');
                  end if;
                elsif tab_out ='1' then
                        data <= temp_data;
                else
                        data <= (others=> 'Z');
                end if;
        end if;

end process;

load: process (rst, clk, Ld_Rd, R_L_table)              --Initialize table entries
        variable val : integer;
        begin
                if rst = '1' then
                        --for x in 0 to 9 loop
                          for x in 0 to 22 loop
                                L_table(x).H_fld <= '0';
                                L_table(x).J_fld <= '0';
                                L_table(x).PN <= "00000";
                                L_table(x).Inst_addr <= (others=>'0');
                                L_table(x).Nxt_PN0 <= "00000";
                                L_table(x).Nxt_PN1 <= "00000";
                                L_table(x).Exit_PN <= "0000";
                        end loop;
                --      L_table.H_fld <= '0';
                --              L_table.J_fld <= '0';
                --              L_table.PN <= "00000";
                --              L_table.Inst_addr <= (others=>'0');
                --              L_table.Nxt_PN0 <= "00000";
                --              L_table.Nxt_PN1 <= "00000";
                --              L_table.Exit_PN <= "0000";
                elsif (clk'event and clk='1') then
                    if Ld_Rd = '0' then
                        case (addr) is
                        when "00000" => val :=0;
                                when "00001" => val :=1;
                                when "00010" => val :=2;
                                when "00011" => val :=3;
```

```
                                        when "00100" => val :=4;
                                        when "00101" => val :=5;
                                        when "00110" => val :=6;
                                        when "00111" => val :=7;
                                        when "01000" => val :=8;
                                        when "01001" => val :=9;
                                        when "01010" => val := 10;
                                        when "01011" => val := 11;
                                        when "01100" => val := 12;
                                        when "01101" => val := 13;
                                        when "01110" => val := 14;
                                        when "01111" => val := 15;
                                        when "10000" => val := 16;
                                        when "10001" => val := 17;
                                        when "10010" => val := 18;
                                        when "10011" => val := 19;
                                        when "10100" => val := 20;
                                        when "10101" => val := 21;
                                        when "10110" => val := 22;
                                        when "10111" => val := 23;
                                        when "11000" => val := 24;
                                        when "11001" => val := 25;
                                        when "11010" => val := 26;
                                        when "11011" => val := 27;
                                        when "11100" => val := 28;
                                        when "11101" => val := 29;
                                        when "11110" => val := 30;
                                        when "11111" => val := 31;
                                        when others => val :=0;
                                end case;
                                case (R_L_table) is
                                  when "01" =>
                                        L_table(val).PN <= temp_data_in1(15 downto 11);
                                        L_table(val).Nxt_PN0 <= temp_data_in1(10 downto 6);
                                        L_table(val).Nxt_PN1 <= temp_data_in1(5 downto 1);
                                        L_table(val).H_fld <= temp_data_in1(0);
                                  when "10" =>
                                        L_table(val).Exit_PN <= temp_data_in2(12 downto 9);
                                    L_table(val).J_fld <= temp_data_in2(8);
                                        L_table(val).Inst_addr <= temp_data_in2(7 downto 0);
                                  when others => L_table(val).Nxt_PN1 <=L_table(val).Nxt_PN1;

                                 end case;
                           end if;
                        end if;
end process;

--control for tab_in tri-state
tab_in <= '1' when (Ld_Rd='0' and R_L_table /="00") else
            '0';
--control for wr_out tri-state
W_en <= '1' when (wr_out = "01" or wr_out = "10" or wr_out = "11") else
          '0';
tab_in_dbg <=tab_in;
send_instr: process (clk, wr_out,Snd_buf_Inst0,Snd_buf_Inst1,Snd_buf_Inst2)   --send instr's to PE
          begin
```

```
                case (wr_out) is
                        when "01" =>
                              Instr_out <= Snd_buf_Inst0; --send 1st instr
                                        --Instr_out <= "1001110000000100";
                        when "10" =>
                                 Instr_out <= Snd_buf_Inst1; --send 2nd instr
                                 --Instr_out <= "0011000000000011";
                        when "11" =>
                              Instr_out <= Snd_buf_Inst2; --send other join data loc
                        when others => Instr_out <= (others=>'0');
                end case;
end process;

end LUT_arch;
```

Module Name : Queue.vhd

```
-- QUEUE.vhd used in synthesis simulation.
-- Top level design for FIFO model

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity queue is       -- total queue source code
          port ( clk, enw, rst_f,rst_r,enr,s:in std_logic;
                time_s: in std_logic_vector(3 downto 0);
                din: in std_logic_vector(17 downto 0);
                ram_add: in std_logic_vector(5 downto 0);
                prog_flag: in std_logic_vector(5 downto 0);

                error: inout std_logic;

                sign: out std_logic;
                ITRC: out std_logic_vector(3 downto 0);
                th_flag: out std_logic;
                count_token:inout std_logic_vector(5 downto 0);
                dout: out std_logic_vector(17 downto 0));
end queue;

architecture queue_body of queue is


component rate
           port ( Clk, Enw, Rst,
          error_full: in std_logic;
          time_s: in std_logic_vector(3 downto 0);
                sign: out std_logic;
                ITRC: out std_logic_vector(3 downto 0));

end component;

component FIFO_block_syn generic(N: integer := 18);
           port (
```

```
                    din: in  std_logic_vector(N-1 downto 0);
                    ENR: in std_logic;
                    ENW: in std_logic;
                    clk, Rst: in std_logic;
                    ram_add: in std_logic_vector(5 downto 0);
                    s:in std_logic;
                    prog_flag: in std_logic_vector(5 downto 0);
                    ENR_out: out std_logic;
                    ENW_out: out std_logic;
                    error: out std_logic;
                    error_full: inout std_logic;
                    th_flag: out std_logic;
                    count_token: inout std_logic_vector(5 downto 0);
                    wptr_out: out std_logic_vector (5 downto 0);
                    rptr_out: out std_logic_vector (5 downto 0);
                    dout: out std_logic_vector(N-1 downto 0));
end component;

component ram
   port (waddr: in std_logic_vector(5 downto 0);
            datain: in std_logic_vector(17 downto 0);
        clk: in std_logic;
            wren: in std_logic;
            rden: in std_logic;
            raddr: in std_logic_vector(5 downto 0);
        dataout: out std_logic_vector(17 downto 0));
end component;

signal error_full: std_logic;
signal dout_ram: std_logic_vector (17 downto 0);
signal dout_FIFO: std_logic_vector (17 downto 0);
signal din_ram: std_logic_vector (17 downto 0);
signal ENR_out, ENW_out: std_logic;
signal wptr_out, rptr_out: std_logic_vector(5 downto 0);

begin

rate1: rate port map (Clk,Enw,Rst_r,error_full,time_s,sign,ITRC);

FIFO_syn1: FIFO_block_syn port map(dout_ram,ENR,ENW,clk,Rst_f,ram_add,s,prog_flag,ENR_out,
                        ENW_out,error,error_full,th_flag,count_token,wptr_out,rptr_out,
                        dout_FIFO);

ram1 : ram port map(wptr_out,din_ram,clk,ENW_out,ENR_out,rptr_out,dout_ram);

 process(s,dout_FIFO,din,dout_ram)
 begin
  case s is
   when '1' => din_ram <= dout_FIFO; dout <= (others => '0');
   when others => din_ram <= din; dout <= dout_ram;
  end case;
 end process;

end queue_body;
```

Module Name: fifo.vhd

```
-- FIFO_block.vhd used in synthesis simulation.
library ieee;
use ieee.std_logic_1164.all;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FIFO_block_syn is generic(N: integer := 18);
        port (
                din: in  std_logic_vector(N-1 downto 0);
                ENR: in std_logic;
                ENW: in std_logic;
                clk, Rst: in std_logic;
                ram_add: in std_logic_vector(5 downto 0);
                s:in std_logic;
         prog_flag: in std_logic_vector(5 downto 0);
                ENR_out: out std_logic;
                ENW_out: out std_logic;
                error: out std_logic;
                error_full: inout std_logic;
                th_flag: out std_logic;
                count_token: inout std_logic_vector(5 downto 0);
                wptr_out: out std_logic_vector (5 downto 0);
                rptr_out: out std_logic_vector (5 downto 0);
                dout: out std_logic_vector(N-1 downto 0));
end FIFO_block_syn;

architecture FIFO_block_body of FIFO_block_syn is

-------------------------------------------------------------------------------------------------------------------------
--
-- Signals used in the Error detection unit block
--
signal error_empty: std_logic;

-------------------------------------------------------------------------------------------------------------------------
--
-- Signals used in the FCU block
--
signal flag_fcu1,flag_fcu2,flag_fcu3,flag_fcu4,
flag_fcu5: std_logic;
-------------------------------------------------------------------------------------------------------------------------
--
-- Signals used when the pseudo-RAM function is evoked
--
signal ASE1,ASE2: std_logic_vector(5 downto 0);
signal dout_ASE : std_logic_vector(5 downto 0);

signal RAM1,RAM2: std_logic_vector(17 downto 0);
signal dout_RAM1, dout_RAM2: std_logic_vector(17 downto 0);
signal din_RAM1, din_RAM2: std_logic_vector(17 downto 0);
-------------------------------------------------------------------------------------------------------------------------
signal rptr,wptr: std_logic_vector(5 downto 0);

begin

 process (wptr, rptr, s, ram_add, dout_ASE)
 begin
```

```
 case s is
  when '1' => rptr_out <= ram_add; wptr_out <= dout_ASE;
  when others => rptr_out <= rptr; wptr_out <= wptr;
 end case;
end process;

process(rst,s,flag_fcu1,flag_fcu2,flag_fcu3, flag_fcu4,flag_fcu5,ENR,ENW,error_empty,error_full)
begin
 if rst = '1' then
  ENW_out <= '0'; ENR_out <= '0';
 else
  if s = '1' then
    if flag_fcu1 = '0' and flag_fcu2 = '0' and
    flag_fcu3 = '0' and flag_fcu4 = '0' and flag_fcu5 = '0' then
      ENR_out <= '1'; ENW_out <='0';
    elsif flag_fcu1 = '1' and flag_fcu2 = '0' and
    flag_fcu3 = '0' and flag_fcu4 = '0' and flag_fcu5 = '0' then
      ENR_out <= '1'; ENW_out <= '0';
    elsif flag_fcu1 = '1' and flag_fcu2 = '1' and
    flag_fcu3 = '1' and flag_fcu4 = '0' and flag_fcu5 = '0' then
      ENR_out <= '0'; ENW_out <= '1';
    elsif flag_fcu1 = '1' and flag_fcu2 = '1' and
    flag_fcu3 = '1' and flag_fcu4 = '1' and flag_fcu5 = '0' then
      ENR_out <= '0'; ENW_out <= '1';
    else
     ENR_out <= '0'; ENW_out <= '0';
    end if;
  else
    ENR_out <= ENR and (not error_empty); ENW_out <= ENW and (not error_full);
  end if;
 end if;
end process;

ASE_block:process(rst,s,clk)
begin
 if rst = '1' then
  ASE1 <= (others => '0'); ASE2 <= (others => '0');
  dout_ASE <= (others => '0');
 else
  if s = '1' then
   if clk'event  and clk = '1' then
    if flag_fcu1 = '0' and flag_fcu2 = '0' and
    flag_fcu3 = '0' and flag_fcu4 = '0' then
     ASE1 <= ram_add;
    elsif flag_fcu1 = '1' and flag_fcu2 = '0' and
    flag_fcu3 = '0' and flag_fcu4 = '0' then
     ASE2 <= ram_add;
    elsif flag_fcu1 = '1' and flag_fcu2 = '1' and
    flag_fcu3 = '0' and flag_fcu4 = '0' then
     dout_ASE <= ASE2;
    elsif flag_fcu1 = '1' and flag_fcu2 = '1' and
    flag_fcu3 = '1' and flag_fcu4 = '0' then
     dout_ASE <= ASE1;
    end if;
   end if;
  end if;
```

```
 end if;
end process;

RAM_block:process(rst,clk)
begin
 if rst = '1' then
  RAM1 <= (others => '0'); RAM2 <= (others =>'0');
  dout<= (others => '0');
 else
  if clk'event and clk = '1' then
   if s = '1' then
    if flag_fcu1 = '0' and flag_fcu2 = '0' and
    flag_fcu3 = '0' and flag_fcu4 ='0' then
     RAM1 <= din;
    elsif flag_fcu1 = '1' and flag_fcu2 = '0' and
    flag_fcu3 = '0' and flag_fcu4 ='0' then
     ram2 <= din;
    elsif flag_fcu1 = '1' and flag_fcu2 = '1' and
    flag_fcu3 = '0' and flag_fcu4 ='0' then
     dout <= RAM1;
    elsif flag_fcu1 = '1' and flag_fcu2 = '1' and
    flag_fcu3 = '1' and flag_fcu4 ='0' then
     dout <= RAM2;
    else
     RAM2 <= (others => '0'); RAM1 <= (others => '0');
    end if;
   end if;
  end if;
 end if;
end process;

WAP_RAP: process (rst,clk)
begin
 if rst = '1' then
  wptr <= (others => '0'); rptr <= (others => '0');
 else
  if clk'event and clk = '1' then
   if s= '0' then
    if enw = '1' and error_full = '0' then
     if wptr /= "111111" then
      wptr <= wptr + "000001";
     else
      wptr <= (others => '0');
     end if;
    end if;

    if enr = '1' and error_empty = '0' then
     if rptr /= "111111" then
      rptr <= rptr + "000001";
     else
      rptr <= (others => '0');
     end if;
    end if;
   end if;
  end if;
 end if;
```

```
end process;

error <= error_full or error_empty;

EDU: process(rst,wptr,rptr,enw,enr,s,count_token)
begin
 if rst = '1' then
  error_full <= '0'; error_empty <= '0';
 else
   if s = '0' then
    if wptr = rptr and enw = '1' and enr = '0'
    and count_token /= "000000" then
     error_full <= '1'; error_empty <= '0';
    elsif rptr = wptr and count_token /= "100000"
    and enw = '0' and enr = '1' then
     error_full <= '0'; error_empty <= '1';
    else
     error_full <= '0'; error_empty <= '0';
    end if;
   end if;
 end if;
end process;

TCU: process(rst,clk)
begin
 if rst = '1' then
  count_token <= (others => '0');
 else
  if clk'event and clk = '1' then
   if s = '0' then
    if enw = '1' and enr = '0' then
     if count_token /= "100000" and error_full /= '1' then
      count_token <= count_token + "000001";
     end if;
    elsif enw = '0' and enr = '1' then
     if count_token /= "000000" and error_empty /= '1' then
      count_token <= count_token - "000001";
     end if;
    end if;
   end if;
  end if;
 end if;
end process;

PTU: process(rst,s,prog_flag,count_token)
begin
 if rst = '1' then
  th_flag <= '0';
 else
  if s = '0' then
   if count_token >= prog_flag then
    th_flag <= '1';
   else
    th_flag <= '0';
   end if;
  end if;
```

```
 end if;
end process;

FCU: process(clk,rst)
begin
 if rst = '1' then
  flag_fcu1 <= '0'; flag_fcu2 <= '0';
  flag_fcu3 <= '0'; flag_fcu4 <= '0';
  flag_fcu5 <= '0';
 else
  if clk'event and clk = '1' then
   if s = '1' then
    if flag_fcu1 = '0' and flag_fcu2 = '0' and
    flag_fcu3 = '0' and flag_fcu4 = '0' and flag_fcu5 = '0' then
     flag_fcu1 <= '1';
    elsif flag_fcu1 = '1' and flag_fcu2 = '0' and
    flag_fcu3 = '0' and flag_fcu4 = '0' and flag_fcu5 = '0' then
     flag_fcu2 <= '1';
    elsif flag_fcu1 = '1' and flag_fcu2 = '1' and
    flag_fcu3 = '0' and flag_fcu4 = '0' and flag_fcu5 = '0' then
     flag_fcu3 <= '1';
    elsif flag_fcu1 = '1' and flag_fcu2 = '1' and
    flag_fcu3 = '1' and flag_fcu4 = '0' and flag_fcu5 = '0' then
     flag_fcu4 <= '1';
    elsif flag_fcu1 = '1' and flag_fcu2 = '1' and
    flag_fcu3 = '1' and flag_fcu4 = '1' and flag_fcu5 = '0' then
     flag_fcu5 <= '1';
    end if;
   else
    flag_fcu1 <= '0'; flag_fcu2 <= '0';
    flag_fcu3 <= '0'; flag_fcu4 <= '0';
    flag_fcu5 <= '0';
   end if;
  end if;
 end if;
end process;



end FIFO_block_body;
```

Module Name: ram.vhd

```
-- RAM.vhd
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE STD.TEXTIO.ALL;

entity ram is
    port (waddr: in std_logic_vector(5 downto 0);
              datain: in std_logic_vector(17 downto 0);
          clk: in std_logic;
              wren: in std_logic;
```

```
            rden: in std_logic;
            raddr: in std_logic_vector(5 downto 0);
          dataout: out std_logic_vector(17 downto 0));
end ram;

architecture ram_body of ram is

constant deep: integer := 63;
type fifo_array is array(deep downto 0) of std_logic_vector(17 downto 0);
signal mem: fifo_array;

signal waddr_int: integer range 0 to 63;
signal raddr_int: integer range 0 to 63;

begin
waddr_int <= conv_integer(waddr);
raddr_int <= conv_integer(raddr);

process (clk)
begin
 if clk'event and clk = '1' then
  if wren = '1' then
   mem(waddr_int) <= datain;
  end if;
 end if;
end process;
dataout <= mem(raddr_int);
end ram_body;
```

Module Name : rate.vhd

```
-- This is the vhdl description of the rate_block

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity rate is
          port ( Clk, Enw, Rst,
                  error_full: in std_logic; -- active high reset(synchronous) and write enable
                  time_s: in std_logic_vector(3 downto 0); -- This specify the time time period one wants to
                                                           -- use for calculating the difference in rate for
                                                           -- 2 time period.

                sign: out std_logic;                       -- If the sign is 0 it means rate decreases and if
                                                           -- it is 1 than it means the rate has increased.

                ITRC: out std_logic_vector(3 downto 0));  -- ITRC gives us the rate comparison of 2 time.
slices

end rate;

architecture body_rate of rate is
```

```
signal time_s_temp: std_logic_vector(3 downto 0);

signal count_clk : std_logic_vector(3 downto 0); -- Output from the clock counter block that tells how
                                                 -- many clock cycle has passed.

signal write_storeRef : std_logic; -- Control signal that acts as the write enable signal for storeRef memory
        -- element.

signal count_t : std_logic_vector(3 downto 0); -- Output from the token_counter block that gives
-- information on how many control token is written into the
-- memory array within a time slice.


signal storeRef : std_logic_vector(3 downto 0); -- Output of the store_ref_rate block and is used as the
    -- reference to count the build up rate.

signal storeComp,fill_flag : std_logic_vector(3 downto 0); -- Output of the store_comp_rate block and is
-- used as the comparator value to count the
-- ITRC.

signal mem_stack: std_logic_vector(7 downto 0);
signal last : std_logic;
signal time_s_temp_lessOne : integer range 0 to 8;


begin

 CCU:process(clk,rst,time_s)                    -- This section describes the clock counter unit block
 begin
  if rst = '1' then
   time_s_temp <= time_s; -- store the desired time period
   count_clk <= (others => '0');
   write_storeRef <= '0';

   case time_s is
    when "0000" => time_s_temp_lessOne <= 0;
    when "0001" => time_s_temp_lessOne <= 0;
    when "0010" => time_s_temp_lessOne <= 0;
    when "0011" => time_s_temp_lessOne <= 1;
    when "0100" => time_s_temp_lessOne <= 2;
    when "0101" => time_s_temp_lessOne <= 3;
    when "0110" => time_s_temp_lessOne <= 4;
    when "0111" => time_s_temp_lessOne <= 5;
    when "1000" => time_s_temp_lessOne <= 6;
    when others => time_s_temp_lessOne <= 0;
   end case;

  elsif (Clk'event and Clk = '1') then
   if error_full = '0' then
    if (count_clk = time_s_temp) then
     count_clk <= "0001";
    else
     if count_clk /= "1000" then
      count_clk <= count_clk + "0001";
     end if;
    end if;
```

```
   if (count_clk = (time_s_temp -"0001")) then
    write_storeRef <= '1';
   else
    write_storeRef <= '0';
   end if;
  end if;
 end if;
end process;

WTCU: process(clk,rst)                          -- This section describes the write token counter unit block
begin
 if rst = '1' then
  count_t <= (others => '0');
 elsif clk'event and clk = '1' then
  if error_full = '0' then
   if count_clk = time_s_temp then
    if enw = '1' then
     count_t <= "0001";
    else
     count_t <= "0000";
    end if;
   else
    if enw = '1' then
     if count_t /= "1000" then
      count_t <= count_t + "0001";
     end if;
    end if;
   end if;
  end if;
 end if;
end process;

SE2:process(clk,rst)  -- This section describes the SE1 block that is used to store the RITB.
begin
 if rst = '1' then
  storeRef <= (others => '0');
 elsif clk'event and clk = '1' then
  if error_full = '0' then
   if write_storeRef = '1' then
    storeRef <= count_t;
   end if;
  end if;
 end if;
end process;


SE3: process(clk,rst)                           -- This section describes the SE3 block that is used to
                                                -- store and determine the NITB.
begin
 if rst = '1' then
  storeComp <= (others => '0');
  fill_flag <= (others => '0');
 elsif clk'event and clk = '1' then
  if error_full = '0' then
   if fill_flag /= time_s_temp then
    fill_flag <= fill_flag + "0001";
```

```
    if enw = '1' and last = '0' then
     storeComp <= storeComp + "0001";
    end if;
   else
    if enw = '1' and storeComp /= time_s_temp and last = '0' then
     storeComp <= storeComp + "0001";
    elsif enw = '0' and storeComp /= "0000" and last = '1' then
     storeComp <= storeComp - "0001";
    end if;
   end if;
  end if;
 end if;
end process;

AU: process (storeComp, storeRef, Rst, error_full)  -- This section describes the arithetic unit block that
                                                    -- is used to count the input token buildup
 begin
  if Rst = '1' then
   sign <= '0'; ITRC <= (others => '0');
  else
   if error_full = '0' then
    if storeRef > storeComp then
     ITRC <= storeRef - storeComp;
     sign <= '0';
    elsif storeRef = storeComp then
     ITRC <= (others => '0');
     sign <= '0';
    else
     ITRC <= storeComp - storeRef;
     sign <= '1';
    end if;
   end if;
  end if;
 end process;

 process(clk,rst)
 begin
  if rst = '1' then
   last <= '0'; mem_stack <= (others => '0');
  elsif clk'event and clk = '1' then
   if error_full = '0' then
    last <= mem_stack(time_s_temp_lessOne);
    if enw = '1' then
     mem_stack <= mem_stack(6 downto 0) & '1';
    else
     mem_stack <= mem_stack(6 downto 0) & '0';
    end if;
   end if;
  end if;
 end process;

end body_rate;
```

Module Name: divpe.vhd

```
--  Code for Divider Processor for HDFCA project
--  File: divpe.vhd
```

```
-- synopsys translate_off

Library XilinxCoreLib;

-- synopsys translate_on

--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Divpe is
        port (Cntrlr_bus : in std_logic_vector(15 downto 0);
            Snd_I : out std_logic;
            clk : in std_logic;
            rst : in std_logic;
            Instr_rdy : in std_logic;
            Fin : out std_logic;
            Data_bus : inout std_logic_vector(15 downto 0);
            Bus_req : out std_logic;
            Bus_gnt : in std_logic;
            Addr : out std_logic_vector(6 downto 0);
            R_W : buffer std_logic;
                --R_W : inout std_logic;
                loc_bus_dbug : out std_logic_vector(7 downto 0);
                Iaddr_bus_dbug : out std_logic_vector(7 downto 0);
                Iaddr_dbug :  out std_logic_vector(7 downto 0);
                R2_out_dbug : out std_logic_vector( 7 downto 0);
                Imem_bus_dbug : out std_logic_vector(15 downto 0 )
                --LR2_dbug : out std_logic
                );
end Divpe;


architecture dpe of Divpe is

-----------------------------------------------------------------
-- This file was created by the Xilinx CORE Generator tool, and     --
-- is (c) Xilinx, Inc. 1998, 1999. No part of this file may be      --
-- transmitted to any third party (other than intended by Xilinx)   --
-- or used without a Xilinx programmable or hardwire device without --
-- Xilinx's prior written permission.                               --
-----------------------------------------------------------------

component div1
        port (
        dividend: IN std_logic_VECTOR(15 downto 0);
        divisor: IN std_logic_VECTOR(15 downto 0);
        quot: OUT std_logic_VECTOR(15 downto 0);
        remd: OUT std_logic_VECTOR(15 downto 0);
        c: IN std_logic);
end component;


-----------------------------------------------------------------
```

```
-- This file was created by the Xilinx CORE Generator tool, and     --
-- is (c) Xilinx, Inc. 1998, 1999. No part of this file may be      --
-- transmitted to any third party (other than intended by Xilinx)   --
-- or used without a Xilinx programmable or hardwire device without --
-- Xilinx's prior written permission.                               --
-----------------------------------------------------------------------

component div_imem
        port (
        addr: IN std_logic_VECTOR(3 downto 0);
        clk: IN std_logic;
        din: IN std_logic_VECTOR(15 downto 0);
        dout: OUT std_logic_VECTOR(15 downto 0);
        we: IN std_logic);
end component;

component add_subber8
        port (
        A: IN std_logic_VECTOR(7 downto 0);
        B: IN std_logic_VECTOR(7 downto 0);
        C_IN: IN std_logic;
        C_OUT: OUT std_logic;
        ADD_SUB: IN std_logic;
        Q_OUT: OUT std_logic_VECTOR(7 downto 0));
end component;

signal Imem_bus, R0_out, R1_out, Inst_in, Inst_out : std_logic_vector(15 downto 0);
signal R2_out, Data_loc1, Data_loc2 : std_logic_vector(7 downto 0);
signal s2, s1, s0, s3 ,s4, s5, s6, s7 : std_logic;
signal Div_out, mux2_out, adder_out : std_logic_vector(7 downto 0);
signal mux1_out, result : std_logic_vector(15 downto 0);
signal div_en, ld_d1, ld_d2, ld_iaddr : std_logic;
signal loc_bus, Iaddr, Iaddr_bus : std_logic_vector(7 downto 0);
constant GoDiv : std_logic_vector(7 downto 0) := "11111111";
constant StoreDL : std_logic_vector(7 downto 0) := "10001000";
type OP_state is (reset,Getop,O1,O2,O3,O4,O5,O5A,O5B,O5C,O6,O7,O8,O9,O10);
signal OP : OP_state;
signal LR2, LR1, Ci, LR0, R2_rst, ld_rslt, I_R_W : std_logic;
signal qout_out, remd_out, rem_rslt : std_logic_vector(15 downto 0);
signal mux5_out, mux6_out, MUX4_OUT : std_logic_vector(7 downto 0);
signal delay : std_logic_vector(19 downto 0);
signal one, zero : std_logic;
signal test :string (1 to 10);

begin
one <= '1';
zero <= '0';


-- added for dbugging
loc_bus_dbug <= loc_bus;
Iaddr_bus_dbug <= Iaddr_bus;
Iaddr_dbug <= Iaddr;
R2_out_dbug <= R2_out;
Imem_bus_dbug <= Imem_bus;
--LR2_dbug <=LR2;
```

```
----
ADD5 : add_subber8
          port map ( A =>R2_out, B =>mux2_out, C_IN => Ci, C_OUT => open,
                ADD_SUB =>one, Q_OUT =>adder_out);

D2 : div1 port map (dividend => R0_out,      divisor => R1_out, quot => qout_out,
                    remd => remd_out, c => clk);

mux2_out <= data_loc2 when (s3='0' and s2='0') else
            data_loc1 when (s3='0' and s2='1') else
            Iaddr when (s3='1' and s2='0') else
            (others=> '0');

mux1_out <= Data_bus when s1='0' else
            Imem_bus;

Addr <= Data_loc2(6 downto 0) when s0='0' else
          data_loc1(6 downto 0);

mux4_out <= Iaddr_bus when s4='0' else
            adder_out;

mux5_out <= loc_bus when s5 = '0' else
            adder_out;

mux6_out <= loc_bus when s6 = '0' else
            adder_out;

DIM1 : div_imem          port map (addr => Iaddr(3 downto 0), clk => clk, din => Inst_in,
                                   dout => Inst_out, we => I_R_W);

Imem_bus <= Inst_out when I_R_W = '0' else
            (others=>'Z');

Inst_in <= Imem_bus when I_R_W = '1' else
            (others=>'0');

Data_bus <= result when (R_W = '1' and S7 = '0') else
            rem_rslt when (R_W = '1' and S7 = '1') else
            (others=>'Z');

control: process(clk, instr_rdy, bus_gnt, cntrlr_bus, rst, delay, data_loc2,Op)


          variable load_delay, ld_del2, del : boolean;

          begin
                    if rst = '1' then
                              OP <= reset;
                    elsif (clk'event and clk = '1') then
                              if Op = reset then
                              test <= "StateReset";
                              snd_i <= '1'; del := false;
                              fin <= '1'; ld_del2 := false;
                              bus_req <= '0'; I_R_W <= '0';
                              r_w <= '0'; LR0 <= '0';
```

```
            s4 <= '0'; s1 <= '0';
            s2 <= '0'; s3 <= '0'; s0 <= '1';
            s5 <= '0'; s6 <= '0'; s7 <= '0';
            Ci <= '0'; LR2 <= '0'; LR1 <= '0';
            LD_D1 <= '0'; LD_D2 <= '0';
            r2_rst <= '1'; load_delay := false;
            ld_rslt <= '0'; ld_Iaddr<= '0';
            delay <= "00000000000000000001";
            Op <= GetOp;
elsif Op = GetOp then                          --ld data loc 1
        r2_rst <= '0'; LD_D2 <= '0';
        LR2 <= '0'; LR1 <= '0';
        bus_req <= '0';
        ld_rslt <= '0'; ld_Iaddr<= '0';
        if instr_rdy = '1' then
                    loc_bus <= Cntrlr_bus(7 downto 0);
                    LD_D1 <= '1';
                    fin <= '0'; s5 <= '0';
                    Snd_i <= '1';
                    Op <= O1;
            else
                    OP <= GetOp;
            end if;
elsif Op = O1 then
        LD_D1 <= '0';
        r2_rst <= '0';
        LR2 <= '0'; LR1 <= '0';
        bus_req <= '0';
        ld_rslt <= '0';
            if (instr_rdy = '1' or load_delay = true) then
                    if cntrlr_bus(15 downto 8) = StoreDL then --ld dl2
                            loc_bus <= cntrlr_bus(7 downto 0);
                            LD_D2 <= '1'; ld_Iaddr<= '0';
                            fin <= '0'; s6 <= '0';
                            snd_i <='1';
                            Op <= O1;
                    elsif cntrlr_bus(15 downto 8) = GoDiv then --start div ops
                            if (load_delay = false) then
                              Iaddr_bus <= cntrlr_bus(7 downto 0);        --ld instr loc
                              LD_D2 <= '0'; s4 <= '0';
                              Ld_Iaddr <= '1';
                              Snd_I <= '0';
                              load_delay := true;
                              Op <= O1;
                            elsif (load_delay = true) then
                              Ld_Iaddr <= '0';
                              Op <= O2; load_delay := false;
                            end if;
                    end if;
            else
                    Op <= O1;
            end if;
elsif Op = O2 then                             --ld R2 with dl1 offset
            r2_rst <= '0'; LD_D2 <= '0';       --from Imem
        LR1 <= '0'; ld_d1 <= '0';
        bus_req <= '0';
```

```
      ld_rslt <= '0';
          ld_Iaddr <= '0';
          I_R_W <= '0'; LR2 <= '1';
          Op <= O3;
elsif Op = O3 then                              --add offset to dl1 str in dl1
          LD_D2 <= '0';
          -- changes for dbugging
          --LR2 <= '1';
     LR2 <= '0';
            LR1 <= '0';
      bus_req <= '0';
      ld_rslt <= '0'; ld_Iaddr<= '0';
          Ci <= '0'; LR2 <= '0';
          LD_D1 <= '1'; S5 <= '1';
          s2 <= '1'; s3 <= '0';
          Op <= O4; r2_rst <= '1';
elsif Op = O4 then                              --Inc Iaddr
          if (ld_del2 = false) then
            LD_D2 <= '0';
       LR2 <= '0'; LR1 <= '0';
       bus_req <= '0';
       ld_rslt <= '0';
            LD_D1 <= '0'; r2_rst <= '0';
            s2 <= '0'; s3 <= '1'; S4<='1';
            ci <= '1'; ld_Iaddr <= '1';
            Op <= O4; ld_del2 := true;
          elsif (ld_del2 = true) then
            ld_Iaddr <= '0';
            Op <= O5;
            ld_del2 := false;
          end if;
elsif Op = O5 then                              --Check for 2nd dl
          r2_rst <= '0'; LD_D2 <= '0';
      bus_req <= '0'; ld_d1 <= '0';
      ld_rslt <= '0';
          ld_Iaddr <= '0';
          if data_loc2 = "00000000" then        --get divisor from IMEM
                  I_R_W <= '0'; lr0 <= '0'; --put in R1
              S1 <= '1'; lr1 <= '1';
                  Op <= O6;
          else                                  --get data from DMEM
                  I_R_W <= '0'; lr0 <= '0'; --get offset to Dl2
                  lr2 <='1';
                  Op <= O5a; lr1<='0';
          end if;
elsif Op = O5a then                             --add offset to  Dl2
          r2_rst <= '0';
          LR1 <= '0';
      bus_req <= '0'; ld_d1 <= '0';
      ld_rslt <= '0'; ld_Iaddr<= '0';
          lr2 <= '0'; s2 <= '0'; s3 <= '0';
          ci <= '0'; s6 <= '1';
          LD_D2 <= '1';
          Op <= O5b;
elsif Op = O5b then
test <= "State O5b ";
```

```
            r2_rst <= '0';
        LR2 <= '0'; LR1 <= '0';
        ld_d1 <= '0';
        ld_rslt <= '0'; ld_Iaddr<= '0';
            LD_D2 <= '0'; s0 <= '0';
            bus_req <= '1'; R_w <= '0';
            Op <= O5c;  s1 <= '0';
elsif Op = O5c then                             --ld R1 with divisor
test <= "State O5c ";

            r2_rst <= '0'; LD_D2 <= '0';        --from DMEM
        LR2 <= '0'; s1 <= '0';
        ld_d1 <= '0';
        ld_rslt <= '0'; ld_Iaddr<= '0';
            if bus_gnt = '1' then
                        lr1 <= '1';
                        Op <= O6;
            else
                        LR1 <= '0';
                        Op <= O5c;
            end if;
elsif Op = O6 then                              --ld R0 with dividend
test <= "State O6  ";

            r2_rst <= '0'; LD_D2 <= '0';
        LR2 <= '0'; LR1 <= '0';
        ld_d1 <= '0';
        ld_rslt <= '0'; ld_Iaddr<= '0';
            s0<= '1'; R_w <= '0';
            bus_req <= '1';
            Op <= O7;
elsif Op = O7 then
            r2_rst <= '0'; LD_D2 <= '0';
        LR2 <= '0'; LR1 <= '0';
        ld_d1 <= '0';
        ld_rslt <= '0'; ld_Iaddr<= '0';
            if bus_gnt = '1' then
                        lr0 <= '1';
                        Op <= O8;
            else
                        lr0 <= '0';
                        OP <= O7;
            end if;
elsif Op = O8 then                              --wait for result 20 CC's
            LD_D2 <= '0';
        LR2 <= '0'; LR1 <= '0';
        bus_req <= '0'; ld_d1 <= '0';
        ld_Iaddr<= '0';lr0 <= '0';
            bus_req <= '0';
            r2_Rst <= '1';
            if delay = "10000000000000000000" then
                        Ld_rslt <= '1';
                        Op <= O9;
            else
                        delay <= delay(18 downto 0)&'0';
```

```
                                ld_rslt <= '0';
                                Op <= O8;
                        end if;
                elsif Op = O9 then
                test <= "State O9  ";
                        r2_rst <= '0';
                    LR2 <= '0'; LR1 <= '0';
                    ld_rslt <= '0'; ld_Iaddr<= '0';
                        r2_Rst <= '0'; R_W <= '1';
                        if data_loc2 = "00000000" then      --use DL1 for store
                                S0<='1';
                                ld_d2 <= '0';
                        else                                --use DL2 for store
                                S0 <= '0';
                                ld_d1 <= '0';
                        end if;
                        Bus_req <= '1';
                        Op <= O10;
                elsif Op = O10 then
                test <= "State O10 ";
                        r2_rst <= '0'; LD_D2 <= '0';
                    LR2 <= '0'; LR1 <= '0';
                    ld_d1 <= '0'; S7 <= '0';
                    ld_rslt <= '0'; ld_Iaddr<= '0';
                        if bus_gnt = '1' then               --Store Quotient in mem
                                fin <= '1';

                                                    bus_req <= '0';
                        Op <= reset;
                    else
                        Op <= O10;
                    end if;
                end if;
            end if;

end process;

reg2 : process (clk, Imem_bus, R2_rst, Lr2)
        begin
                if clk'event and clk='1' then
                        if R2_rst = '1' then
                                R2_out <= (others=>'0');
                        elsif lr2 = '1' then
                                R2_out <= Imem_bus(7 downto 0);
                        else
                                R2_out <= R2_out;
                        end if;
                end if;
end process;

reg_dl1: process (clk, mux5_out, rst, LD_D1)
        begin
                if rst ='1' then
                        data_loc1 <= (others=>'0');
                elsif clk'event and clk='1' then
                        if LD_D1 = '1' then
```

```
                        data_loc1 <= mux5_out;
                    else
                        data_loc1 <= data_loc1;
                    end if;
                end if;
end process;

reg_dl2: process (clk, mux6_out, rst, LD_D2)
        begin
                if rst ='1' then
                        data_loc2 <= (others=>'0');
                elsif clk'event and clk='1' then
                        if LD_D2 = '1' then
                                data_loc2 <= mux6_out;
                        else
                                data_loc2 <= data_loc2;
                        end if;
                end if;
end process;

reg_R0: process (clk, data_bus, rst, lR0)
        begin
                if rst ='1' then
                        R0_out <= (others=>'0');
                elsif clk'event and clk='1' then
                        if lR0 = '1' then
                                R0_out <= data_bus;
                        else
                                R0_out <= R0_out;
                        end if;
                end if;
end process;

reg_R1: process (clk, mux1_out, rst, lR1)
        begin
                if rst ='1' then
                        R1_out <= (others=>'0');
                elsif clk'event and clk='1' then
                        if lR1 = '1' then
                                R1_out <= mux1_out;
                        else
                                R1_out <= R1_out;
                        end if;
                end if;
end process;

reg_Iaddr: process (clk, mux4_out, rst, ld_Iaddr)
        begin
                if rst ='1' then
                        Iaddr <= (others=>'0');
                elsif clk'event and clk='1' then
                        if ld_Iaddr = '1' then
                                Iaddr <= mux4_out;
                        else
                                Iaddr <= Iaddr;
                        end if;
```

```
                end if;
end process;

reg_Rslt: process (clk, qout_out, remd_out, rst, ld_Rslt)
        begin
                if rst ='1' then
                        result <= (others=>'0');
                        rem_rslt <= (others=>'0');
                elsif clk'event and clk='1' then
                        if ld_Rslt = '1' then
                                result <= qout_out;
                                rem_rslt <= remd_out;
                        else
                                result <= result;
                                rem_rslt <= rem_rslt;
                        end if;
                end if;

end process;

end architecture;
```

Module Name : addsub8_synthable.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
--use ieee.std_logic_arith.all;

ENTITY add_subber8 IS

  PORT(
    A: IN std_logic_vector(7 DOWNTO 0);
    B: IN std_logic_vector(7 DOWNTO 0);
    C_IN: IN std_logic;
    C_OUT: OUT std_logic;
    ADD_SUB: IN std_logic;
    Q_OUT: OUT std_logic_vector(7 DOWNTO 0));
END add_subber8;

ARCHITECTURE sim OF add_subber8 IS
    SIGNAL S: std_logic_vector(7 DOWNTO 0);
    SIGNAL S1: std_logic_vector(7 DOWNTO 0);
    SIGNAL AA: std_logic_vector(7 DOWNTO 0);
    SIGNAL C: std_logic_vector(8 DOWNTO 0);
    SIGNAL T: std_logic_vector(7 DOWNTO 0);

BEGIN
Q_OUT<=S;
PROCESS(A,B,C_IN,ADD_SUB,C,T,AA,S1,S)
begin
if ADD_SUB='1' THEN
                C(0)<= C_IN;
                for i in 0 to 7 loop
                        S(i) <= A(i) xor B(i) xor C(i);
```

```
                    C(i+1)<= (A(i) and B(i)) or (A(i) and C(i)) or (B(i) and C(i));
                end loop;
                C_OUT <= C(8);
else
                T<=NOT (B+C_IN);
                AA<=A+1;

                C(0) <= C_in;
                for i in 0 to 7 loop
                    S1(i) <= AA(i) xor T(i) xor C(i);
                    C(i+1)<= (AA(i) and T(i)) or (AA(i) and C(i)) or (T(i) and C(i));
                end loop;
                --C_OUT <= NOT C(8);
                C_OUT <=  C(8);
                if C(8) = '0'
                 then
                --if s1(7) = '1' and A(7) = '0' then
                    s <= (not s1) +1;
                else s <= s1;
                end if;
end if;
end process;
END sim;
```

Module Name: div1.xco (Xilinx IP Core)

```
------------------------------------------------------------------------------
--  This file is owned and controlled by Xilinx and must be used          --
--  solely for design, simulation, implementation and creation of         --
--  design files limited to Xilinx devices or technologies. Use           --
--  with non-Xilinx devices or technologies is expressly prohibited       --
--  and immediately terminates your license.                              --
--                                                                        --
--  XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         --
--  SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR               --
--  XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION       --
--  AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION           --
--  OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS             --
--  IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,               --
--  AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      --
--  FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              --
--  WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               --
--  IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        --
--  REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       --
--  INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       --
--  FOR A PARTICULAR PURPOSE.                                             --
--                                                                        --
--  Xilinx products are not intended for use in life support              --
--  appliances, devices, or systems. Use in such applications are         --
--  expressly prohibited.                                                 --
--                                                                        --
--  (c) Copyright 1995-2003 Xilinx, Inc.                                  --
--  All rights reserved.                                                  --
------------------------------------------------------------------------------
-- You must compile the wrapper file div1.vhd when simulating
-- the core, div1. When compiling the wrapper file, be sure to
```

```
-- reference the XilinxCoreLib VHDL simulation library. For detailed
-- instructions, please refer to the "CORE Generator Guide".

-- The synopsys directives "translate_off/translate_on" specified
-- below are supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity
-- synthesis tools. Ensure they are correct for your synthesis tool(s).

-- synopsys translate_off
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

Library XilinxCoreLib;
ENTITY div1 IS
        port (
        dividend: IN std_logic_VECTOR(15 downto 0);
        divisor: IN std_logic_VECTOR(15 downto 0);
        quot: OUT std_logic_VECTOR(15 downto 0);
        remd: OUT std_logic_VECTOR(15 downto 0);
        c: IN std_logic);
END div1;

ARCHITECTURE div1_a OF div1 IS

component wrapped_div1
        port (
        dividend: IN std_logic_VECTOR(15 downto 0);
        divisor: IN std_logic_VECTOR(15 downto 0);
        quot: OUT std_logic_VECTOR(15 downto 0);
        remd: OUT std_logic_VECTOR(15 downto 0);
        c: IN std_logic);
end component;

-- Configuration specification
        for all : wrapped_div1 use entity XilinxCoreLib.dividervht(behavioral)
                generic map(
                        dividend_width => 16,
                        signed_b => 0,
                        fractional_b => 0,
                        divisor_width => 16,
                        fractional_width => 16,
                        divclk_sel => 1);
BEGIN

U0 : wrapped_div1
                port map (
                        dividend => dividend,
                        divisor => divisor,
                        quot => quot,
                        remd => remd,
                        c => c);
END div1_a;

-- synopsys translate_on
```

Module Name : div_imem.xco (Xilinx IP Core)

```
--------------------------------------------------------------------------------
--  This file is owned and controlled by Xilinx and must be used          --
--  solely for design, simulation, implementation and creation of         --
--  design files limited to Xilinx devices or technologies. Use           --
--  with non-Xilinx devices or technologies is expressly prohibited       --
--  and immediately terminates your license.                              --
--                                                                        --
--  XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         --
--  SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR               --
--  XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION       --
--  AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION           --
--  OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS             --
--  IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,               --
--  AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      --
--  FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              --
--  WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               --
--  IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        --
--  REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       --
--  INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       --
--  FOR A PARTICULAR PURPOSE.                                             --
--                                                                        --
--  Xilinx products are not intended for use in life support              --
--  appliances, devices, or systems. Use in such applications are         --
--  expressly prohibited.                                                 --
--                                                                        --
--  (c) Copyright 1995-2002 Xilinx, Inc.                                  --
--  All rights reserved.                                                  --
--------------------------------------------------------------------------------
-- You must compile the wrapper file div_imem.vhd when simulating
-- the core, div_imem. When compiling the wrapper file, be sure to
-- reference the XilinxCoreLib VHDL simulation library. For detailed
-- instructions, please refer to the "Coregen Users Guide".

-- The synopsys directives "translate_off/translate_on" specified
-- below are supported by XST, FPGA Express, Exemplar and Synplicity
-- synthesis tools. Ensure they are correct for your synthesis tool(s).

-- synopsys translate_off
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

Library XilinxCoreLib;
ENTITY div_imem IS
        port (
        addr: IN std_logic_VECTOR(3 downto 0);
        clk: IN std_logic;
        din: IN std_logic_VECTOR(15 downto 0);
        dout: OUT std_logic_VECTOR(15 downto 0);
        we: IN std_logic);
END div_imem;

ARCHITECTURE div_imem_a OF div_imem IS

component wrapped_div_imem
        port (
        addr: IN std_logic_VECTOR(3 downto 0);
```

```
        clk: IN std_logic;
        din: IN std_logic_VECTOR(15 downto 0);
        dout: OUT std_logic_VECTOR(15 downto 0);
        we: IN std_logic);
end component;

-- Configuration specification
        for all : wrapped_div_imem use entity XilinxCoreLib.blkmemsp_v5_0(behavioral)
                generic map(
                        c_sinit_value => "0",
                        c_reg_inputs => 0,
                        c_yclk_is_rising => 1,
                        c_has_en => 0,
                        c_ysinit_is_high => 1,
                        c_ywe_is_high => 1,
                        c_ytop_addr => "1024",
                        c_yprimitive_type => "4kx1",
                        c_yhierarchy => "hierarchy1",
                        c_has_rdy => 0,
                        c_has_limit_data_pitch => 0,
                        c_write_mode => 0,
                        c_width => 16,
                        c_yuse_single_primitive => 0,
                        c_has_nd => 0,
                        c_enable_rlocs => 0,
                        c_has_we => 1,
                        c_has_rfd => 0,
                        c_has_din => 1,
                        c_ybottom_addr => "0",
                        c_pipe_stages => 0,
                        c_yen_is_high => 1,
                        c_depth => 16,
                        c_has_default_data => 0,
                        c_limit_data_pitch => 8,
                        c_has_sinit => 0,
                        c_mem_init_file => "div_imem.mif",
                        c_default_data => "0",
                        c_ymake_bmm => 0,
                        c_addr_width => 4);
BEGIN

U0 : wrapped_div_imem
                port map (
                        addr => addr,
                        clk => clk,
                        din => din,
                        dout => dout,
                        we => we);
END div_imem_a;

-- synopsys translate_on
```

Module Name : ic_hdca_gate.vhd

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity gate_ic_a is
   Port ( clk: in std_logic ;
          rst: in std_logic ;
          ctrl: in std_logic_vector(3 downto 0) ;
          qdep: in std_logic_vector(19 downto 0) ;
          addr_bus: in std_logic_vector(27 downto 0) ;
          data_in0,data_in1,data_in2,data_in3 : in std_logic_vector(15 downto 0) ;
          rw: in std_logic_vector(3 downto 0) ;
          flag: out std_logic_vector(3 downto 0) ;
          data_out0,data_out1,data_out2,data_out3: out std_logic_vector(15 downto 0)
                    -- f_s_out0,f_s_out1,f_s_out2,f_s_out3 : out std_logic_vector(3 downto 0);
            --            dco_out0,dco_out1,dco_out2,dco_out3 : out std_logic_vector(3 downto 0)
          );
end gate_ic_a;

architecture gate_level of gate_ic_a is


-- component listing

component Dec_ic_a is
    port(dec_out : out std_logic_vector( 3 downto 0);
         ctrl_dec : in std_logic;
                    addr_blk : in std_logic_vector(1 downto 0)
                    );
end component;

component prl_behav is
   Port (clk,rst : in std_logic;
          d0,d1,d2,d3 : in std_logic;
          q0,q1,q2,q3 : in std_logic_vector( 4 downto 0);
                     sub_flg : out std_logic_vector ( 3 downto 0)
          );
end component;


-- memory array ----
type mem_array is array ( 127 downto 0) of std_logic_vector(15 downto 0);

--signal list
signal d_sig0,d_sig1,d_sig2,d_sig3 : std_logic_vector(3 downto 0);
signal flg_sig0,flg_sig1,flg_sig2,flg_sig3: std_logic_vector( 3 downto 0);
signal memory : mem_array;
signal flag_decide0,flag_decide1,flag_decide2,flag_decide3: std_logic_vector(3 downto 0);
signal flag_wire: std_logic_vector(3 downto 0);
-- make qdep as signal
--signal qd00,qd01,qd02,qd03 : std_logic_vector( 3 downto 0);
```

```
-- signal list end here


begin


-- signals to ports if any

--f_s_out0 <= flg_sig0;
--f_s_out1 <= flg_sig1;
--f_s_out2 <= flg_sig2;
--f_s_out3 <= flg_sig3;

--dco_out0 <= d_sig0;
--dco_out1 <= d_sig1;
--dco_out2 <= d_sig2;
--dco_out3 <= d_sig3;
flag <= flag_wire;


flag_decide0<= flg_sig0(0)&flg_sig1(0)&flg_sig2(0)&flg_sig3(0);
flag_decide1<= flg_sig0(1)&flg_sig1(1)&flg_sig2(1)&flg_sig3(1);
flag_decide2<= flg_sig0(2)&flg_sig1(2)&flg_sig2(2)&flg_sig3(2);
flag_decide3<= flg_sig0(3)&flg_sig1(3)&flg_sig2(3)&flg_sig3(3);

-- port mapping
-- decoder instantiated 4 times

DEC0 : Dec_ic_a port map(dec_out => d_sig0,
                ctrl_dec => ctrl(0),
                        addr_blk => addr_bus(6 downto 5)
                  );

DEC1 : Dec_ic_a port map(dec_out => d_sig1,
                ctrl_dec => ctrl(1),
                        addr_blk => addr_bus(13 downto 12)
                  );

DEC2 : Dec_ic_a port map(dec_out => d_sig2,
                ctrl_dec => ctrl(2),
                        addr_blk => addr_bus(20 downto 19)
                  );

DEC3 : Dec_ic_a port map(dec_out => d_sig3,
                ctrl_dec => ctrl(3),
                           addr_blk => addr_bus(27 downto 26)
                  );
-- decoder instantiation ends ----

-- pr logic instantiation 4 times ----

PRL_LOGIC0 : prl_behav port map( clk => clk,
                    rst => rst,
```

```
                d0  => d_sig0(0),
                                                d1  => d_sig1(0),
                                                d2  => d_sig2(0),
                                                d3  => d_sig3(0),
                q0  => qdep(4 downto 0),
                                                q1  => qdep(9 downto 5),
                                                q2  => qdep(14 downto 10),
                                                q3  => qdep(19 downto 15),
                        sub_flg   => flg_sig0
                                                );

PRL_LOGIC1 : prl_behav port map( clk => clk,
                rst => rst,
                d0  => d_sig0(1),
                                                d1  => d_sig1(1),
                                                d2  => d_sig2(1),
                                                d3  => d_sig3(1),
                q0  => qdep( 4 downto 0),
                                                q1  => qdep(9 downto 5),
                                                q2  => qdep(14 downto 10),
                                                q3  => qdep(19 downto 15),
                                sub_flg   => flg_sig1

                                                );

PRL_LOGIC2 : prl_behav port map( clk => clk,
                rst => rst,
                d0  => d_sig0(2),
                                                d1  => d_sig1(2),
                                                d2  => d_sig2(2),
                                                d3  => d_sig3(2),
                q0  => qdep(4 downto 0),
                                                q1  => qdep(9 downto 5),
                                                q2  => qdep(14 downto 10),
                                                q3  => qdep(19 downto 15),
                        sub_flg   => flg_sig2

                                                );

PRL_LOGIC3 : prl_behav port map( clk => clk,
                rst => rst,
                d0  => d_sig0(3),
                                                d1  => d_sig1(3),
                                                d2  => d_sig2(3),
                                                d3  => d_sig3(3),
                q0  => qdep(4 downto 0),
                                                q1  => qdep(9 downto 5),
                                                q2  => qdep(14 downto 10),
                                                q3  => qdep(19 downto 15),
                        sub_flg   => flg_sig3

                                                );


-- extra logic to be added since all the prl_blks give output flag value ...
```

```
-- there would be conflict as to what the final value is
-- try and include it in a process ... so that flag value changes in accordance with the
-- clk ..
flag_assign : process (clk,rst,flag_decide0,flag_decide1,flag_decide2,flag_decide3)

begin

if(rst ='1') then
flag_wire <= "0000";

elsif (clk'event and clk ='0') then
 case flag_decide0 is
            when "0000" => flag_wire(0) <= '0';
            when others => flag_wire(0) <= '1';
 end case;

case flag_decide1 is
            when "0000" => flag_wire(1) <= '0';
            when others => flag_wire(1) <= '1';
 end case;

 case flag_decide2 is
            when "0000" => flag_wire(2) <= '0';
            when others => flag_wire(2) <= '1';
 end case;

 case flag_decide3 is
            when "0000" => flag_wire(3) <= '0';
            when others => flag_wire(3) <= '1';
 end case;



end if;

end process flag_assign;


-- end of extra logic added ----------


-- write about r_w logic,shall come along with flag thing ----

data_transfer : process(rst,data_in0,data_in1,data_in2,data_in3,flag_wire,rw,clk)

begin

if (rst ='1') then
--flag <= "0000";
data_out0 <=x"0000";
data_out1 <=x"0000";
data_out2 <=x"0000";
data_out3 <=x"0000";
-- making the memory array all zeroes
MEM : for i in 0 to 127 loop
memory(i)<=x"0000";
```

```
end loop MEM;
else

if (clk'event and clk ='1') then

if (flag_wire(0) ='1')then
if (rw(0) ='1') then
        memory(conv_integer(addr_bus( 6 downto 0))) <= data_in0;
elsif (rw(0)='0')then
        data_out0 <= memory(conv_integer(addr_bus( 6 downto 0)));
end if;
end if;

if (flag_wire(1) ='1') then
if (rw(1) ='1') then
memory(conv_integer(addr_bus( 13 downto 7))) <= data_in1;
--data_out1 <=(others =>'Z'); --commented later
else
data_out1 <= memory(conv_integer(addr_bus( 13 downto 7)));
end if;
end if;

if (flag_wire(2) ='1') then
if (rw(2) ='1') then
memory(conv_integer(addr_bus( 20 downto 14))) <= data_in2;
--data_out2 <=(others =>'Z');
else
data_out2 <= memory(conv_integer(addr_bus(20 downto 14)));
end if;
end if;

if (flag_wire(3) ='1') then
if (rw(3) ='1') then
memory(conv_integer(addr_bus( 27 downto 21))) <= data_in3;
--data_out3 <=(others =>'Z');
else
data_out3 <= memory(conv_integer(addr_bus(27 downto 21)));
end if;
end if;

end if;
end if;

end process data_transfer;

end gate_level;
```

Module Name : dec_ic_a.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity dec_ic_a is
   Port (dec_out : out std_logic_vector( 3 downto 0);
        ctrl_dec : in std_logic;
                   addr_blk : in std_logic_vector(1 downto 0)
         );
end dec_ic_a;

architecture Behavioral of dec_ic_a is

signal ctrl_bar,addr1_bar,addr0_bar : std_logic;

begin
ctrl_bar <= not ctrl_dec;
addr1_bar <= not addr_blk(1);
addr0_bar <= not addr_blk(0);
dec_out(0)<= ctrl_dec and addr1_bar and addr0_bar;
dec_out(1)<= ctrl_dec and addr1_bar and addr_blk(0);
dec_out(2)<= ctrl_dec and addr_blk(1) and addr0_bar;
dec_out(3)<= ctrl_dec and addr_blk(1) and addr_blk(0);

end Behavioral;
```

Module Name : prl_behav.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity prl_behav is
   Port (clk,rst : in std_logic;
        d0,d1,d2,d3 : in std_logic;
        q0,q1,q2,q3 : in std_logic_vector( 4 downto 0);
                   sub_flg : out std_logic_vector ( 3 downto 0)
        );
end prl_behav;

architecture Behavioral of prl_behav is

-- signal listing -----
signal d3d2d1d0 :std_logic_vector(3 downto 0);

--- end of signal list----

begin
```

```
-- process for the selection of proper PE ---

sel : process ( d0,d1,d2,d3,clk,rst)

variable max : std_logic_vector(4 downto 0);

begin

if (rst ='1') then
   sub_flg <= "0000";
else
 if (clk'event and clk='0') then
   d3d2d1d0 <= d3&d2&d1&d0;

  case d3d2d1d0 is
   when "0001" => sub_flg <= "0001" ;
   when "0010" => sub_flg <= "0010";
   when "0100" => sub_flg <= "0100";
   when "1000" => sub_flg <= "1000";
   when "0011" =>
       max:= q0;
              if((max < q1)and (max = q1)) then
              max:= q1;
              sub_flg <="0010";
              else
              sub_flg <="0001";
              end if;

   when "0111" =>
       max:= q0;
              if(max<=q1) then
              max := q1;
                if(max<=q2) then
                       max := q2;
                       sub_flg <="0100";
                       else
                       sub_flg <="0010";
                       end if;
              else
              sub_flg <="0001";
              end if;

   when "0110" =>
       max :=q1;
              if(max<=q2) then
              max:= q2;
              sub_flg <="0100";
              else
              sub_flg <="0010";
              end if;

   when "0101" =>
       max :=q0;
              if(max<=q2)then
              max:=q2;
```

```
            sub_flg <="0100";
            else
            sub_flg <="0001";
            end if;

when "1111" =>
    max :=q0;
            if(max<=q1) then
            max:=q1;
              if(max<=q2) then
                    max:=q2;
                      if(max<=q3) then
                      max:=q3;
                      sub_flg<="1000";
                      else
                      sub_flg <="0100";
                      end if;
                    else
                    sub_flg<="0010";
                    end if;
            else
            sub_flg <="0001";
            end if;

when "1110" =>
    max :=q1;
            if(max<=q2)then
            max:=q2;
              if(max<=q3) then
                    max:=q3;
                    sub_flg <="1000";
                    else
                    sub_flg <="0100";
                    end if;
            else
            sub_flg <="0010";
            end if;

when "1010" =>
    max :=q1;
            if(max<=q3) then
            max:=q3;
            sub_flg <="1000";
            else
            sub_flg <="0010";
            end if;

when "1001"=>
    max:=q0;
            if(max<=q3)then
            max:=q3;
            sub_flg<="1000";
            else
            sub_flg<="0001";
            end if;
```

```
   when "1101" =>
       max :=q0;
              if(max<=q2)then
              max:=q2;
                if(max<=q3) then
                       max:=q3;
                       sub_flg<="1000";
                       else
                       sub_flg<="0100";
                       end if;
              else
              sub_flg<="0001";
              end if;

   when "1100" =>
       max :=q2;
              if(max<=q3) then
              max:=q3;
              sub_flg <="1000";
              else
              sub_flg <="0100";
              end if;

   when "1011" =>
       max :=q0;
              if(max<=q1)then
              max:=q1;
                if(max<=q3)then
                       max:=q3;
                       sub_flg <="1000";
                       else
                       sub_flg<="0010";
                       end if;
              else
              sub_flg <="0001";
              end if;

   when others => sub_flg<="0000";

 end case;
end if ;
end if;

end process;

end Behavioral;
```

Module Name : multpe.vhd

```
----------------------------------------------------
-- Multiplier PE
-- Version 1.00
-- Coded by Kanchan,Sridhar
----------------------------------------------------
 -- synopsys translate_off
```

```
 Library XilinxCoreLib;
 -- synopsys translate_on

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;


entity multpe is
   Port ( mcntl_bus : in std_logic_vector(15 downto 0);
                Snd_I : out std_logic;
                clk : in std_logic;
                rst : in std_logic;
                Instr_rdy : in std_logic;
                Fin : out std_logic;
                mdata_bus : inout std_logic_vector(15 downto 0);
                bus_req : out std_logic;
                bus_gnt : in std_logic;
                multaddr : out std_logic_vector(7 downto 0);--Output address to shared dmem
                --r_w : buffer std_logic;
                        r_w : inout std_logic;
                        cbusout_dbug : out std_logic_vector(7 downto 0);
                        Iaddr_bus_dbug : out std_logic_vector(7 downto 0);
                        --Iaddr_dbug :  out std_logic_vector(7 downto 0);
                        R2out_dbug : out std_logic_vector( 7 downto 0);
                        Imem_bus_dbug : out std_logic_vector(15 downto 0 );

                        mux3out_dbg:out std_logic_vector(7 downto 0);
                        ms3dbg:out std_logic_vector(1 downto 0);
                        ms1dbg : out std_logic;
                        ms2dbg : out std_logic;
                        adderout_dbug : out std_logic_vector(7 downto 0);
                        ms4dbg : out std_logic;
                        lmd_dbg,lmr_dbg : out std_logic;
                        ndout : out std_logic;
                        multout_fin : out std_logic_vector( 15 downto 0);
                        tomultr_dbg:out std_logic_vector(7 downto 0);
                        tomultd_dbg:out std_logic_vector(7 downto 0)

                        );
end multpe;

architecture Behavioral of multpe is

component mult is
   Port ( a : in std_logic_vector(7 downto 0);
          b : in std_logic_vector(7 downto 0);
          q : out std_logic_vector(15 downto 0);
                        clk:in std_logic;
          newdata : in std_logic);
end component;
------------------------------------------------------------------------------
--     This file is owned and controlled by Xilinx and must be used          --
--     solely for design, simulation, implementation and creation of         --
```

```
--     design files limited to Xilinx devices or technologies. Use           --
--     with non-Xilinx devices or technologies is expressly prohibited       --
--     and immediately terminates your license.                              --
--                                                                           --
--     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         --
--     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR               --
--     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION       --
--     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION           --
--     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS             --
--     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,               --
--     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      --
--     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              --
--     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               --
--     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        --
--     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       --
--     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       --
--     FOR A PARTICULAR PURPOSE.                                             --
--                                                                           --
--     Xilinx products are not intended for use in life support              --
--     appliances, devices, or systems. Use in such applications are         --
--     expressly prohibited.                                                 --
--                                                                           --
--     (c) Copyright 1995-2002 Xilinx, Inc.                                  --
--     All rights reserved.                                                  --
-------------------------------------------------------------------------------

component mult_imem IS
            port (
            addr: IN std_logic_VECTOR(2 downto 0);
            clk: IN std_logic;
            din: IN std_logic_VECTOR(15 downto 0);
            dout: OUT std_logic_VECTOR(15 downto 0);
            we: IN std_logic);
end component;

component add_subber8 IS

  PORT(
    A: IN std_logic_vector(7 DOWNTO 0);
    B: IN std_logic_vector(7 DOWNTO 0);
    C_IN: IN std_logic;
    C_OUT: OUT std_logic;
    ADD_SUB: IN std_logic;
    Q_OUT: OUT std_logic_vector(7 DOWNTO 0));
END component;


--All control signals for the various components used


--Control signals for the multiplexors used in the design
signal ms0,ms1,ms2,ms4,ms5:std_logic;
signal ms3:std_logic_vector(1 downto 0);
--control signals for datalocations,reg R2
signal mldl1,mldl2,mldr2,lmr,lmd,lmar:std_logic;
signal mlresult:std_logic;
```

```
--output of data locations 1 and 2
signal mdloc1out,mdloc2out:std_logic_vector(7 downto 0);
signal r2out:std_logic_vector(7 downto 0);
signal mux3out,mux5out,mux0out,mux1out,adderout:std_logic_vector(7 downto 0);
--output from controller to data locations
signal cbusout:std_logic_vector(7 downto 0);
signal mux4out:std_logic_vector(15 downto 0);
-- signal added to supplement the mdatabus port ...
signal mdata_sig  : std_logic_vector(15 downto 0);

--outputs of multiplier and multiplicand registers


signal mrout,mdout:std_logic_vector(7 downto 0);
--output from pipelined multiplier and output from result register
signal multout,multrslt:std_logic_vector(15 downto 0);



--Core instruction memory signals
signal inst_in,inst_out:Std_logic_vector(15 downto 0);
signal imem_bus:std_logic_vector(15 downto 0);


--Adder signal that is not being used
signal ci:std_logic;
--signal iaddr:std_logic_vector(7 downto 0);
signal iaddr_bus:std_logic_vector(7 downto 0);
signal from_cntl : std_logic_vector(7 downto 0);
signal rwmem:std_logic;
type OP_state is (reset,Getop,Op1,Op2,Op3,Op4,Op5,Op6,Op7,Op8,Op9,Op10,Op11,Op12,Op13,Op14);
signal OP : OP_state;
signal delay : std_logic_vector(1 downto 0); --Need a 2 CC delay for multiplication to get over
signal r2_rst : std_logic;
signal ndsig:std_logic;

--Start the multiplication operation
constant startmult : std_logic_vector(7 downto 0) := "11111111";
constant storemultdl : std_logic_vector(7 downto 0) := "10001000";

--Alias list starts here



alias toimem:std_logic_vector(2 downto 0) is iaddr_bus( 2 downto 0);
alias tomultr:std_logic_vector(7 downto 0) is mdata_bus(7 downto 0);
alias tomultd:std_logic_vector(7 downto 0) is mux4out(7 downto 0);
alias to_r2:std_logic_vector(7 downto 0) is imem_bus(7 downto 0);

begin
tomultr_dbg<=tomultr;
tomultd_dbg<=tomultd;
ms3dbg<=ms3;
ms2dbg<= ms2;
ms1dbg<= ms1;
```

```
ms4dbg<= ms4;
lmd_dbg <= lmd;
lmr_dbg<= lmr;
mux3out_dbg<=mux3out;
ndout<= ndsig;
adderout_dbug <= adderout;
multout_fin<= multrslt;
-- added for debugging
cbusout_dbug <= cbusout;
--Iaddr_dbug <= Iaddr;
iaddr_bus_dbug<=Iaddr_bus;
R2out_dbug <= r2out;
Imem_bus_dbug <= imem_bus;
--Port maps and when else statements come here outside the process

addermap: add_subber8
                                        port
map(a=>r2out,b=>mux3out,c_in=>ci,c_out=>open,add_sub=>'1',q_out=>adderout);



multmap: mult port map(a=>mrout,b=>mdout,q=>multout,clk=>clk,newdata=>ndsig);


multimemmap:mult_imem port map(addr=>toimem,clk=>clk,din=>inst_in,dout=>inst_out,we=>rwmem);


--End port maps for components

--Mux functionality starts here
imem_bus <=inst_out when rwmem = '0' else
            (others=>'Z');

mdata_bus<=multrslt when mlresult='1' else
            (others=>'Z');
--tomultr <= mdata_bus( 7 downto 0) when lmr='1' else
  --       ( others=>'z');



mux0out<= cbusout when ms0='0' else
        adderout when ms0='1'else
                    (others=>'Z');

mux1out<= cbusout when ms1='0' else
        adderout when ms1='1'else
                    (others=>'Z');

--Mux 2 output
multaddr<= mdloc1out when ms2='0' else
                     mdloc2out when ms2='1' else
                     (others=>'Z');

mux3out<= mdloc1out when ms3="00" else
                    mdloc2out when  ms3="01" else
                    iaddr_bus when ms3="10" else
```

```
                    (others=>'Z');

mux4out<= mdata_bus when ms4='0' else
        imem_bus when ms4='1' else
                    (others=>'Z');
mux5out <= from_cntl when ms5='0' else
                     adderout when ms5='1' else
                     (others=>'Z');

-- The main process that controls the functioning of the multiplier
control:process(clk,rst,instr_rdy, bus_gnt, mcntl_bus,mdloc2out,Op,r2_rst,ndsig,delay)
variable load_delay, ld_del2, del : boolean;
--Start editing here
begin
                    if rst = '1' then
                              OP <= reset;
                    elsif (clk'event and clk = '1') then
                              if Op = reset then
                              snd_i <= '1';
                              del := false;
                              fin <= '1';
                              ld_del2 := false;
                              bus_req <= '0';
                              rwmem <= '0';
                              r_w <= '0';
                              lmr <= '0';
                              ms4 <= '0';
                              ms1 <= '0';
                              ms3 <= "00";
                              ms0 <= '1';
                              ms2<='0';
                              ms5 <= '0';
                              Ci <= '0';
                              mldr2<= '0';
                              lmd<= '0';
                              mldl1<= '0';
                              mldl2 <= '0';
                              load_delay := false;
                              mlresult <= '0';
                              lmar<= '0';
                              r2_rst <= '1'; -- active high resets R2
                              delay <= "01";
                              ndsig<='0';
                              assert  not(Op=reset) report "----------Reset State----------" severity
Note;

                              Op <= GetOp;



                    elsif Op = GetOp then                    --ld data loc 1
                        mldl2 <= '0';
                        mldr2 <= '0';
                              lmd <= '0';
                        bus_req <= '0';
                        mlresult <= '0';
```

```
                    lmar<= '0';
                    r2_rst <= '0';
                if instr_rdy = '1' then
                        cbusout <= mcntl_bus(7 downto 0);
                        mldl1 <= '1';
                        fin <= '0';
                        ms0 <= '0';
                        Snd_i <= '1';
                        Op <= Op1;
                            assert  not(Op=GetOp) report "--------------------Get Op----------
-------" severity Note;
                    else
                        OP <= GetOp;
                    end if;



            elsif Op = Op1 then
                mldl1 <= '0';
                r2_rst <= '0';
                mldr2 <= '0'; lmd <= '0';
                bus_req <= '0';
                mlresult <= '0';
                    if (instr_rdy = '1' or load_delay = true) then
                        if mcntl_bus(15 downto 8) = storemultdl then --ld dl2
                            assert  not(Op=Op1) report "--------------------Op1:inside
storemultdl-----------------" severity Note;
                            cbusout <= mcntl_bus(7 downto 0);
                            mldl2 <= '1';
                            lmar<= '0';
                            fin <= '0';
                            ms1 <= '0';
                            snd_i <='1';
                            Op <= Op1;
                        elsif mcntl_bus(15 downto 8) = startMult then --start multiplication

                            if (load_delay = false) then
        assert  not(Op=Op1) report "--------------------Op1:inside startMult-----------------" severity Note;

                              from_cntl <= mcntl_bus(7 downto 0);        --ld instr loc
                              mldl2 <= '0';
                              ms5 <= '0';
                              lmar <= '1';
                              Snd_I <= '0';
                              load_delay := true;
                              Op <= Op1;
                            elsif (load_delay = true) then
                              lmar <= '0';
                              Op <= Op2;
                              load_delay := false;
                            end if;
                        end if;
                    else
                        Op <= Op1;
                    end if;
```

```
                elsif Op = Op2 then                             --ld R2 with dl1 offset
assert  not(Op=Op2) report "-------------------Op2:inside Op2-----------------" severity Note;

                        mldl2 <= '0';     --from Imem
                lmd <= '0';
                        mldl1 <= '0';
                bus_req <= '0';
                mlresult <= '0';
                        lmar <= '0';
                        rwmem <= '0';
                        mldr2 <= '1';
                        r2_rst <= '0';
                        Op <= Op3;



                elsif Op = Op3 then                             --add offset to dl1 str in dl1
assert  not(Op=Op3) report "-------------------Op3:add ofset to dl1-----------------" severity Note;
                        mldl2 <= '0';
                        -- changes for dbugging
                        --mldr2 <= '1';
                mldr2 <= '0';
                        lmd <= '0';
                bus_req <= '0';
                mlresult <= '0';
                        lmar<= '0';
                        Ci <= '0';
                        mldr2 <= '0';
                        mldl1 <= '1';
                        ms0 <= '1';
                        ms3(0) <= '0';
                        ms3(1) <= '0';
                        r2_rst <= '0';
                        Op <= Op4;


                elsif Op = Op4 then                             --Inc Iaddr
                        if (ld_del2 = false) then
assert  not(Op=Op4) report "-------------------Op4:Inc Addr-----------------" severity Note;

                        mldl2 <= '0';
                mldr2 <= '0';
                        lmd <= '0';
                bus_req <= '0';
                mlresult <= '0';
                        mldl1 <= '0';
                        ms3 <= "10";
                        ms5<='1';
                        ci <= '1';
                        lmar <= '1';
                        ld_del2 := true;
                        r2_rst <= '1';
                        Op <= Op4;
```

```
                    elsif (ld_del2 = true) then
                      lmar <= '0';
                      Op <= Op5;
                      ld_del2 := false;
                    end if;


                elsif Op = Op5 then                              --Check for 2nd dl
assert  not(Op=Op5) report "-------------------Op5:Check for dl2-----------------" severity Note;

                      mldl2 <= '0';
                bus_req <= '0';
                      mldl1 <= '0';
                mlresult <= '0';
                      lmar <= '0';
                    if mdloc2out = "00000000" then       --get divisor from IMEM
                      rwmem <= '0';
                      lmr <= '0'; --put in R1
                      ms4 <= '1';
                      lmd <= '1';
                      Op <= Op9;
                    else                                 --get data from DMEM
                      rwmem <= '0';
                      lmr <= '0'; --get offset to Dl2
                      mldr2 <='1';
                      lmd<='0';
                      Op <= Op6;
                    end if;


                elsif Op = Op6 then                              --add offset to  Dl2
assert  not(Op=Op6) report "-------------------Op6:add ofset to dl2-----------------" severity Note;
                      r2_rst <= '0';
                      lmd <= '0';
                bus_req <= '0';
                      mldl1 <= '0';
                mlresult <= '0';
                      lmar<= '0';
                      mldr2 <= '0';
                      ms3<= "00";
                      ci <= '0';
                      ms1 <= '1';
                      mldl2 <= '1';
                      Op <= Op7;


                elsif Op = Op7 then
assert  not(Op=Op7) report "-------------------Op7:bus req state-----------------" severity Note;
                mldr2 <= '0';
                      lmd <= '0';
                mldl1 <= '0';
                mlresult <= '0';
```

```
                lmar<= '0';
                mldl2 <= '0';
                ms2 <= '0';
                bus_req <= '1';
                R_W <= '0';
                ms4 <= '0';
                Op <= Op8;



        elsif Op = Op8 then                             --ld R1 with divisor
assert  not(Op=Op8) report "--------------------Op8:ld multiplicand ------------------" severity Note;
                mldl2 <= '0';       --from DMEM
            mldr2 <= '0';
                ms4 <= '0';
            mldl1 <= '0';
            mlresult <= '0';
                lmar<= '0';

                if bus_gnt = '1' then
                   lmd <= '1';
                   Op <= Op9;
                else
                        lmd <= '0';
                        Op <= Op8;
                end if;



        elsif Op = Op9 then                             --ld R0 with dividend
assert  not(Op=Op9) report "--------------------Op9:ld multiplier------------------" severity Note;
                mldl2 <= '0';
            mldr2 <= '0';
                lmd <= '0';
            mldl1 <= '0';
            mlresult <= '0';
                lmar<= '0';
                ms2<= '0';
                R_W <= '0';
                bus_req <= '1';
                r2_rst <= '0';
                Op <= Op10;



        elsif Op = Op10 then
assert  not(Op=Op10) report "--------------------Op10:Bus grant=1------------------" severity Note;

                mldl2 <= '0';
            mldr2 <= '0';
                lmd <= '0';
            mldl1 <= '0';
            mlresult <= '0';
                lmar<= '0';
```

```
                        if bus_gnt = '1' then
                          lmr <= '1';
                          Op <= Op11;
                        else
                                lmr <= '0';
                                OP <= Op10;
                        end if;



                elsif Op = Op11 then                            --wait for result 20 CC's
assert  not(Op=Op11) report "-------------------Op11:20 cc ruko-----------------" severity Note;
                        mldl2 <= '0';
                    mldr2 <= '0';
                        lmd <= '0';
                    bus_req <= '0';
                        mldl1 <= '0';
                    lmar<= '0';
                        lmr <= '0';

                        ndsig<='1';--This signal tells the multiplier to process the inputs
                        if delay = "10" then
                         -- if rdy_sig ='1' then
                                mlresult <= '1';
                                --r_w<='1';--added here not in original list
                                bus_req<='1';
                                ndsig<='0';
                                Op <= Op12;
                        else
                                delay <= delay(0 downto 0)&'0';
                                mlresult <= '0';
                                Op <= Op11;
                        end if;



                elsif Op = Op12 then
assert  false report "-------------------Op12:use dl1/dl2 to store-----------------" severity Note;

                    --ndsig<='1';--added this while testing mult_icm module.Not there originally
                         --ndsig<='1'; -- change made to check
                         mldr2 <= '0';
                         lmd <= '0';
                    mlresult <= '1';
                         lmar<= '0';
                        -- R_W <= '1';

                        if mdloc2out = "00000000" then      --use DL1 for store
                                ms2<='0';
                                mldl2 <= '0';
                        else                                --use DL2 for store
                                ms2 <= '1';
                                mldl1 <= '0';
                        end if;
                        --Bus_req <= '1';
```

```
                    Op <= Op13;


                elsif Op = Op13 then
                        assert  false report "--------------------Op13:------------------" severity Note;

                        mldl2 <= '0';
                    mldr2 <= '0';
                        lmd <= '0';
                    mldl1 <= '0';
                        mlresult <= '1';
                        lmar<= '0';
                        Bus_req <= '1';
                        ndsig <= '0';
                                if bus_gnt = '1' then                   --Store Quotient in mem
                            --      fin <= '1';
                                        R_W<='1';

                                        --bus_req <= '0';

                                    --Op <= reset;
                                    Op<=Op14;
                         else
                             Op <= Op13;
                         end if;
                elsif Op=Op14 then
                assert false report "Op14 state " severity note;
                        bus_req<='0';
                        fin<='1' ;
                        R_W<='0';
                --          r_w <= '1'; -- change made to c if correct value gets written

                        Op<= reset;


                end if;
            end if;


end process;



multiplierreg: process (clk, tomultr, rst, lmr)
        begin
                if rst ='1' then
                        mrout <= (others=>'0');
                elsif clk'event and clk='1' then
                        if lmr = '1' then
                                mrout <= tomultr;
                        end if;
                end if;
end process;
```

```
multiplicandreg: process (clk,rst,lmd,tomultd)
        begin
                if rst ='1' then
                        mdout <= (others=>'0');
                elsif clk'event and clk='1' then
                        if lmd = '1' then
                                mdout <= tomultd;
                        end if;
                end if;
end process;


regr2:process(clk,r2_rst,to_r2,mldr2)
        begin
                if r2_rst='1' then
                        r2out <=(others=>'0');
                elsif clk'event and clk='1' then
                        if mldr2='1' then
                                r2out<=to_r2;
                        end if;
                end if;
end process;



dataloc1:process(clk,rst,mldl1,mux0out)
        begin
                if rst='1' then
                        mdloc1out <=(others=>'0');
                elsif clk'event and clk='1' then
                        if mldl1='1' then
                                mdloc1out<=mux0out;
                        end if;
                end if;
end process;


dataloc2:process(clk,rst,mldl2,mux1out)
        begin
                if rst='1' then
                        mdloc2out <=(others=>'0');
                elsif clk'event and clk='1' then
                        if mldl2='1' then
                                mdloc2out<=mux1out;
                        end if;
                end if;
end process;

Instmar:process(clk,rst,mux5out,lmar)
         begin
                if rst='1' then
                        iaddr_bus <=(others=>'0');
                elsif clk'event and clk='1' then
                        if lmar='1' then
                                iaddr_bus<=mux5out;
                        end if;
```

```
                end if;
end process;

reg_result: process (clk,rst,multout, mlresult)
        begin
                if rst ='1' then
                        multrslt <= (others=>'0');

                elsif clk'event and clk='1' then
                        if mlresult = '1' then
                                multrslt <= multout;
                        end if;
                end if;
end process;
end Behavioral;
```

Module Name : mult.vhd

```
----------------------------------------------------------------------------
--Multiplier version 1.0
--Date: 02/27/2004
----------------------------------------------------------------------------
----------------------------------------------------------------------------
--Explanation of signals
--a and b are 8 bit inputs(unsigned) and can be thought of as the muliplier and
--multiplicand.They produce an output which can be max 16 bits
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity mult is
   Port ( a : in std_logic_vector(7 downto 0);
        b : in std_logic_vector(7 downto 0);
        q : out std_logic_vector(15 downto 0);
                clk:in std_logic;
        newdata : in std_logic);
end mult;

architecture Behavioral of mult is
--signal listings here
signal qsig: std_logic_vector(15 downto 0);
begin
q<=qsig;
multiply: process(clk,newdata,a,b) is
begin
if (clk'event and clk='1') then
  if (newdata='1') then
                qsig<=a*b;--Multiply the inputs
  else
                qsig<=qsig;--Latch on to the values
   end if;
```

```
end if;
end process;
end Behavioral;
```

Module Name : mult_imem.xco (Xilinx IP Core)

```
------------------------------------------------------------------------------
--  This file is owned and controlled by Xilinx and must be used          --
--  solely for design, simulation, implementation and creation of         --
--  design files limited to Xilinx devices or technologies. Use           --
--  with non-Xilinx devices or technologies is expressly prohibited       --
--  and immediately terminates your license.                              --
--                                                                        --
--  XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         --
--  SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR               --
--  XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION       --
--  AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION           --
--  OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS             --
--  IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,               --
--  AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      --
--  FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              --
--  WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               --
--  IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        --
--  REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       --
--  INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       --
--  FOR A PARTICULAR PURPOSE.                                             --
--                                                                        --
--  Xilinx products are not intended for use in life support              --
--  appliances, devices, or systems. Use in such applications are         --
--  expressly prohibited.                                                 --
--                                                                        --
--  (c) Copyright 1995-2003 Xilinx, Inc.                                  --
--  All rights reserved.                                                  --
------------------------------------------------------------------------------
-- You must compile the wrapper file mult_imem.vhd when simulating
-- the core, mult_imem. When compiling the wrapper file, be sure to
-- reference the XilinxCoreLib VHDL simulation library. For detailed
-- instructions, please refer to the "CORE Generator Guide".

-- The synopsys directives "translate_off/translate_on" specified
-- below are supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity
-- synthesis tools. Ensure they are correct for your synthesis tool(s).

-- synopsys translate_off
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

Library XilinxCoreLib;
ENTITY mult_imem IS
        port (
        addr: IN std_logic_VECTOR(2 downto 0);
        clk: IN std_logic;
        din: IN std_logic_VECTOR(15 downto 0);
        dout: OUT std_logic_VECTOR(15 downto 0);
        we: IN std_logic);
```

```
END mult_imem;

ARCHITECTURE mult_imem_a OF mult_imem IS

component wrapped_mult_imem
	port (
	addr: IN std_logic_VECTOR(2 downto 0);
	clk: IN std_logic;
	din: IN std_logic_VECTOR(15 downto 0);
	dout: OUT std_logic_VECTOR(15 downto 0);
	we: IN std_logic);
end component;

-- Configuration specification
	for all : wrapped_mult_imem use entity XilinxCoreLib.blkmemsp_v5_0(behavioral)
		generic map(
			c_sinit_value => "0",
			c_reg_inputs => 0,
			c_yclk_is_rising => 1,
			c_has_en => 0,
			c_ysinit_is_high => 1,
			c_ywe_is_high => 1,
			c_ytop_addr => "1024",
			c_yprimitive_type => "16kx1",
			c_yhierarchy => "hierarchy1",
			c_has_rdy => 0,
			c_has_limit_data_pitch => 0,
			c_write_mode => 0,
			c_width => 16,
			c_yuse_single_primitive => 0,
			c_has_nd => 0,
			c_enable_rlocs => 0,
			c_has_we => 1,
			c_has_rfd => 0,
			c_has_din => 1,
			c_ybottom_addr => "0",
			c_pipe_stages => 0,
			c_yen_is_high => 1,
			c_depth => 8,
			c_has_default_data => 0,
			c_limit_data_pitch => 18,
			c_has_sinit => 0,
			c_mem_init_file => "mult_imem.mif",
			c_default_data => "0",
			c_ymake_bmm => 0,
			c_addr_width => 3);
BEGIN

U0 : wrapped_mult_imem
		port map (
			addr => addr,
			clk => clk,
			din => din,
			dout => dout,
			we => we);
END mult_imem_a;
```

```
-- synopsys translate_on
```

Module Name : pe.vhd

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(),
--etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity PE is
  port (Data_Bus : inout std_logic_vector(15 downto 0);
       R_W : out std_logic;
       Cntl_bus : in std_logic_vector(15 downto 0);
              RST, ODR, IDV : in std_logic;
              clk, Bus_grant : in std_logic;
              CInstr_rdy : in std_logic;
              inpt : in std_logic_vector(15 downto 0);
              Bus_req, Snd_Instr, Fin : out std_logic;
              Addr : out std_logic_vector(7 downto 0);
              Rq_inpt, Rq_outpt : out std_logic;
              STOPLOOP : out std_logic;
              -- added for dbugging
              R3_out_dbug : out std_logic_vector( 15 downto 0);
              shft_out_dbug : out std_logic_vector( 15 downto 0 );
                     dbug_st_pe : out std_logic_vector( 3 downto 0);
                     tmp4_dbug : out std_logic_vector(15 downto 0);
                     m5outdbg: out std_logic_vector(15 downto 0);
                     R0_out_dbug : out std_logic_vector(15 downto 0);
                     tmp3_dbug: out std_logic_vector(2 downto 0);
                     tmp2_dbug: out std_logic_vector(1 downto 0);
                     tmp1_dbug: out std_logic_vector(1 downto 0)          ;
                     tmp44_dbug: out std_logic_vector(4 downto 0)         ;
           tmp5_dbug: out std_logic_vector(3 downto 0)          ;
                     count_out_pe : out std_logic_vector (7 downto 0)
                    -- tmp6_dbug: out std_logic_vector(1 downto 0)
           ) ;
end PE;

Architecture pe_arch of pe is
component Reg_B_in is
port( din: in std_logic_vector(15 downto 0);              -- data from data_bus
                     dout:out std_logic_vector(15 downto 0);      -- register output
                     clk: in std_logic;                                                                    -- clk
                     rst: in std_logic;                                                                    --
Asynch Reset
                     ctrlreg: in std_logic
          -- Control signal
            );
end component;


component Controller2 is
```

```
        port (reset,clk, Int_Pend : in std_logic;
                Z, S, V, IDV, ODR : in std_logic;
            IR : in std_logic_vector(15 downto 12);
            Int_rdy, B_grnt : in std_logic;
                CE, R_W, LMDR1, LMDR0 : out std_logic;
                LMAR,LV, LZ, LS : out std_logic;
                S0, S1, S2, S3, S4 : out std_logic;
                S5, S6, S7, S8, S9 : out std_logic;
                S10, LR5, Snd_Inst, B_req : out std_logic;
                Ci, LPC, INC_PC, S11 : out std_logic;
                LIR0, LIR1, LR4 : out std_logic;
                Clr_dec, Ld_dec : out std_logic;
                Req_inpt, Req_otpt : out std_logic;
                STOPLOOP : out std_logic;
                dbug_st : out std_logic_vector( 3 downto 0);
                m5ctrl : out std_logic;
                count_out : out std_logic_vector (7 downto 0);
                decide: out std_logic
                );
end component;

component mem_1 is
        port (data_bus : inout std_logic_vector(15 downto 0);
            Idata_bus : inout std_logic_vector(15 downto 0);
            clk, rst, CE: in std_logic;
            LMAR : in std_logic;
            LMDR1, LMDR0 : in std_logic;
            Addr : in std_logic_vector(7 downto 0);
            mux16 : in std_logic_vector(15 downto 0);
            Fin, sel_Ibus : out std_logic;
            MAddr_out : out std_logic_vector(7 downto 0));
end component;




component mux16_4x1
        Port (line_out : out std_logic_vector(15 downto 0);
                Sel : in std_logic_vector(1 downto 0);
                line_in3,line_in2,line_in1,line_in0 : in std_logic_vector(15 downto
0));
end component;

component mux16_5x1
        Port (line_out : out std_logic_vector(15 downto 0);
            Sel : in std_logic_vector(2 downto 0);
            line_in4,line_in3,line_in2,line_in1,line_in0 : in
std_logic_vector(15 downto 0));
end component;

component mux8_4x1
        Port (line_out : out std_logic_vector(7 downto 0);
                Sel : in std_logic_vector(1 downto 0);
                line_in3,line_in2,line_in1,line_in0 : in std_logic_vector(7 downto
0));
```

```
end component;

component PC
        Port (q_out : buffer std_logic_vector(7 downto 0);
                --q_out : inout std_logic_vector(7 downto 0);
                clk, clr : in std_logic;
                D : in std_logic_vector(7 downto 0);
                load, inc : in std_logic);
end component;

component REGS
        port (q_out : buffer std_logic_vector(15 downto 0);
                 --q_out : inout std_logic_vector(15 downto 0);
            clk, clr : in std_logic;
            D : in std_logic_vector(15 downto 0);
            Load : in std_logic);
End component;

component Shifter_16
        port(ALU_out : in std_logic_vector(15 downto 0);
            Sel : in std_logic_vector(1 downto 0);
            Shf_out : out std_logic_vector(15 downto 0)) ;
End component;

component ALU
        port(a, b : in std_logic_vector(15 downto 0);
            S8, S7, Cntl_I : in std_logic;
            C_out : out std_logic;
            Result : out std_logic_vector(15 downto 0)) ;
End component;



component mux16bit_2x1 is
        Port (line_out : out std_logic_vector(15 downto 0);
                        Sel : in std_logic;
                        line_in1,line_in0 : in std_logic_vector(15 downto 0));
end component;




Signal PC_out,MAR_val : std_logic_vector(7 downto 0);
signal PC_VAL: std_logic_vector(7 downto 0);
Signal R4_out, IR0_70, IR1_70, IR1_158 : std_logic_vector(7 downto 0);
signal R0_out, R1_out,R2_out, R3_out: std_logic_vector(15 downto 0);
signal shft_out, Alu_out, MDR_val: std_logic_vector(15 downto 0);
signal Alu_in : std_logic_vector(15 downto 0);
signal Inpt_Sel, Dec_Sel : std_logic_vector(1 downto 0);
signal IR_1512: std_logic_vector(15 downto 12);
signal Co, Ci : std_logic;
signal reg, Reg0_en, Reg1_en,Reg2_en, Reg3_en : std_logic;
signal Vo, So, Zo : std_logic;
signal CE, R_W1 : std_logic;
signal LMDR1, LMDR0, LMAR : std_logic;
```

```
signal LPC, INC_PC, LIR0, LIR1 : std_logic;
signal  S9, S8, S7, S6 : std_logic;
signal LR4:std_logic;
signal S5, S4, S3, S2, S1, S0 : std_logic;
signal V, S, Z, LV, LS, LZ : std_logic;
signal temp1, temp2, val2 : std_logic_vector(1 downto 0);
signal temp4, sixteen0, val1, B_in : std_logic_vector(15 downto 0);
-- added for debugging
signal val11 : std_logic_vector(15 downto 0);
signal Clr_dec, Ld_dec, one0, Instr_rdy : std_logic;
signal eight0, R5_out, mem_addr_out : std_logic_vector(7 downto 0);
signal LR5, sel_Ibus : std_logic;
signal S10,S11: std_logic;
signal Instr_bus, Idata_bus : std_logic_vector(15 downto 0);
signal temp3 : std_logic_vector(2 downto 0);
signal m5out:std_logic_vector(15 downto 0);
signal m5ctrl:Std_logic;
signal temp44 : std_logic_vector( 4 downto 0);
signal temp5 : std_logic_vector ( 3 downto 0);
signal count_out : std_logic_vector( 7 downto 0);
signal bus_req_pe : std_logic;
signal dout_bin: std_logic_vector(15 downto 0);-- Data ouput of the Register Reg_Bin
signal decide : std_logic; -- Control for the register Reg_Bin before ALU mux
signal R5mod: std_logic_vector(15 downto 0);
begin

-- added for dbugging
R5mod <= eight0&R5_out;
tmp1_dbug <= temp1;
tmp2_dbug <= temp2;
tmp3_dbug <= temp3;
R3_out_dbug <= R3_out;
R0_out_dbug <= R0_out;
shft_out_dbug <= shft_out;
tmp4_dbug <= temp4;
m5outdbg<=m5out;
count_out_pe <= count_out;
--
sixteen0 <= "0000000000000000";
eight0 <= "00000000";
one0 <= '0';

temp1 <= S9&S4;
temp2 <= S3&S2;
temp3 <= S11&S1&S0;
IR_1512 <= temp4(15 downto 12);
Dec_Sel <= temp4(11 downto 10);
Inpt_Sel <= temp4(9 downto 8);
IR0_70 <= temp4(7 downto 0);
-- added ports for viewing the control signals -----

temp44 <= s10&s8&s7&s6&s5;
temp5 <= LMDR1&LMDR0&LMAR&LPC;
--temp6 <= R_W& B_req;

tmp44_dbug <= temp44;
```

```
tmp5_dbug <= temp5;
--tmp6_dbug <= temp6;
Vo <= V;
So <= S;
Zo <= Z;
-- added for debugging assignment to a signal --------
bus_req <= bus_req_pe;



Status: process (clk)
  Begin
    If (clk'event and clk='0') then
      if Alu_out = "0000000000000000" then
                    Z <= '1';
      else
                    Z <= '0';
      end if;
      S <= Alu_out(15);
      V <= (Co xor Ci);
    End if;
  End process;

--B_in <= eight0&R5_out when S10 = '1' else                    --new mux for immediate  ops
              --Data_bus;

------------ change #1 to bring out correct values at the other input of the ALU
--------------------------
--B_in <= eight0&R5_out when S10 = '1' else                    --new mux for immediate   ops
--              Data_bus when S10 ='0';-- else

RegBin_mux: mux16bit_2x1 port map(line_out => B_in,Sel => S10,
line_in0=>dout_bin, line_in1 =>
R5mod);
RegBin: Reg_B_in port map(clk=> clk, rst => rst, din => data_bus, dout
=> dout_bin,ctrlreg =>
decide);
M1: mux8_4x1  port map(PC_val,temp1,eight0,R4_out,IR1_158,IR0_70);
M2: mux8_4x1 port map(MAR_val,temp2,R3_out(7 downto
0),IR1_70,IR0_70,PC_out);
M3: mux16_5x1 port
map(MDR_val,temp3,Instr_Bus,sixteen0,shft_out,Alu_in,inpt);
M4: mux16_4x1 port map(Alu_in,Inpt_Sel,R3_out,R2_out,R1_out,R0_out);
M5 : mux16bit_2x1 port map(m5out,m5ctrl,shft_out,temp4);

P1: PC  port map(PC_out, clk, RST, PC_val, LPC, INC_PC);
R5: PC port map(R5_out, clk, RST, IR0_70, LR5, one0);
R4: PC  port map(R4_out, clk, one0, PC_out, LR4,one0);   --modified needed 8 bit reg
--R0: REGS  port map(R0_out, clk, one0, shft_out, Reg0_en);
R0: REGS  port map(R0_out, clk, RST, shft_out, Reg0_en);
--R1: REGS  port map(R1_out, clk, one0, shft_out, Reg1_en);
--R2: REGS  port map(R2_out, clk, one0, shft_out, Reg2_en);
--R3: REGS  port map(R3_out, clk, one0, m5out, Reg3_en);

R1: REGS  port map(R1_out, clk, RST, shft_out, Reg1_en);
R2: REGS  port map(R2_out, clk, RST, shft_out, Reg2_en);
```

```
R3: REGS  port map(R3_out, clk, RST, m5out, Reg3_en);

-- Get input from Controller or Instr. Mem

Instr_Bus <= IData_bus when sel_Ibus = '1' else
                Cntl_bus  when sel_Ibus = '0' else                    --added to fix bus  conflicts
             (others=>'0');

--Ir0: REGS port map(temp4, clk, one0, Instr_Bus, LIR0);

Ir0: REGS port map(temp4, clk, RST, Instr_Bus, LIR0);


-- option 1 : considering that the IR1 is not used at all
-- commenting  the val1 which caused the buffer problem.

--val1 <= IR1_158&IR1_70;
-- added for dbugging
--val11<= val1;
--Ir1: REGS port map(val11, clk, one0, Instr_Bus, LIR1);

val2 <= s6&s5;
SH1: Shifter_16 port map(Alu_out, val2, shft_out) ;

A1: ALU port map(Alu_in, B_in, S8, S7, Ci, Co, Alu_out) ;

R_W <= R_W1;                        --sent to DMEM
Addr <= mem_addr_out;               --sent to DMEM
Mem1: mem_1 port map(DATA_bus, IData_bus, clk, RST, CE,
LMAR,LMDR1,LMDR0,
                    MAR_val,Mdr_val, FIN, sel_Ibus, mem_addr_out);


-- This provides Control for getting instructions from PE Controller
Instr_Rdy <= CInstr_Rdy when ((PC_out="00000000") or
(PC_out="00000001")
                                    or (PC_out="00000010")) else
        '1';



C1: Controller2 port map(RST, clk, one0, Zo, So, Vo, IDV, ODR, IR_1512,
Instr_Rdy, Bus_grant,
                        CE, R_W1, LMDR1,LMDR0, LMAR,LV, LZ, LS, S0, S1, S2, S3, S4, S5, S6,
S7,
                        S8, S9, S10, LR5, Snd_Instr, bus_req_pe, Ci, LPC, INC_PC, S11, LIR0,
LIR1,
                        LR4, Clr_dec, Ld_dec, Rq_inpt, Rq_outpt,
STOPLOOP,dbug_st_pe,m5ctrl,count_out,decide =>
decide);

Decoder: process (clk, Clr_dec)
   begin
        if (clk'event and clk='1') then
          if (Clr_dec = '1') then
                Reg3_en <='0'; Reg2_en <='0';
```

```
                Reg1_en<='0'; Reg0_en <='0';
          elsif (Ld_dec='1') then
             case (Dec_Sel) is
                when "11" => Reg3_en <='1';
                             Reg2_en <='0';
                             Reg1_en <='0';
                             Reg0_en <='0';
                When "10" => Reg3_en <='0';
                             Reg2_en <='1';
                             Reg1_en <='0';
                             Reg0_en <='0';
                When "01" => Reg3_en <='0';
                             Reg2_en <='0';
                             Reg1_en <='1';
                             Reg0_en <='0';
                When "00" => Reg3_en <='0';
                             Reg2_en <='0';
                             Reg1_en <='0';
                             Reg0_en <='1';
                When others => null;
             End case;
      End if;
   End if;
End process;
End architecture;
```

Module Name : aluv.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity ALU is
        port(a, b : in std_logic_vector(15 downto 0);
            S8, S7, Cntl_I : in std_logic;
            C_out : out std_logic;
            Result : out std_logic_vector(15 downto 0)) ;
End entity;

Architecture alu_arch of alu is

signal sel : std_logic_vector(2 downto 0);

component add_subber16

        port (
        A: IN std_logic_VECTOR(15 downto 0);
        B: IN std_logic_VECTOR(15 downto 0);
        C_IN: IN std_logic;
        C_OUT: OUT std_logic;
        ADD_SUB: IN std_logic;
        Q_OUT: OUT std_logic_VECTOR(15 downto 0));
end component;
```

```
signal as_out : std_logic_vector(15 downto 0);
signal asC_out, A_S : std_logic;
signal carryI : std_logic;

begin

sel <= S8&S7&Cntl_i;

ad_sb: add_subber16 port map
        ( A => a, B => b, C_IN=>CarryI, C_OUT =>asC_out,ADD_SUB => A_S,Q_OUT => as_out);

ops: process (sel, a, b, as_out, asC_out)
        begin
                case (sel) is
                        when "000" => result <= a or b;
                                      C_out<='0'; CarryI <='0';
                                      A_S <= '1';
                        When "001" => result <= a or b;
                                      C_out<='0'; CarryI <='0';
                                      A_S <= '1';
                        When "100" => A_S <= '1';                   --add op
                                  result <= as_out;
                                  C_out <= asC_out;
                                  CarryI <='0';
                        When "101" => A_S <= '0';                   --sub op
                                      result <= as_out;
                                      C_out <= asC_out;
                                      CarryI <='0';
                        When "010" => result <= b;                  --pass through
                                      C_out <='0'; CarryI <='0';
                                      A_S <= '1';
                        When "011" => result <= b;                  --pass through
                                      C_out <='0'; CarryI <='0';
                                      A_S <= '1';
                        When "110" => result <= a and b;
                                      C_out<='0'; CarryI <='0';
                                      A_S <= '1';
                        When "111" => result <= as_out;             --Increment op
                                      C_out<= asC_out;
                                      A_S <= '1';
                                      CarryI <='1';
                        When others => null;
                    End case;
        End process;

End architecture;
```

Module Name : addsub16_synthable.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
--use ieee.std_logic_arith.all;

ENTITY add_subber16 IS
```

```
  PORT(
    A: IN std_logic_vector(15 DOWNTO 0);
    B: IN std_logic_vector(15 DOWNTO 0);
    C_IN: IN std_logic;
    C_OUT: OUT std_logic;
    ADD_SUB: IN std_logic;
    Q_OUT: OUT std_logic_vector(15 DOWNTO 0));
END add_subber16;

ARCHITECTURE sim OF add_subber16 IS
    SIGNAL S: std_logic_vector(15 DOWNTO 0);
    SIGNAL S1: std_logic_vector(15 DOWNTO 0);
    SIGNAL AA: std_logic_vector(15 DOWNTO 0);
    SIGNAL C: std_logic_vector(16 DOWNTO 0);
    SIGNAL T: std_logic_vector(15 DOWNTO 0);

BEGIN
Q_OUT<=S;
PROCESS(A,B,C_IN,ADD_SUB,C,T,AA,S1,S)
begin
if ADD_SUB='1' THEN
                    C(0)<= C_IN;
                    for i in 0 to 15 loop
                            S(i) <= A(i) xor B(i) xor C(i);
                            C(i+1)<= (A(i) and B(i)) or (A(i) and C(i)) or (B(i) and C(i));
                    end loop;
                    C_OUT <= C(16);
else
                    T<=NOT (B+C_IN);
                    AA<=A+1;

                    C(0) <= C_in;
                    for i in 0 to 15 loop
                            S1(i) <= AA(i) xor T(i) xor C(i);
                            C(i+1)<= (AA(i) and T(i)) or (AA(i) and C(i)) or (T(i) and C(i));
                    end loop;
                    --C_OUT <= NOT C(16);
                    C_OUT <=  C(16);
                    if C(16) = '0'
                     then
                    --if s1(15) = '1' and A(15) = '0' then
                            s <= (not s1) +1;
                    else s <= s1;
                    end if;
end if;
end process;
END sim;
```

Module Name : controller.vhd

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

```
use IEEE.std_logic_unsigned.all;
entity Controller2 is
 port (reset,clk, Int_Pend : in std_logic;
  Z, S, V, IDV, ODR : in std_logic;
        IR : in std_logic_vector(15 downto 12);
        Int_rdy, B_grnt : in std_logic;
  CE, R_W, LMDR1, LMDR0 : out std_logic;
  LMAR,LV, LZ, LS : out std_logic;
   S0, S1, S2, S3, S4 : out std_logic;
  S5, S6, S7, S8, S9 : out std_logic;
  S10, LR5, Snd_Inst, B_req : out std_logic;
  Ci, LPC, INC_PC, S11 : out std_logic;
  LIR0, LIR1, LR4 : out std_logic;
  Clr_dec, Ld_dec : out std_logic;
  Req_Inpt, Req_Otpt : out std_logic;
  STOPLOOP: out std_logic;
  dbug_st : out std_logic_vector( 3 downto 0);
  m5ctrl : out std_logic;
  count_out : out std_logic_vector (7 downto 0);
  decide : out std_logic
  );
End controller2;
Architecture cont_arch of controller2 is
Type state_type is (RST, InstF, ID, OP0, OP1, OP2, OP3, OP4, OP5,OP6, OP7, OP8, OP9,
     OP10, OP11, OP12,OP13);
Signal STATE : state_type;
Signal count : std_logic_vector(7 downto 0);   --shift reg for internal states
signal dbug_st_sig : std_logic_vector( 3 downto 0); -- added for checking the states
begin
contl: process (clk, reset)
 begin
    if (reset='1') then
  STATE<=RST;
    elsif (clk'event and clk='1') then
    if (STATE=RST) then
    dbug_st_sig <= "1111";
         Snd_Inst <= '0';
         LMDR1 <= '1'; LMDR0 <= '1'; B_req <='0';
        CE <= '0'; R_W <='0'; Count <= "00000001";
        LMAR<='0'; LV<='0'; LZ<='0'; LS<='0';
        S0<='0'; S1<='0'; S2<='0'; S3<='0'; S4<='0';
        S5<='0'; S6<='0'; S7<='0'; S8<='0'; S9<='0';
        Ci<='0'; LR4<='0'; LIR0<='0'; LIR1<='0';
         Clr_dec <= '1'; Ld_dec <='0'; S11 <= '0';
        INC_PC<='0'; LPC<='0'; STATE <= InstF;
        S10 <= '0'; LR5 <= '0';
        req_inpt <= '0'; req_otpt <= '0';
        STOPLOOP <= '0';decide <= '0';
    m5ctrl <='0'; -- send shiftout to M5
    elsif (STATE=InstF) then
    dbug_st_sig <= "1110";
    m5ctrl <='0';
       decide <='0';
          LMDR1<='0'; LMDR0<='0'; S11 <= '0';
          LR5 <= '0'; S10 <='0'; Ci <='0';
          Ld_dec <='0'; S0 <= '1'; B_req <='0';
```

```
LPC <= '0'; INC_PC<='0'; LMAR<='0';
req_inpt <= '0'; req_otpt <= '0';
      CE<='0'; LIR0<='0'; LIR1<='0'; R_W <='0';  --added R_W part here
    STOPLOOP <= '0';
           if ((Int_Pend='1')or (Count="00000010")) then
           if (Count="00000001") then
             LR4 <= '1';Clr_dec<='1';
             Count<= Count(6 downto 0)&'0';
             STATE<=InstF;
           elsif (Count="00000010") then
             LPC <= '1'; S4 <= '1'; S9 <= '1'; LR4 <='0';
             Count<=Count(6 downto 0)&'0'; STATE <= InstF;
           End if;
      elsif ((Int_Pend='0')or(Count="00000100")) then
           LMAR <= '1'; Clr_dec <= '1';
           S2 <= '0'; S3 <= '0'; Snd_Inst <= '1';
           STATE <= ID;
           if (Count="00000100") then
 Count <= "00"&Count(7 downto 2);
           End if;
      End if;
elsif (STATE=ID) then
   dbug_st_sig <= "1101";
     if (Count="00000001") then
              if Int_rdy = '1' then   --check to see if Instr ready
                LR4<='0'; LPC<='0'; LMAR<='0';
          CE <= '1'; R_W <='0'; Clr_dec <= '0';
          S10 <= '0'; Snd_Inst <= '0';
          LMDR1 <='1'; LMDR0<='0'; -- mdr output is mux16
    S11 <= '1'; S0 <= '0'; S1 <= '0';    -- mux output is instr_bus
           -- added m5ctrl signal to select IR0
  -- m5ctrl <='0';
          INC_PC <='1'; B_req <= '0';
          req_inpt <= '0'; req_otpt <= '0';
          Count <= Count(6 downto 0)&'0';
     STATE <= ID;
   else
     Count <= Count;
     STATE <= ID;
   end if;
elsif (Count="00000010") then
     INC_PC <= '0'; CE <= '0';
     LIR0<='1'; -- instruction loaded in the IR0
 LMDR1 <= '1'; LMDR0 <= '1';  --hold MDR memory
     Count <= Count(6 downto 0)&'0';
     STATE <= ID;
elsif (Count="00000100") then
    case (IR) is                    --decode opcode
           when "0000" => STATE <= OP0;
           when "0001" => STATE <= OP1;
           when "0010" => STATE <= OP2;
           when "0011" => STATE <= OP3;
           when "0100" => STATE <= OP4;
           when "0101" => STATE <= OP5;
           when "0110" => STATE <= OP6;
           when "0111" => STATE <= OP7;
```

```
            when "1000" => STATE <= OP8;
            when "1001" => STATE <= OP9;
            when "1010" => STATE <= OP10;
        when "1011" => STATE <= OP11;
        when "1100" => STATE <= OP12;
                                when "1101" => STATE <= OP13;

            when others => STATE <= RST; --error has occurred RST
          end case;
  Count <= "00"&Count(7 downto 2); LIR0 <= '0';
End if;
elsif (STATE=OP0) then
dbug_st_sig <= "0000";
      if (Count="00000001") then
        S10 <= '0'; S11 <= '0';
        req_inpt <= '1'; req_otpt <= '0';  --signal input wanted
        if (IDV='0') then
       STATE <= OP0; Count <= Count;
        else
            STATE <= OP0;
            Count <= Count(6 downto 0)&'0';
        End if;
      elsif (Count="00000010") then
            req_inpt <= '0'; req_otpt <= '0';
            LMDR1<='1'; LMDR0 <='0';
            LMAR<='1'; S2<='1'; S0<='0';
            S3<='1'; S1<='0'; B_req <= '1';
            Count <= Count(6 downto 0)&'0';
            STATE <= OP0;
      elsif (Count="00000100") then
            if B_grnt = '1' then  --check bus access
              LMDR1<='0'; LMDR0<='1';
              LMAR<='0';
              CE <='1'; R_W<='1';
              Count <= "00"&Count(7 downto 2);
              STATE <= InstF;
            else
              Count <= Count;
              STATE <= OP0;
            end if;
      end if;
elsif (STATE=OP1) then
dbug_st_sig <= "0001";
      if (Count = "00000001") then
            LMAR <= '1'; S2<='1'; S3<='1';
            S10 <= '0'; B_req <= '0'; S11 <= '0';
            Count <= Count(6 downto 0)&'0';
            STATE <= OP1;
      elsif (Count = "00000010") then
       LMAR <= '0'; B_req <= '1';
       Count <= Count(6 downto 0)&'0';
             STATE <= OP1;
      elsif (Count = "00000100") then
            if B_grnt = '1' then   --check bus access
              CE <='1'; R_W<='0'; Ld_dec <='1';
              LMDR1<='0'; LMDR0<='0'; decide <= '1';
```

```
            LMAR <= '0';
            Count <= Count(6 downto 0)&'0';
            STATE<=OP1;
          else
            Count <= Count;
            STATE<= OP1;
          end if;
      elsif (Count = "00001000") then
          CE <='0'; LMDR0<='1'; B_req <='0';
          S8<='1'; S7<='0'; Ci<='0';
          ld_dec <= '0'; clr_dec <= '1';
          Count <= "000"&Count(7 downto 3);
          STATE <= InstF;
      End if;
 elsif (STATE=OP2) then
 dbug_st_sig <= "0010";
      if (Count = "00000001") then
          LMAR<='1'; S2<='1'; S3<='1';
          LMDR1<='1'; LMDR0<='0'; B_req <= '1';
          S0<='1'; S1<='0'; S10 <= '0';
          Count <= Count(6 downto 0)&'0';
          STATE <= OP2; S11 <= '0';
      elsif (Count="00000010") then
          if B_grnt = '1' then
            LMAR <= '0'; LMDR1<='0'; LMDR0<='1';
            CE<='1'; R_W <= '1';
            Count <= '0'&Count(7 downto 1);
            STATE <= InstF;
          else
            Count <= Count;
            STATE <= OP2;
          end if;
      end if;
 elsif (STATE=OP3) then
 dbug_st_sig <= "0011";
      LPC <= '1'; S4 <= '0'; S9<='0';
      S10 <= '0'; B_req <= '0';
      STATE <= InstF; S11 <= '0';
elsif (STATE=OP4) then
 dbug_st_sig <= "0100";
      if (Count="00000001") then
         LMAR <= '1'; S2<='1'; S3<='1';
              S10 <= '0'; B_req <= '0'; S11 <= '0';
              Count <= Count(6 downto 0)&'0';
         STATE <= OP4;
      elsif (Count="00000010") then
         LMAR <= '0'; B_req <= '1';
         Count <= Count(6 downto 0)&'0';
         STATE <= OP4;
      elsif (Count = "00000100") then
         if B_grnt = '1' then
          LMAR<='0'; --Ld_dec <='1';
          LMDR1 <='0'; LMDR0 <= '0';  --place in MDR
          CE <= '1'; R_W<='0'; S8<='0'; S7<='1';
          Ci<='0'; S5 <= '0'; S6<='0';
          Count <= Count(6 downto 0)&'0';
```

```
            STATE <= OP4;
           else
            Count <= Count;
            STATE <= OP4;
           end if;
        elsif (Count="00001000") then
            CE <= '0'; --Ld_dec <= '0';
            LMDR0 <= '1'; S8 <= '1'; S7 <= '0';
            Ci <= '1';   --subtract
            --LMAR <= '1';
            S2<='0'; S3<='0'; B_req <= '0';
            Count <= Count(6 downto 0)&'0';
            STATE <= OP4;
        elsif (Count="00010000") then
            if ((S xor V)='0') then
                    LMDR0<= '0';
              LPC<='1'; S4<='0'; S9<='0';
              Count <= "0000"&Count(7 downto 4);
              STATE <= InstF;
           else
         Count <= "0000"&Count(7 downto 4);
          STATE <= InstF;
           end if;
        end if;

--   elsif (STATE=OP5) then
--      dbug_st_sig <= "0101";
--        if (Count = "00000001") then
--            LMAR<='1'; S2<='1'; S3<='1';
--            S10 <= '0'; B_req <='0'; S11 <= '0';
--            Count <= Count(6 downto 0)&'0';
--            STATE<= OP5;
--       elsif (Count = "00000010") then
--            LMAR <= '0'; B_req <= '1';
--
--            Count <= Count(6 downto 0)&'0';
--            STATE<= OP5;
--       elsif (Count = "00000100") then
--            if B_grnt = '1' then
--              LMAR<='0';
--              CE<='1'; R_W<='0';
--              S8<='1'; S7<='0'; Ci<='1';
--              s11<='0'; s1<='1'; s0<='0';
--              LMDR1 <='1'; LMDR0 <= '0';
--            S2<='1'; S3<='1'; LMAR <= '1';
--              Count <= Count(6 downto 0)&'0';
--              STATE<=OP5;
--            else
--             Count <= Count;
--             STATE <= OP5;
--            end if;
--       elsif (Count = "00001000") then
--             LMDR1 <='0'; LMDR0 <= '1';
--             R_W <='1'; CE <='1';
--             LMAR <= '0';
--             Count <= Count(6 downto 0)&'0';
```

```
--                STATE <= OP5;
--         elsif (Count = "00010000") then
--                B_req <='0';
--                Count <= "0000" & Count(7 downto 4);
--                STATE <= InstF;
--         end if;



-- Replaced logic for subtraction with logic for addition making suitable changes in
-- ALU signals.
      elsif (STATE=OP5) then
    dbug_st_sig <= "0101";
         if (Count = "00000001") then
              LMAR <= '1'; S2<='1'; S3<='1';
              S10 <= '0'; B_req <= '0'; S11 <= '0';
              Count <= Count(6 downto 0)&'0';
              STATE <= OP5;
         elsif (Count = "00000010") then
          LMAR <= '0'; B_req <= '1';
          Count <= Count(6 downto 0)&'0';
               STATE <= OP5;
         elsif (Count = "00000100") then
              if B_grnt = '1' then   --check bus access
                CE <='1'; R_W<='0'; Ld_dec <='1';
                LMDR1<='0'; LMDR0<='0';
                LMAR <= '0';
                                                          decide <= '1';
                Count <= Count(6 downto 0)&'0';
                STATE<=OP5;
              else
                Count <= Count;
                STATE<= OP5;
              end if;
         elsif (Count = "00001000") then
              CE <='0'; LMDR0<='1'; B_req <='0';
              S8<='1'; S7<='0'; Ci<='1';
              ld_dec <= '0'; clr_dec <= '1';
              Count <= "000"&Count(7 downto 3);
              STATE <= InstF;
         End if;


-- End changed part



   elsif (STATE=OP6) then
      dbug_st_sig <= "0110";
        if (Count = "00000001") then
             LMAR<='1'; S2<='1'; S3<='1';
             S10 <= '0'; B_req <= '0';
             Count <= Count(6 downto 0)&'0';
             STATE <= OP6; S11 <= '0';
        elsif (Count = "00000010") then
             LMAR <='0'; B_req <= '1';
```

```
            Count <= Count(6 downto 0)&'0';
            STATE <= OP6;
        elsif (Count = "00000100") then
            if B_grnt = '1' then
              LMAR<='0';
              LMDR1<='0'; LMDR0<='0';
              CE<='1'; R_W<='0';
              req_inpt <= '0'; req_otpt <= '1'; --signal output rdy
              Count <= Count(6 downto 0)&'0';
              STATE <= OP6;
            else
              Count <= Count;
              STATE <= OP6;
            end if;
         elsif (Count = "00001000") then
            CE<='0';
            if (ODR='0') then
              LMDR1 <='1'; LMDR0 <= '1';   --MAINTAIN DATA
              STATE <= OP6; Count <= Count;
              B_req <= '0';
            else
              LMDR1<='0'; LMDR0<='1'; B_req <= '0';
              req_inpt <= '0'; req_otpt <= '0';
              Count <= "000"&Count(7 downto 3);
              STATE <= InstF;
            end if;
        end if;
    elsif (STATE=OP7) then
       dbug_st_sig <= "0111";
        if (Count = "00000001") then
            LMAR <= '1'; S2 <='1'; S3<='1';
            Count <= Count(6 downto 0)&'0';
            S10 <='0'; B_req <= '0';
            STATE <= OP7; S11 <= '0';
        elsif (Count = "00000010") then
            LMAR <= '0'; B_req <= '1';
            Count <= Count(6 downto 0)&'0';
            STATE <= OP7;
        elsif (Count = "00000100") then
            if B_grnt = '1' then
              LMAR<='0'; Ld_dec <= '1';
              LMDR1<='0'; LMDR0<='0';
              CE<='1'; R_W<='0';
                                                  decide <= '1';
              Count <= Count(6 downto 0)&'0';
              STATE <= OP7;
            else
              Count <= Count;
              STATE <= OP7;
            end if;
        elsif (Count = "00001000") then
            CE<='0'; clr_dec <= '1'; B_req <= '0';
            LMDR0<='1'; ld_dec <= '0';
            S9<='0'; S7<='1'; S8<='0';
            Ci<='0'; S5<='0'; S6<='0';
            Count <= "000"&Count(7 downto 3);
```

```
            STATE <= InstF;
        end if;
elsif (STATE=OP8) then            -- STOP PROCESS LOOP
  dbug_st_sig <= "1000";
     if (Count="00000001") then
            LMAR <= '1'; S2<='1'; S3<='1';
                  S10 <= '0'; B_req <= '0'; S11 <= '0';
                  Count <= Count(6 downto 0)&'0';
            STATE <= OP8;
        elsif (Count="00000010") then
            LMAR <= '0'; B_req <= '1';
            Count <= Count(6 downto 0)&'0';
            STATE <= OP8;
        elsif (Count = "00000100") then
            if B_grnt = '1' then
              LMAR<='0';
              LMDR1 <='0'; LMDR0 <= '0';  --place in MDR
              CE <= '1'; R_W<='0'; S8<='0'; S7<='1';
              Ci<='0'; S5 <= '0'; S6<='0';
              Count <= Count(6 downto 0)&'0';
              STATE <= OP8;
            else
              Count <= Count;
              STATE <= OP8;
            end if;
        elsif (Count="00001000") then
              CE <= '0';
              LMDR0 <= '1'; S8 <= '1'; S7 <= '0';
              Ci <= '1';   --subtract
              S2<='0'; S3<='0'; B_req <= '0';
              Count <= Count(6 downto 0)&'0';
              STATE <= OP8;
        elsif (Count="00010000") then
              if (Z='1') then
                    STOPLOOP <= '1';
               LPC<='1'; S4<='0'; S9<='0';
       end if;
       Count <= "0000"&Count(7 downto 4);
            STATE <= InstF;
        end if;
elsif (STATE=OP9) then
   dbug_st_sig <= "1001";
      if (Count = "00000001") then
        LMDR1 <= '1'; LMDR0 <= '1';
        S11 <= '0'; Ld_dec <= '1';
  -- extra logic added to get the output of IR0 directly to R3
  m5ctrl <='0';
        Count <= Count(6 downto 0)&'0';
        STATE <= OP9; B_req <= '0';
      elsif (Count = "00000010") then
        LMDR1 <= '0'; LMDR0 <= '1';
  S8 <= '0'; S7 <= '1'; Ci <= '0';
  S5 <= '0'; S6 <= '0'; S10 <='0';
  Ld_dec <= '0'; clr_dec <= '1';
  Count <= '0'&Count(7 downto 1);
  STATE <= InstF;
```

```
    end if;
elsif (STATE=OP10) then
 dbug_st_sig <= "1010";
  B_req <= '0';
    if (Count = "00000001") then
  -- added to get output from shifter
  m5ctrl <= '1';
        S0 <= '1'; S1 <= '1'; S10 <='0';  --Ld MDR with 0
        LMDR1 <= '1'; LMDR0 <= '0';
    ld_dec<='1'; S11 <= '0';
        Count <= Count(6 downto 0) &'0';
        STATE <= OP10;
    elsif (Count = "00000010") then
        LMDR1 <='0'; LMDR0 <= '1'; --ADD one, INC OP
        S8 <= '1'; S7 <='1'; Ci <='1';
        S5 <= '0'; S6 <='0';
        ld_dec <= '0'; clr_dec<='1';
        Count <= '0'& Count(7 downto 1);
        STATE <= InstF;
    end if;
 elsif (STATE=OP11) then
  dbug_st_sig <= "1011";
     B_req <= '0';
  if (Count = "00000001") then
   LR5 <= '1'; S11 <= '0'; --ld_dec <= '1';
 ld_dec <= '0';
   Count <= Count(6 downto 0)&'0';
   STATE <= OP11;
  elsif (Count = "00000010") then
   LR5 <= '0'; S10 <='1';
   --ld_dec <= '0'; clr_dec <= '1';
 -- we need R3_out to appear at MAR input
 -- so m5ctrl<= '1'; so that shifter output is selected and M2 output
 -- should be R3_out( 7 downto 0) so set proper values for s3 and s2 => "11"
 m5ctrl <= '1'; -- get output from shifter
 ld_dec <= '1'; clr_dec <= '0';
   S8 <= '1'; S7 <= '0'; Ci <= '0';
   S5 <= '0'; S6 <= '0';
 s3<= '1'; s2 <= '1';
 Count <= Count(6 downto 0)&'0';
      State <= OP11;
    elsif ( count = "00000100") then
   LMAR <= '1';
 ld_dec<='0';clr_dec <='1';
   Count <= "00"& Count(7 downto 2);
   STATE <= InstF;
  end if;
 elsif (STATE=OP12) then              -- sub rd, imm
    dbug_st_sig <= "1100";
      B_req <= '0';
  if (Count = "00000001") then
   LR5 <= '1'; S11 <= '0'; ld_dec <= '1';
   Count <= Count(6 downto 0)&'0';
   STATE <= OP12;
  elsif (Count = "00000010") then
   LR5 <= '0'; S10 <='1';
```

```
    ld_dec <= '0'; clr_dec <= '1';
    S8 <= '1'; S7 <= '0'; Ci <= '1';
    S5 <= '0'; S6 <= '0';
    Count <= '0'&Count(7 downto 1);
    STATE <= InstF;
   end if;

-- addition of and extra no -op state ----

elsif (STATE=OP13) then
     dbug_st_sig <= "1101";
       if (Count = "00000001") then
            LMAR<='1'; S2<='1'; S3<='1';
            S10 <= '0'; B_req <= '0';
            Count <= Count(6 downto 0)&'0';
            STATE <= OP6; S11 <= '0';
       elsif (Count = "00000010") then
            LMAR <='0';
            Count <= Count(6 downto 0)&'0';
            STATE <= OP13;
       elsif (Count = "00000100") then
          --  if B_grnt = '1' then
             LMAR<='0';
          --   LMDR1<='0'; LMDR0<='0';
          --    CE<='1'; R_W<='0';
          --     req_inpt <= '0'; req_otpt <= '1'; --signal output rdy
             Count <= Count(6 downto 0)&'0';
             STATE <= OP13;
           --else
           --  Count <= Count;
           --  STATE <= OP13;
          -- end if;
        elsif (Count = "00001000") then
           CE<='0';
            Count <= "000"&Count(7 downto 3);
            STATE <= InstF;
          -- end if;
       end if;

  else STATE <= RST;   --error, goto reset state
       end if;
    end if;
end process;
 dbug_st <=dbug_st_sig;
 count_out <= count;
end architecture;
```

Module Name : mempe.vhd

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
-- synopsys translate_off
```

```
Library XilinxCoreLib;

-- synopsys translate_on

entity mem_1 is
        port (data_bus : inout std_logic_vector(15 downto 0);
            Idata_bus : inout std_logic_vector(15 downto 0);
            clk, rst, CE: in std_logic;
            LMAR : in std_logic;
            LMDR1, LMDR0 : in std_logic;
            Addr : in std_logic_vector(7 downto 0);
            mux16 : in std_logic_vector(15 downto 0);
            Fin, Sel_Ibus : out std_logic;
            Maddr_out : out std_logic_vector(7 downto 0));
end entity;

architecture mem_arch of mem_1 is
------------------------------------------------------------------
-- This file was created by the Xilinx CORE Generator tool, and     --
-- is (c) Xilinx, Inc. 1998, 1999. No part of this file may be      --
-- transmitted to any third party (other than intended by Xilinx)   --
-- or used without a Xilinx programmable or hardwire device without --
-- Xilinx's prior written permission.                               --
------------------------------------------------------------------
component proc_imem
        port (
        addr: IN std_logic_VECTOR(7 downto 0);
        clk: IN std_logic;
        din: IN std_logic_VECTOR(15 downto 0);
        dout: OUT std_logic_VECTOR(15 downto 0);
        we: IN std_logic);
end component;


signal Mq_out : std_logic_vector(15 downto 0);
signal r_en : std_logic;
signal Mdata_out, Mdata_in : std_logic_vector(15 downto 0);
signal sel : std_logic_vector(1 downto 0);
signal q_out : std_logic_vector(7 downto 0);
signal data_in, data_out : std_logic_vector(15 downto 0);
signal Idata_out, Ddata_out : std_logic_vector(15 downto 0);
signal one, zero : std_logic;

Begin
one <= '1';
zero <= '0';

MARreg: process (clk, LMAR, rst)                                    --MAR register
        begin
          if rst = '1' then
                q_out <= (others=>'0');
          elsif (clk'event and clk='1') then
            if (LMAR='1') then
                q_out <= addr;
            else q_out <= q_out;
```

```
            end if;
          end if;
        end process;

Maddr_out <= q_out;
sel_Ibus <= '0' when (q_out = "00000000" or q_out= "00000001" or q_out="00000010") else
            '1';                                    --determine source of Instruction

FIN <= '1' when q_out = "00000000" else                 --get instr from PE Controller not IMEM
    '0';

data_bus <= Mq_out when (r_en = '0') else
        (others=>'Z');

-------------------------------------------------------
-- Component Instantiation
-------------------------------------------------------
Instr_mem : proc_imem port map (addr => q_out, clk => clk, din => data_in,
                                dout => Idata_out, we => ZERO);

Idata_bus <= Idata_out when (CE='0') else
             (others=>'0');


--MDR register
Mdata_in <= Data_bus when r_en='1' else
            (others=>'0');

r_en <= '0' when ((LMDR1='0')and(LMDR0='1')) else
        '1';

sel <= LMDR1 & LMDR0;

regout: process (clk, rst)
        begin
          if rst = '1' then
                Mq_out <= (others=>'0');
          elsif (clk'event and clk='0') then  -- at negative edge of the clock
                case (sel) is
                        when "00" => Mq_out <= Mdata_in;
                        when "01" => Mq_out <= Mq_out;
                        when "10" => Mq_out <= mux16;
                        when "11" => Mq_out <= Mq_out;
            when others => null;
          end case;
        end if;
  end process;

end architecture;
```

Module Name : proc_imem.xco (Xilinx IP Core)

```
------------------------------------------------------------------------------
--    This file is owned and controlled by Xilinx and must be used          --
--    solely for design, simulation, implementation and creation of         --
```

```
--     design files limited to Xilinx devices or technologies. Use           --
--     with non-Xilinx devices or technologies is expressly prohibited       --
--     and immediately terminates your license.                              --
--                                                                           --
--     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         --
--     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR               --
--     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION       --
--     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION           --
--     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS             --
--     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,               --
--     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      --
--     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              --
--     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               --
--     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        --
--     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       --
--     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       --
--     FOR A PARTICULAR PURPOSE.                                             --
--                                                                           --
--     Xilinx products are not intended for use in life support              --
--     appliances, devices, or systems. Use in such applications are         --
--     expressly prohibited.                                                 --
--                                                                           --
--     (c) Copyright 1995-2003 Xilinx, Inc.                                  --
--     All rights reserved.                                                  --
-------------------------------------------------------------------------------
-- You must compile the wrapper file proc_imem.vhd when simulating
-- the core, proc_imem. When compiling the wrapper file, be sure to
-- reference the XilinxCoreLib VHDL simulation library. For detailed
-- instructions, please refer to the "CORE Generator Guide".

-- The synopsys directives "translate_off/translate_on" specified
-- below are supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity
-- synthesis tools. Ensure they are correct for your synthesis tool(s).

-- synopsys translate_off
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

Library XilinxCoreLib;
ENTITY proc_imem IS
        port (
        addr: IN std_logic_VECTOR(7 downto 0);
        clk: IN std_logic;
        din: IN std_logic_VECTOR(15 downto 0);
        dout: OUT std_logic_VECTOR(15 downto 0);
        we: IN std_logic);
END proc_imem;

ARCHITECTURE proc_imem_a OF proc_imem IS

component wrapped_proc_imem
        port (
        addr: IN std_logic_VECTOR(7 downto 0);
        clk: IN std_logic;
        din: IN std_logic_VECTOR(15 downto 0);
        dout: OUT std_logic_VECTOR(15 downto 0);
```

```
        we: IN std_logic);
end component;

-- Configuration specification
        for all : wrapped_proc_imem use entity XilinxCoreLib.blkmemsp_v5_0(behavioral)
                generic map(
                        c_sinit_value => "0",
                        c_reg_inputs => 0,
                        c_yclk_is_rising => 1,
                        c_has_en => 0,
                        c_ysinit_is_high => 1,
                        c_ywe_is_high => 1,
                        c_ytop_addr => "1024",
                        c_yprimitive_type => "16kx1",
                        c_yhierarchy => "hierarchy1",
                        c_has_rdy => 0,
                        c_has_limit_data_pitch => 0,
                        c_write_mode => 0,
                        c_width => 16,
                        c_yuse_single_primitive => 0,
                        c_has_nd => 0,
                        c_enable_rlocs => 0,
                        c_has_we => 1,
                        c_has_rfd => 0,
                        c_has_din => 1,
                        c_ybottom_addr => "0",
                        c_pipe_stages => 0,
                        c_yen_is_high => 1,
                        c_depth => 256,
                        c_has_default_data => 0,
                        c_limit_data_pitch => 18,
                        c_has_sinit => 0,
                        c_mem_init_file => "proc_imem.mif",
                        c_default_data => "0",
                        c_ymake_bmm => 0,
                        c_addr_width => 8);
BEGIN

U0 : wrapped_proc_imem
                port map (
                        addr => addr,
                        clk => clk,
                        din => din,
                        dout => dout,
                        we => we);
END proc_imem_a;

-- synopsys translate_on
```

Module Name : mux16b.vhd

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

```
Entity mux16_4x1 is
        Port (line_out : out std_logic_vector(15 downto 0);
                Sel : in std_logic_vector(1 downto 0);
                line_in3,line_in2,line_in1,line_in0 : in std_logic_vector(15 downto 0));
end entity;

architecture mux16 of mux16_4x1 is

begin

it3: process(Sel,line_in3,line_in2,line_in1,line_in0)
  begin
        case (Sel) is
                when "00" => line_out <= line_in0;
                when "01" => line_out <= line_in1;
                when "10" => line_out <= line_in2;
                when "11" => line_out <= line_in3;
                when others =>line_out <= (others=>'X');
        end case;
   end process;

end architecture;
```

Module Name : mux16b5.vhd

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

Entity mux16_5x1 is
        Port (line_out : out std_logic_vector(15 downto 0);
            Sel : in std_logic_vector(2 downto 0);
            line_in4, line_in3, line_in2: in std_logic_vector(15 downto 0);
            line_in1, line_in0 : in std_logic_vector(15 downto 0));
end entity;

architecture mux165 of mux16_5x1 is

begin

it3: process(Sel,line_in4,line_in3,line_in2,line_in1,line_in0)
  begin
        case (Sel) is
                when "000" => line_out <= line_in0;
                when "001" => line_out <= line_in1;
                when "010" => line_out <= line_in2;
                when "011" => line_out <= line_in3;
                when "100" => line_out <= line_in4;
                when others => null;
        end case;
   end process;
```

```
end architecture;
```

Module Name : mux_2x1.vhd

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

Entity mux16bit_2x1 is
        Port (line_out : out std_logic_vector(15 downto 0);
                        Sel : in std_logic;
                        line_in1,line_in0 : in std_logic_vector(15 downto 0));
end entity;

architecture myarch of mux16bit_2x1 is

begin

muxproc: process(Sel,line_in1,line_in0)
  begin
        case Sel is
                when '0' => line_out <= line_in0;
                when '1' => line_out <= line_in1;
                when others =>NULL;--line_out <= (others=>'X');
        end case;
   end process;

end architecture;
```

Module Name : mux8b.vhd

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

Entity mux8_4x1 is
        Port (line_out : out std_logic_vector(7 downto 0);
                Sel : in std_logic_vector(1 downto 0);
                line_in3,line_in2,line_in1,line_in0 : in std_logic_vector(7 downto 0));
end entity;

architecture mux8 of mux8_4x1 is

begin

it3: process(Sel,line_in3,line_in2,line_in1,line_in0)
  begin
        case (Sel) is
                when "00" => line_out <= line_in0;
                when "01" => line_out <= line_in1;
                when "10" => line_out <= line_in2;
```

```
                when "11" => line_out <= line_in3;
                when others =>line_out <= (others=>'X');
        end case;
   end process;

end architecture;
```

Module Name : pc.vhd

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

Entity PC is
        Port (q_out : buffer std_logic_vector(7 downto 0);
                --q_out : inout std_logic_vector(7 downto 0);
                clk, clr : in std_logic;
                D : in std_logic_vector(7 downto 0);
                load, inc : in std_logic);
end entity;

architecture pc_arch of PC is

signal d_in : std_logic_vector(7 downto 0);

begin

it5: process (clk, clr)
        begin
          if (clr='1') then
                  q_out <= (others=>'0');
          elsif (clk'event and clk='1') then
                if ((inc='1') and (load='0')) then
                        q_out <= (q_out+1);
          elsif ((load='1') and (inc='0')) then
                        q_out <= D;
                else q_out <= q_out;
                end if;
          end if;
        end process;

end architecture;
```

Module Name : reg_bin.vhd

```
-- This Register isolates the Data bus from the Input Mux before the ALU
-- which prevents "X" and "Z"s from appearing on the mux output
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Reg_B_in is
port( din: in std_logic_vector(15 downto 0);  -- data from data_bus
```

```
 dout:out std_logic_vector(15 downto 0); -- register output
 clk: in std_logic;        -- clk
 rst: in std_logic;        -- Asynch Reset
 ctrlreg: in std_logic        -- Control signal
  );
end Reg_B_in;
architecture Behavioral of Reg_B_in is
begin
process(rst,clk)
begin
if rst = '1' then
 dout<=(others=>'0');
elsif(clk'event and clk='1') then
 case ctrlreg is
  when '0' => dout <=(others=>'0');
  when others => dout <= din;
 end case;
end if;
end process;
end Behavioral;
```

Module Name : regpe.vhd

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity REGS is
            port (q_out : buffer std_logic_vector(15 downto 0);
                        --q_out : inout std_logic_vector(15 downto 0);
                  clk, clr : in std_logic;
                  D : in std_logic_vector(15 downto 0);
                  Load : in std_logic);
End entity;

Architecture regs_arch of regs is

Begin

It: process(clk, clr)
            Begin
   if (clr='1') then
                        q_out <= (others=>'0');
             elsif (clk'event and clk='0') then
               if (load='1') then
                                    q_out <= D;
               else
                                    q_out <= q_out;
               end if;
            end if;
            end process;
```

```
end architecture;
```

Module Name : shifter_16.vhd

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Shifter_16 is
        port(ALU_out : in std_logic_vector(15 downto 0);
            Sel : in std_logic_vector(1 downto 0);
            Shf_out : out std_logic_vector(15 downto 0)) ;
End entity;

Architecture shift of shifter_16 is

begin

it2: process (ALU_out, Sel)
        begin
          case (Sel) is
                when "00" => Shf_out <= ALU_out;
                when "01" => Shf_out <= (ALU_out(14 downto 0) &'0');
                when "10" => Shf_out <= ('0'&ALU_out(15 downto 1));
                when "11" => Shf_out <= (others=>'0');
                when others => null;
          end case;
        end process;

end architecture;
```

Module Name : token_mapr.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Token_mapr is
   port (
      token_bus: inout STD_LOGIC_VECTOR (31 downto 0);
      --bus_req: buffer STD_LOGIC;
            bus_req: inout STD_LOGIC;
      clk : in std_logic;
      rst : in std_logic;
      bus_grnt: in STD_LOGIC;
      Avail3: in STD_LOGIC_VECTOR (4 downto 0);
      Avail4: in STD_LOGIC_VECTOR (4 downto 0);
      Avail2: in STD_LOGIC_VECTOR (4 downto 0);
            Avail5: in STD_LOGIC_VECTOR (4 downto 0);
            obstemp6_prtdbug,t6_prtdbug: out std_logic_vector(22 downto 0)
            --Pl_in_dbug :out std_logic_vector(6 downto 0);
            --tok_in_dbug : out std_logic_vector(16 downto 0)
   );
end Token_mapr;
```

```
architecture Token_mapr_arch of Token_mapr is

component PRT_Cntl
   port (
      Tokbus: inout STD_LOGIC_VECTOR (31 downto 0);
      clk : in std_logic;
      rst : in std_logic;
      tbus_grant: in STD_LOGIC;
      --tbus_req: buffer STD_LOGIC;
            tbus_req: inout STD_LOGIC;
      tok_in : out std_logic_vector(16 downto 0);
      Pl_in : out std_logic_vector(6 downto 0);
      Addr : out std_logic_vector(7 downto 0);
      clr : out std_logic;
      q2 : out std_logic;
      chip_on : out std_logic;
      nxt_token : in std_logic_vector(22 downto 0)
   );
end component;

component dy_load_bal_ckt
            port(   Clk: in std_logic;
                        Clear : in std_logic;
                        On1 : in std_logic;
                    Tok_in: in std_logic_vector(16 downto 0);
                        PL_in: in std_logic_vector(6 downto 0);
                        Aval0, Aval1, Aval2,Aval3,Aval4,Aval5,Aval6,Aval7 : in std_logic_vector(4 downto 0);
                        Addr: in std_logic_vector(7 downto 0);
                        OBUS: out std_logic_vector(22 downto 0);
                        Q2:  in std_logic;
                        obstemp6_dbug,t6_dbug:out std_logic_vector(22 downto 0));
end component;

signal prt_tok_in : std_logic_vector(16 downto 0);
signal prt_pl_in : std_logic_vector(6 downto 0);
signal prt_addr : std_logic_vector(7 downto 0);
signal prt_clr, prt_q2, en : std_logic;
signal prt_out : std_logic_vector(22 downto 0);
signal five1 : std_logic_vector(4 downto 0);

begin

  five1 <= "11111";

  C1: PRT_CNTL port map(Tokbus=> token_bus, clk => clk, rst => rst, tbus_grant=> bus_grnt,
              tbus_req=> bus_req, tok_in => prt_tok_in, Pl_in =>prt_pl_in,
              Addr =>prt_addr, clr =>prt_clr, q2 => prt_q2, chip_on => en,
              nxt_token => prt_out);

  M1: dy_load_bal_ckt port map (Clk => clk, Clear => prt_clr, On1 => en, Tok_in =>prt_tok_in,
                        PL_in => prt_pl_in, Aval0=> five1, Aval1=> Avail2, Aval2=> Avail3,
                        Aval3=> Avail4, Aval4=> Avail5, Aval5=> five1, Aval6=> five1,
                        Aval7=> five1, Addr=> prt_addr, OBUS=> prt_out, Q2=> prt_q2,
                        obstemp6_dbug =>obstemp6_prtdbug,t6_dbug=>t6_prtdbug);
end Token_mapr_arch;
```

Module Name : dy_load_bal_ckt.vhd

```
--      FILENAME : dlbc.v
--      MODULE   : dy_load_bal_ckt
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity dy_load_bal_ckt is
 port(   Clk: in std_logic;
   Clear : in std_logic;
  On1 : in std_logic;
        Tok_in: in std_logic_vector(16 downto 0);
  PL_in: in std_logic_vector(6 downto 0);
  Aval0, Aval1, Aval2,Aval3,Aval4,Aval5,Aval6,Aval7 : in std_logic_vector(4 downto 0);
  Addr: in std_logic_vector(7 downto 0);
  OBUS: out std_logic_vector(22 downto 0);
  Q2:  in std_logic;
  obstemp6_dbug,t6_dbug:out std_logic_vector(22 downto 0)
  );
End dy_load_bal_ckt;
Architecture mapr of dy_load_bal_ckt is
component mcntrlr
  port(start : buffer std_logic;
  c1,c2,c3,c4,c5,c6,c7,c8,c9 : out std_logic;
  q1, q2, q3: in std_logic;
  On1, clr : in std_logic;
  Clk: in std_logic);
End component;
component dec3x5
 port( do: out std_logic_vector(5 downto 1);
  s : in std_logic_vector(2 downto 0));
end component;
component map_Fifo
    port ( data_out : out std_logic_vector(16 downto 0);
   data_in: in std_logic_vector(16 downto 0);
   stack_full : inout std_logic;
   sigl : out std_logic;
   clk, rst : in std_logic;
   write_to_stack, read_from_stack: in std_logic);
end component;
component ic_net
 port( A1,A2,A3,A4,A5 : out std_logic_vector(5 downto 1);
  S1,S2,S3,S4,S5 : in std_logic_vector(7 downto 1);
  Aval0,Aval1,Aval2 : in std_logic_vector(5 downto 1);
  Aval3,Aval4,Aval5 : in std_logic_vector(5 downto 1);
  Aval6,Aval7 : in std_logic_vector(5 downto 1));
End component;
component register_R0
 port( outr0 : buffer std_logic_vector(16 downto 0);
  clk, clear : in std_logic;
  Prt_in : in std_logic_vector(16 downto 0);
  C2 : in std_logic);
```

```
End component;
component mux_2x1
 port( muxout : out std_logic;
  in1, in0 : in std_logic;
  sel : in std_logic);
end component;
component ram_unit
 port ( Ramout : out std_logic_vector(6 downto 0);
  Ramin : in std_logic_vector(6 downto 0);
  PN : in std_logic_vector(4 downto 0);
  C4, c9, Dec_in, clk : in std_logic);
End component;
component regA1_5
 port( out_reg : buffer std_logic_vector(4 downto 0);
  clk, clear : in std_logic;
  reg_in : in std_logic_vector(4 downto 0);
  c7 : in std_logic);
end component;
component reg_Pl
 port( out_pl : buffer std_logic_vector(6 downto 0);
  clk, clear : in std_logic;
  Pl_in : in std_logic_vector(6 downto 0);
  C5 : in std_logic);
End component;
component comparator
    port( a_lt_b: out std_logic;
     a_gte_b : out std_logic;
  a, b : in std_logic_vector(5 downto 1));
end component;
component regR1_4
 port( regout : buffer std_logic_vector(22 downto 0);
  clk, clear : in std_logic;
  regin : in std_logic_vector(22 downto 0);
  c5, c6, y : in std_logic);
end component;
component regR5
 port( regout : buffer std_logic_vector(22 downto 0);
  clk, clear : in std_logic;
  regin : in std_logic_vector(22 downto 0);
  c5,c6,y,f : in std_logic);
end component;
component regR6
 port( regout : buffer std_logic_vector(22 downto 0);
  clk, clear : in std_logic;
  regin : in std_logic_vector(22 downto 0);
  c8, c10, c11 : in std_logic);
end component;
component regR7
 port( regout : buffer std_logic_vector(22 downto 0);
  clk, clear : in std_logic;
  regin : in std_logic_vector(22 downto 0);
  c5, c6, y, F : in std_logic);
end component;
Constant one : std_logic := '1';
Constant zero: std_logic := '0';
Signal  fifo_out, OUT_R0: std_logic_vector(16 downto 0);
```

```
Signal dec_out: std_logic_vector(5 downto 1);
Signal PN, A1, A2, A3, A4, A5, OUT_A1, OUT_A2 : std_logic_vector(4 downto 0);
Signal OUT_A3, OUT_A4, OUT_A5: std_logic_vector(4 downto 0);
Signal PL_out1, PL_out2, PL_out3, PL_out4: std_logic_vector(6 downto 0);
Signal PL_out5,PL1, PL2, PL3, PL4, PL5: std_logic_vector(6 downto 0);
Signal ORC2_C7,q1,C1, C2, C3, C4, C5, C6, C7, C8, C9: std_logic;
Signal a, b, c, d, e, f, g, h, i, j, a_bar, b_bar, c_bar: std_logic;
signal d_bar, e_bar, f_bar, g_bar, h_bar, i_bar, j_bar: std_logic;
signal Y1, Y2, Y3, Y4, Y5, start, stack_full: std_logic;
signal F1, fifo_wr : std_logic;
signal t1,t2,t3,t4,t5,t6, t7 : std_logic_vector(22 downto 0);
signal OBUS_sig : std_logic_vector( 22 downto 0);
--signal OBStemp : std_logic_vector(22 downto 0);
-- trying to dbug the OBUStemp buffer problem
signal OBStemp1,OBStemp2,OBStemp3 : std_logic_vector(22 downto 0);
signal OBStemp4,OBStemp5,OBStemp6,OBStemp7 : std_logic_vector(22 downto 0);
signal OBStemp5_7 : std_logic_vector(22 downto 0);
--signal not_F : std_logic;
begin
--**** FIFO ****
FI_EN: process (tok_in)
 begin
  if tok_in = "0000000000000000" then
   fifo_wr <= '0';
        else
         fifo_wr <= '1';
        end if;
 end process;

f0:  map_FIFO port map(fifo_out, tok_in, stack_full, q1, CLK, CLEAR, fifo_wr, C1);
--**** REGISTER R0 ****
r0: register_R0 port map(OUT_R0, CLK, CLEAR, fifo_out, C1);
--**** DECODER ****
d0:  dec3x5 port map(dec_out, ADDR(2 downto 0));
--**** OR_(C2&C7) ****
orc2_c7 <= c2 or c7;

--**** MUX AFTER REG_R0 ****
mux_r0_0: mux_2x1 port map(PN(0), ADDR(3), OUT_R0(8), C7);
mux_r0_1: mux_2x1 port map(PN(1), ADDR(4), OUT_R0(9), C7);
mux_r0_2: mux_2x1 port map(PN(2), ADDR(5), OUT_R0(10), C7);
mux_r0_3: mux_2x1 port map(PN(3), ADDR(6), OUT_R0(11), C7);
mux_r0_4: mux_2x1 port map(PN(4), ADDR(7), OUT_R0(12), C7);
--**** RAM_UNITS 1_5 ****
ram0: ram_unit port map(PL_out1, PL_in, PN, C2, C7, dec_out(1), clk);
ram1: ram_unit port map(PL_out2, PL_in, PN, C2, C7, dec_out(2), clk);
ram2: ram_unit port map(PL_out3, PL_in, PN, C2, C7, dec_out(3), clk);
ram3: ram_unit port map(PL_out4, PL_in, PN, C2, C7, dec_out(4), clk);
ram4: ram_unit port map(PL_out5, PL_in, PN, C2, C7, dec_out(5), clk);
--**** REGISTER FOR LOADING PL FROM RAM ****
reg_PL0: reg_PL port map(PL1, CLK, CLEAR, PL_out1, C3);
reg_PL1: reg_PL port map(PL2, CLK, CLEAR, PL_out2, C3);
reg_PL2: reg_PL port map(PL3, CLK, CLEAR, PL_out3, C3);
reg_PL3: reg_PL port map(PL4, CLK, CLEAR, PL_out4, C3);
reg_PL4: reg_PL port map(PL5, CLK, CLEAR, PL_out5, C3);
--**** IC_NET(Nx5) ****
```

```
ic0: ic_net port map(A1, A2, A3, A4, A5, PL1, PL2, PL3, PL4, PL5, Aval0, Aval1, Aval2, Aval3, Aval4,
Aval5, Aval6, Aval7);
--**** DETERMINE WHETHER THERE IS A FAULT IN PL5 ****
faultdet: process (A1,A2,A3,A4,A5)
       begin
            if   ((A1="11111") and (A2="11111") and (A3="11111")
  and (A4="11111")and (A5="11111")) then
       F1<='1';
      else
      F1<='0';
      end if;
        End process;

--**** REGISTER FOR LOADING AVAILABILITIES ****
regA0: regA1_5 port map(OUT_A1, CLK, CLEAR, A1, C4);     --changed from c5
regA1: regA1_5 port map(OUT_A2, CLK, CLEAR, A2, C4);
regA2: regA1_5 port map(OUT_A3, CLK, CLEAR, A3, C4);
regA3: regA1_5 port map(OUT_A4, CLK, CLEAR, A4, C4);
regA4: regA1_5 port map(OUT_A5, CLK, CLEAR, A5, C4);
--**** COMPARATORS ****
com1:   comparator port map(a, a_bar, OUT_A1, OUT_A2);
com2: comparator port map(b, b_bar, OUT_A1, OUT_A3);
com3: comparator port map(c, c_bar, OUT_A2, OUT_A3);
com4: comparator port map(d, d_bar, OUT_A1, OUT_A4);
com5: comparator port map(e, e_bar, OUT_A2, OUT_A4);
com6: comparator port map(f, f_bar, OUT_A3, OUT_A4);
com7: comparator port map(g, g_bar, OUT_A1, OUT_A5);
com8: comparator port map(h, h_bar, OUT_A2, OUT_A5);
com9:  comparator port map(i, i_bar, OUT_A3, OUT_A5);
com10:  comparator port map(j, j_bar, OUT_A4, OUT_A5);

--**** AND GATES TO OBTAIN MOST AVAILABLE PROCESS ****
y1 <= a and b and d and g and c6;
y2 <= a_bar and c and e and h and c6;
y3 <= b_bar and c_bar and f and i and c6;
y4 <= d_bar and e_bar and f_bar and j and c6;
y5 <= g_bar and h_bar and i_bar and j_bar and c6;
--**** REGISTERS R1 THRU R7 ****
t1 <= (Out_R0(16 downto 14)&PN(4 downto 0)&PL1&OUT_R0(7 downto 0));
t2 <= (Out_R0(16 downto 14)&PN(4 downto 0)&PL2&OUT_R0(7 downto 0));
t3 <= (Out_R0(16 downto 14)&PN(4 downto 0)&PL3&OUT_R0(7 downto 0));
t4 <= (Out_R0(16 downto 14)&PN(4 downto 0)&PL4&OUT_R0(7 downto 0));
t5 <= (Out_R0(16 downto 14)&PN(4 downto 0)&PL5&OUT_R0(7 downto 0));
--t6 <= (Out_R0(16 downto 14)&OBStemp6(19 downto 8)&OUT_R0(7 downto 0));
t6 <= (Out_R0(16 downto 14)&OBuS_sig(19 downto 8)&OUT_R0(7 downto 0));
t7 <= (Out_R0(16 downto 14)&PN(4 downto 0)&"1110011"&OUT_R0(7 downto 0));

--OBUS <= OBStemp when (y1='1' or y2='1' or y3='1' or y4='1' or y5='1'
       --or c9='1') else
       --(others=>'0');
-- Debug signal added to view the contents on obstemp6
obstemp6_dbug<=OBStemp6;
t6_dbug<=t6;
OBUS_sig <= OBStemp1 when (y1='1')else
       OBStemp2 when (y2='1')else
    OBStemp3 when (y3='1')else
```

```
   OBStemp4 when (y4='1')else
   OBStemp6 when (c9='1')else
   OBStemp5_7 when (y5='1')else
  (others => '0');
OBStemp5_7 <= OBStemp5 when (F='0')
           else OBStemp7 ;
-- changes done for debugging  to include it in t6
obus <= obus_sig ;
regR1: regR1_4 port map(OBStemp1, CLK, CLEAR, t1, C3, C4, Y1);
RegR2: regR1_4 port map(OBStemp2, CLK, CLEAR, t2, C3, C4, Y2);
RegR3: regR1_4 port map(OBStemp3, CLK, CLEAR, t3, C3, C4, Y3);
regR4:  regR1_4 port map(OBStemp4, ClK, CLEAR, t4, C3, C4, Y4);
reR5: regR5 port map(OBStemp5, CLK, CLEAR, t5, C3, C4, Y5, F);
reR6: regR6 port map(OBStemp6, CLK, CLEAR, t6, C6, C8, C9);
reR7: regR7 port map(OBStemp7,CLK,CLEAR, t7, C3, C4, Y5, F);

--**** CONTROLLER ****
cntr0: mcntrlr port map(start, C1, C2, C3, C4, C5, C6, C7, C8, C9, q1, q2,
OUT_R0(13), ON1, CLEAR, CLK);
End architecture;
```

Module Name : comparator.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;


entity comparator is
           port(      a_lt_b: out std_logic;
                      a_gte_b : out std_logic;
                      a, b : in std_logic_vector(5 downto 1));
end comparator;

architecture  comp of comparator is
signal altb: std_logic;
begin
process (a,b) is
begin
if a<b then altb <='1';
else altb <= '0';
end if;
end process;
a_gte_b <= not altb;
a_lt_b <= altb;
end architecture;
```

Module Name : Dec3x5.vhd

```
--  FILENAME : dec3x5.v
--  MODULE   : dec3x5

--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity dec3x5 is
        port(   do: out std_logic_vector(5 downto 1);
                s : in std_logic_vector(2 downto 0));
end dec3x5;

architecture decs of dec3x5 is

-- Internal wire declarations
signal s0_bar, s1_bar, s2_bar: std_logic;

begin
-- Gate instantiations
   s0_bar <= not s(0);
   s1_bar <= not s(1);
   s2_bar <= not s(2);
   do(1) <= s2_bar and s1_bar and s0_bar;
   do(2) <= s2_bar and s1_bar and s(0);
   do(3) <= s2_bar and s(1) and s0_bar;
   do(4) <= s2_bar and s(1) and s(0);
   do(5) <= s(2) and s1_bar and s0_bar;

end architecture;
```

Module Name : ic_net.vhd

```
--  FILENAME : IC_NET.v
--  MODULE   : ic_net

--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity ic_net is
        port(   A1,A2,A3,A4,A5 : out std_logic_vector(5 downto 1);
                S1,S2,S3,S4,S5 : in std_logic_vector(7 downto 1);
                Aval0,Aval1,Aval2 : in std_logic_vector(5 downto 1);
                Aval3,Aval4,Aval5 : in std_logic_vector(5 downto 1);
                Aval6,Aval7 : in std_logic_vector(5 downto 1));
End ic_net;

Architecture icn of ic_net is

Begin
   The: process (S1, S2, S3, S4, S5, Aval0, Aval1, Aval2, Aval3,
         Aval4, Aval5, Aval6, Aval7)
      begin
          case S1 is
         when "0000001" => A1 <= Aval0;
```

```
   when "0000010" => A1 <= Aval1;
   when "0000011" => A1 <= Aval2;
   when "0000100" => A1 <= Aval3;
   when "0000101" => A1 <= Aval4;
   when "0000110" => A1 <= Aval5;
   when "0000111" => A1 <= Aval6;
   when "0001000" => A1 <= Aval7;
   when others => A1 <="11111";
    end case;

case S2 is
            when "0000001" => A2 <= Aval0;
            when "0000010" => A2 <= Aval1;
            when "0000011" => A2 <= Aval2;
            when "0000100" => A2 <= Aval3;
            when "0000101" => A2 <= Aval4;
            when "0000110" => A2 <= Aval5;
            when "0000111" => A2 <= Aval6;
            when "0001000" => A2 <= Aval7;
when others => A2 <= "11111";
end case;

case S3 is
when "0000001" => A3 <= Aval0;
            when "0000010" => A3 <= Aval1;
            when "0000011" => A3 <= Aval2;
            when "0000100" => A3 <= Aval3;
            when "0000101" => A3 <= Aval4;
            when "0000110" => A3 <= Aval5;
            when "0000111" => A3 <= Aval6;
 when "0001000" => A3 <= Aval7;
            when others => A3 <= "11111";
end case;

case S4 is
            when "0000001" => A4 <= Aval0;
            when "0000010" => A4 <= Aval1;
            when "0000011" => A4 <= Aval2;
            when "0000100" => A4 <= Aval3;
            when "0000101" => A4 <= Aval4;
            when "0000110" => A4 <= Aval5;
            when "0000111" => A4 <= Aval6;
            when "0001000" => A4 <= Aval7;
            when others => A4 <= "11111";
end case;

case S5 is
when "0000001" => A5 <= Aval0;
            when "0000010" => A5 <= Aval1;
            when "0000011" => A5 <= Aval2;
 when "0000100" => A5 <= Aval3;
            when "0000101" => A5 <= Aval4;
            when "0000110" => A5 <= Aval5;
            when "0000111" => A5 <= Aval6;
            when "0001000" => A5 <= Aval7;
            when others => A5 <= "11111";
```

```
        end case;

end process;

end architecture;
```

Module Name : mapfifo.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity MAP_Fifo is
   port (data_out : out std_logic_vector(16 downto 0);
         data_in: in std_logic_vector(16 downto 0);
         --stack_full : buffer std_logic;
         stack_full : inout std_logic;
         sigl : out std_logic;
         clk, rst : in std_logic;
         write_to_stack, read_from_stack: in std_logic);
end MAP_Fifo;

architecture fif1 of MAP_fifo is



component add_subber4
port (
         A: IN std_logic_VECTOR(3 downto 0);
         B: IN std_logic_VECTOR(3 downto 0);
         C_IN: IN std_logic;
         C_OUT: OUT std_logic;
         ADD_SUB: IN std_logic;
         Q_OUT: OUT std_logic_VECTOR(3 downto 0));
end component;


component add_subber5
port (
         A: IN std_logic_VECTOR(4 downto 0);
         B: IN std_logic_VECTOR(4 downto 0);
         C_IN: IN std_logic;
         C_OUT: OUT std_logic;
         ADD_SUB: IN std_logic;
         Q_OUT: OUT std_logic_VECTOR(4 downto 0));
end component;




signal stack_empty: std_logic;
```

```
signal read_ptr,write_ptr: std_logic_vector(3 downto 0);          -- Pointer for reading and writing
signal ptr_diff: std_logic_vector(4 downto 0);                              -- Distance between ptrs
type stkarray is array(15 downto 0) of std_logic_vector(16 downto 0);
signal     stack: stkarray;                                                 -- memory array
signal fourB1, rsum, wsum : std_logic_vector(3 downto 0);
signal valone, zero : std_logic;
signal psum_add, psum_sub, fiveB1 : std_logic_vector(4 downto 0);

begin

stack_empty <=  '1' when ptr_diff = "00000" else
                        '0';
stack_full <= '1' when ptr_diff = "10000" else
                    '0';
sigl <= not stack_empty;

-- begin data_transfer
datatrn: process (clk, rst)
variable i, j : integer;
begin
          if (rst='1') then
                    data_out <= (others=>'0');
          elsif (clk'event and clk='0') then

               case read_ptr is
                              when "0000" => i := 0;
                              when "0001" => i := 1;
                              when "0010" => i := 2;
                              when "0011" => i := 3;
                              when "0100" => i := 4;
                              when "0101" => i := 5;
                              when "0110" => i := 6;
                              when "0111" => i := 7;
                              when "1000" => i := 8;
                              when "1001" => i := 9;
                              when "1010" => i := 10;
                              when "1011" => i := 11;
                              when "1100" => i := 12;
                              when "1101" => i := 13;
                              when "1110" => i := 14;
                              when "1111" => i := 15;
                              when others => null;
          end case;
          case write_ptr is
                              when "0000" => j := 0;
                              when "0001" => j := 1;
                              when "0010" => j := 2;
                              when "0011" => j := 3;
                              when "0100" => j := 4;
                              when "0101" => j := 5;
                              when "0110" => j := 6;
                              when "0111" => j := 7;
                              when "1000" => j := 8;
                              when "1001" => j := 9;
                              when "1010" => j := 10;
                              when "1011" => j := 11;
```

```
                                when "1100" => j := 12;
                                when "1101" => j := 13;
                                when "1110" => j := 14;
                                when "1111" => j := 15;
                                when others => null;
            end case;
       if ((read_from_stack='1') and (write_to_stack='0') and (stack_empty='0')) then
                                data_out <= stack(i);
       elsif ((write_to_stack='1') and (read_from_stack='0') and (stack_full='0')) then
                                stack(j) <= data_in;
       elsif ((write_to_stack='1') and (read_from_stack='1') and (stack_empty='0') and
                                (stack_full='0')) then
                           stack(j) <= data_in;
                           data_out <= stack(i);
      end if;
              end if;
end process;

-----------------------------------------------------
-- Component Instantiation
-----------------------------------------------------
fourB1 <= "0001";
valone <= '1';
fiveB1 <= "00001";
zero <= '0';

rptr_add : add_subber4
                port map (A=>read_ptr, B =>fourB1, C_IN=>zero, C_OUT=>open,
                        ADD_SUB=>valone, Q_OUT=>rsum);

wptr_add : add_subber4
                port map (A=>write_ptr, B =>fourB1, C_IN=>zero, C_OUT=>open,
                        ADD_SUB=>valone, Q_OUT=>wsum);

ptr_add : add_subber5
               port map (A=>ptr_diff, B=>fiveB1, C_IN=>zero, C_OUT=>open,
                       ADD_SUB=>valone, Q_OUT=>psum_add);

ptr_sub : add_subber5
               port map (A=>ptr_diff, B=>fiveB1, C_IN=>zero, C_OUT=>open,
                        ADD_SUB=>zero, Q_OUT=>psum_sub);

unkn: process(clk, rst)
  begin
   if (rst='1') then
      read_ptr <= (others=>'0');
      write_ptr <= (others=>'0');
      ptr_diff <= (others=>'0');
    elsif (clk'event and clk='0') then
                        if ((write_to_stack='1') and (stack_full='0') and (read_from_stack='0')) then
                write_ptr <= wsum;                    --address for next clock edge
            ptr_diff <= psum_add;
    elsif ((write_to_stack='0') and (stack_empty='0') and (read_from_stack='1')) then
                read_ptr <= rsum;
                ptr_diff <= psum_sub;
    elsif ((write_to_stack='1') and (stack_empty='0') and (stack_full='0') and
```

```
                    (read_from_stack='1')) then
                read_ptr <= rsum;
                write_ptr <= wsum;
          ptr_diff <= ptr_diff;
     end if;
    end if;
end process;

end architecture;
```

Module Name : Mapcntlr.vhd

```
--  FILENAME : mapcntlr.vhd
--  MODULE   : mCntrlr

--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity mcntrlr is
          port(start : buffer std_logic;
                    c1,c2,c3,c4,c5,c6,c7,c8,c9 : out std_logic;
                    q1, q2, q3: in std_logic;
                    On1, clr : in std_logic;
                    Clk: in std_logic);
End mcntrlr;

Architecture mcont of mcntrlr is

signal     T, D : std_logic_vector(11 downto 1);
signal out1,out2: std_logic;
signal Din1, Din2: std_logic;

begin
-- Synchronous Sequential Process
-- Synchronous start circuit (negative edge triggered)
startckt: process (clk, clr)
          begin
            if (clr = '1') then
                    out1 <= '0';
                    out2 <= '0';
                elsif (clk'event and clk='0') then
                    out1 <= Din1;
                    out2 <= Din2;
                end if;
end process;

-- sequential controller flip flops (positive edge triggered)
contff: process (clk, clr)
          begin
            if (clr = '1') then
                    T <= (others=>'0');
                elsif (clk'event and clk='1') then
```

```
                    T <= D;
                End if;
End process;

-- Combinational Process
comb: process (T,out1,out2, q1, q2, q3, ON1, start)
        begin
           -- Generate 'start' signal
            Din1<= ON1;
            Din2 <= out1;
        start  <= out1 and (not out2);

         -- Generate Flip Flop Next State Equations
            d(1) <= (start or (T(9) and (not q2)) or T(8) or T(11));
            D(2) <= (T(1) and q1);
            D(3) <= T(2);
            D(4) <= (T(3) and (not q3));
            D(5) <= T(4) and (not q2);
            D(6) <= T(5);
            D(7) <= T(6);
            D(8) <= T(7);
            D(9) <= (T(1) and (not q1)) or (T(9) and q2) or (T(4) and q2);
            D(10) <= T(3) and q3;
            D(11) <= T(10);

       -- Generate Control Equations
            c1 <= T(2);
            c2 <= T(4);
            c3 <= T(5);
            c4 <= T(6);
            c5 <= T(7);
            c6 <= T(8);
            c7 <= T(9);
            c8 <= T(10);
            c9 <= T(11);

end process;

end architecture;
```

Module Name : Ram_unit.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;


entity ram_unit is
        port (  Ramout : out std_logic_vector(6 downto 0);
                Ramin : in std_logic_vector(6 downto 0);
                PN : in std_logic_vector(4 downto 0);
                C4, c9, Dec_in, clk : in std_logic);
End ram_unit;
```

```
Architecture rams of ram_unit is

component mapram2
        port (
        a: IN std_logic_VECTOR(4 downto 0);
        clk: IN std_logic;
        d: IN std_logic_VECTOR(6 downto 0);
        we: IN std_logic;
        spo: OUT std_logic_VECTOR(6 downto 0));
end component;

component mux_2x1
        port(   muxout : out std_logic;
                in1, in0 : in std_logic;
                sel : in std_logic);
end component;

Signal ram_in: std_logic_vector(6 downto 0);
Signal INEN: std_logic;
Signal MUX_OUT, INN: std_logic;
signal one : std_logic;

begin
   one <= '1';
   -- Instantiate 2x1 mux for CE of Ram
   m0: mux_2x1 port map(MUX_OUT, DEC_IN, one, INEN);

  -- and gate for RW
   INN <= Dec_in and c9;
   INEN <= c4 or c9;

 -- Bi-directional Buffers

   ram_in <= ramin when INEN = '1' else (others=>'Z');
   --ramout <= ram_out when INEN = '1' else (others=>'Z');

-- Instantiate 32x7 Ram
ram1 : mapram2 port map
        (a =>PN, CLK => clk, D =>ram_in, WE =>INN, spo => ramout);

end architecture;
```

Module Name : Mapram.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE STD.TEXTIO.ALL;

entity mapram2 is
   port (a: in std_logic_vector(4 downto 0);
        clk: in std_logic;
        d: in std_logic_vector(6 downto 0);
```

```
            we: in std_logic;
        spo: out std_logic_vector(6 downto 0));
end mapram2;

architecture ram_body of mapram2 is

constant deep: integer := 31;
type fifo_array is array(deep downto 0) of std_logic_vector(6 downto 0);
signal mem: fifo_array;

signal addr_int: integer range 0 to 31;

begin
addr_int <= conv_integer(a);

process (clk)
begin
 if clk'event and clk = '1' then
  if we = '1' then
   mem(addr_int) <= d;
  end if;
 end if;
end process;
spo <= mem(addr_int);
end ram_body;
```

Module Name : reg_pl.vhd

```
--  FILENAME : reg_PL.v
--  MODULE   : reg_PL

--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity reg_Pl is
        port(   out_pl : buffer std_logic_vector(6 downto 0);
                clk, clear : in std_logic;
                Pl_in : in std_logic_vector(6 downto 0);
                C5 : in std_logic);
End reg_pl;

Architecture regp of reg_pl is

begin

Regit: process(clk, clear)
        Begin
                If clear = '1' then
                        out_pl <= (others=>'0');
                elsif (clk'event and clk='0') then
                  if c5 = '1' then
                        out_pl <= pl_in;
```

```
                    else
                            out_pl <= out_pl;
                    end if;
                  end if;
          end process;

end architecture;
```

Module Name : prt_cntl.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity PRT_Cntl is
   port (
      Tokbus: inout STD_LOGIC_VECTOR (31 downto 0);
      clk : in std_logic;
      rst : in std_logic;
      tbus_grant: in STD_LOGIC;
      --tbus_req: buffer STD_LOGIC;
            tbus_req: inout STD_LOGIC;
      tok_in : out std_logic_vector(16 downto 0);
      Pl_in : out std_logic_vector(6 downto 0);
      Addr : out std_logic_vector(7 downto 0);
      clr : out std_logic;
      q2 : out std_logic;
      chip_on : out std_logic;
      nxt_token : in std_logic_vector(22 downto 0)
   );
end PRT_Cntl;

architecture PRT_Cntl_arch of PRT_Cntl is

component mapbuf
          port (
          din: IN std_logic_VECTOR(24 downto 0);
          clk: IN std_logic;
          wr_en: IN std_logic;
          rd_en: IN std_logic;
          ainit: IN std_logic;
          dout: OUT std_logic_VECTOR(24 downto 0);
          full: OUT std_logic;
          empty: OUT std_logic);
end component;

signal w_en : std_logic;
signal tline_in, tline_out : std_logic_vector(31 downto 0);
type optype is (reset, Ld_Ram, Operate, Hold, Normal);
signal op : optype;
signal tok_buf, tok_temp, bufout : std_logic_vector(24 downto 0);
constant lcl_addr : std_logic_vector(6 downto 0) := "0000001";
constant Load_R : std_logic_vector(5 downto 0) := "111010";
type jbuf is array(1 downto 0) of std_logic_vector(15 downto 0);
signal join_buf : jbuf;
signal join0_avl, join1_avl : std_logic;
signal buf_num, full, empty1, we, re : std_logic;
```

```
signal out_buf : std_logic_vector(31 downto 0);

begin

  tline_in <= Tokbus when w_en = '0' else (others=>'0');
  Tokbus <= tline_out when w_en = '1' else (others=>'Z');
  chip_on <= '1';
  w_en <= '1' when (tbus_grant='1' and tbus_req='1') else
        '0';

Inbuf : mapbuf port map (din => tok_buf,clk => clk,wr_en => we,rd_en => re,
                         ainit => rst,dout => bufout,full => full,
                         empty => empty1);

  iptproc: process (clk, tline_in, rst, full)
     begin
        if rst = '1' then
            we <= '0';
            tok_buf <= (others=>'0');
        elsif (clk'event and clk='1') then
          if tline_in(30 downto 24) = lcl_addr then
            tok_buf <= tline_in(31)&tline_in(23 downto 0);
            if full = '0' then
              we <= '1';                                  --place Token in buffer
            else
              we <= '0';
            end if;
         else
                we <= '0';
                tok_buf <= (others=>'0');
                  end if;
           end if;
         end process;

  control: process (rst, clk, op, empty1)
          variable cont, ld_delay, del2, inpt_delay, inpt_del2 : boolean;
          begin
            if rst ='1' then op <= reset;
            elsif (clk'event and clk='1') then

              case (op) is
                  when reset =>   clr <= '1';
                                  q2  <= '0'; re <= '0';
                                  cont := false;
                                  ld_delay := false;
                                  del2 := false; inpt_del2 := false;
                                  inpt_delay := false;
                                  tok_temp <= (others=>'0');
                                  tbus_req <= '0';
                                  buf_num <= '0';
                                  out_buf <= (others=> '0');
                                  tok_in <= (others=>'0');
                                  Pl_in <= (others=>'0');
                                  Addr <= (others=> '0');
                                  join_buf(0) <= (others=>'0');
                                  join_buf(1) <= (others=>'0');
```

```
                join0_avl <= '1';
                join1_avl <= '1';
                op <= Operate;

when Operate => clr <= '0';
        q2 <= '0';
        tok_in <= (others=>'0');
              Pl_in <= (others=>'0');
              Addr <= (others=> '0');
        if (tbus_grant = '1' and tbus_req = '1') then
          tline_out <= out_buf;
          out_buf <= (others=> '0');
          re <= '0';
          op <= Operate;
        elsif (empty1 = '0' and inpt_delay = false) then
          re <= '1';                                --get token from queue
          inpt_delay := true;
          op <= Operate;
        elsif (inpt_delay = true and inpt_del2 = false) then
          re <= '0';
          inpt_del2 := true;
          op <= Operate;
        elsif (inpt_del2 = true) then --parse read token
          if (bufout(24 downto 19)) = Load_R then
                tok_temp <= bufout;         --Load RAM token
                inpt_delay := false;
                op <= Ld_Ram;
          elsif bufout(24) = '1' then           --hold token
              tok_temp <= bufout;
              inpt_delay := false;
              op <= Hold;
          else
              tok_temp <= bufout;
              inpt_delay := false;
              op <= Normal;                             --normal token
          end if;
          inpt_delay := false;
          inpt_del2 := false;
        else
                re <= '0';
                op <= Operate;                          --wait for token
        end if;

when Ld_Ram =>          clr <= '0';
                q2  <= '1';
                re <= '0';
                if (ld_delay = false and del2 = false) then
                  op <= Ld_Ram;
                  ld_delay := true;
                elsif (ld_delay = true and del2 = false) then
                  op <= Ld_Ram;
                  del2 := true;
                else
                  Pl_in <= tok_temp(14 downto 8);
                  Addr <= tok_temp(7 downto 0);
                  tok_in <= (others=>'0');
```

```
                    op <= Operate;
                    del2 := false;
                    ld_delay := false;
                    --tok_temp <= (others=>'0');
                  end if;

when Normal =>  clr <= '0';
                  q2 <= '0';
                  re <= '0';
                  tok_in(13) <= tok_temp(24);
                  tok_in(12 downto 8) <= tok_temp(20 downto 16);
                  tok_in(7 downto 0) <= tok_temp (7 downto 0);
                  tok_in(16 downto 14) <= tok_temp(23 downto 21);
                  Pl_in <= (others=>'0');
                Addr <= (others=> '0');
                --tok_buf <= (others=>'0');
                op <= Operate;

when Hold =>   clr <= '0';
               q2 <= '0'; re <= '0';
               Pl_in <= (others=>'0');
               Addr <= (others=> '0');
               if (cont = true) then                    --send 2nd token in join
                  tok_in(16 downto 14) <= "000";
                  tok_in(13) <= '1';
                  if buf_num = '0' then
                    tok_in(12 downto 0) <= join_buf(0)(12 downto 0);
                    join0_avl <= '1';
                    join_buf(0) <= (others=>'0');
                  else
                    tok_in(12 downto 0) <= join_buf(1)(12 downto 0);
                    join1_avl <= '1';
                    join_buf(1) <= (others=>'0');
                  end if;
                  cont := false;
                  op <= Operate;
               elsif tok_temp(23 downto 16) = join_buf(0)(15 downto 8) then
                                              --send first token
                  tok_in(13) <= '0';
                       tok_in(12 downto 8) <= tok_temp(20 downto 16);
                       tok_in(7 downto 0) <= tok_temp(7 downto 0);
                       tok_in(16 downto 14) <= tok_temp(23 downto 21);
                       cont := true;
                       buf_num <= '0';
                       --tok_buf <= (others=>'0');
                       op <= Hold;
               elsif tok_temp(23 downto 16) = join_buf(1)(15 downto 8) then
                                              --send first token
                  tok_in(13) <= '0';
                       tok_in(12 downto 8) <= tok_temp(20 downto 16);
                       tok_in(7 downto 0) <= tok_temp(7 downto 0);
                       tok_in(16 downto 14) <= tok_temp(23 downto 21);
                       cont := true;
                       buf_num <= '1';
                       --tok_buf <= (others=>'0');
                       op <= Hold;
```

```
                                elsif (cont = false and join0_avl = '1') then   --wait for other token
                                   join_buf(0)(15 downto 8) <= tok_temp(23 downto 16);
                                   join_buf(0)(7 downto 0) <= tok_temp(7 downto 0);
                                   join0_avl <= '0';
                                   --tok_buf <= (others=>'0');
                                   op <= Operate;
                                elsif (cont = false and join1_avl = '1') then   --wait for other token
                                   join_buf(1)(15 downto 8) <= tok_temp(23 downto 16);
                                   join_buf(1)(7 downto 0) <= tok_temp(7 downto 0);
                                   join1_avl <= '0';
                                   --tok_buf <= (others=>'0');
                                   op <= Operate;
                                else                                            --join buffer overflow
                                        --tok_buf <= (others=>'0');
                                        op <= Operate;
                                end if;

                        end case;
                        if out_buf /= "00000000000000000000000000000000" then
                        tbus_req <= '1';
                    else
                         tbus_req <= '0';
                    end if;
                    if nxt_token /= "00000000000000000000" then
                        out_buf(31) <= '0';
                        out_buf(30 downto 24) <= nxt_token(14 downto 8);
                        out_buf(23 downto 21) <= nxt_token(22 downto 20);
                        out_buf(20 downto 16) <= nxt_token(19 downto 15);


                                                out_buf(7 downto 0) <= nxt_token(7 downto 0);

                                        out_buf(15 downto 8) <= "00000000";

                    end if;
                end if;
            end process;

end PRT_Cntl_arch;
```

Appendix B: Application 1: Acyclic Process Flow Graph Model

Sets of Table Load, Table Input and Load PRT tokens to be fed in the LUT are as follows:

For CE0:

Process Number	Table Load	Table Input	Load PRT
P1	83f80003	83f04430	81d0030c
P2	83f80017	83F08800	81D00314
P3	83F80024	83F0CA00	81D0031B
P6	83F80232	83F18E00	81D00334
P7	83F80039	83F1C000	81D0033C

For CE1:

Process Number	Table Load	Table Input	Load PRT
P1	82f80003	82f04430	81D0020B
P2	82f80117	82F08800	81D00213
P3	82F80024	82F0CA00	81D0021C
P6	82F80232	82F10E00	81D00233
P7	82F80039	82F1C000	81D0023D

For CE2:

Process Number	Table Load	Table Input	Load PRT
P5	84F80104	84F14C00	81D0042B

For CE3:

Process Number	Table Load	Table Input	Load PRT
P4	85F80104	85F10C00	81D00523

Contents of Instruction Memory:

Process P1:

Instruction Memory Address	Data	Operation
3	0300	INPUT MEM[R3]
4	AF00	INC R3
5	0300	INPUT MEM[R3]
6	AF00	INC R3
7	0300	INPUT MEM[R3]

8	AF00	INC R3
9	0300	INPUT MEM[R3]
A	AF00	INC R3
B	0300	INPUT MEM[R3]
C	AF00	INC R3
D	0300	INPUT MEM[R3]
E	AF00	INC R3
F	0300	INPUT MEM[R3]
10	AF00	INC R3
11	0300	INPUT MEM[R3]
12	AF00	INC R3
13	0300	INPUT MEM[R3]
14	AF00	INC R3
15	0300	INPUT MEM[R3]
16	3000	JUMP #0

Process P2:

Instruction Memory Address	Data	Operation
17	7300	LD R0, MEM[R3]
18	AF00	INC R3
19	1000	ADD R0, MEM[R3]
1A	AF00	INC R3
1B	1000	ADD R0, MEM[R3]
1C	AF00	INC R3
1D	1000	ADD R0, MEM[R3]
1E	AF00	INC R3
1F	1000	ADD R0, MEM[R3]
20	BF06	ADD R3, #6
21	2000	STORE MEM[R3], R0
22	6300	OUTPUT MEM[R3]
23	3000	JMP #0

Process P3:

Instruction Memory Address	Data	Operation
24	BF05	ADD R3, #5
25	7300	LD R0, MEM[R3]
26	AF00	INC R3
27	1000	ADD MEM[R3], R0
28	AF00	INC R3
29	1000	ADD MEM[R3], R0
2A	AF00	INC R3
2B	1000	ADD MEM[R3], R0
2C	AF00	INC R3

2D	1000	ADD MEM[R3], R0
2E	BF02	ADD R3, #5
2F	2000	STORE MEM[R3], R0
30	6300	OUTPUT MEM[R3]
31	3000	JMP #0

Process P6:

Instruction Memory Address	Data	Operation
32	BF0A	ADD R3, #10
33	7300	LD R0, MEM[R3]
34	AF00	INC R3
35	5000	SUB MEM[R3], R0
36	AF00	INC R3
37	2000	STORE MEM[R3], R0
38	3000	JMP #0

Process P7:

Instruction Memory Address	Data	Operation
39	BF0C	ADD R3, #12
3A	6300	OUTPUT MEM[R3]
3B	3000	JMP #0

Process P4: Multiplication

Instruction Memory Address	Data	Operation
04	000A	OFFSET ADDITION
05	0002	MULTIPLICAND VAL

Process P5: Division

Instruction Memory Address	Data	Operation
04	000B	OFFSET ADDITION
05	0002	DIVISOR VAL

Contents of the shared data memory:

For the copy 1 of application, the data stored in the data memory is decimal ‘2’ from location x”03” to x”0C”.The data location after addition of first five numbers is stored in location x”0D” and similarly the result of the addition of the next five numbers is stored at x”0E”.

The value changes after the division and multiplication process takes place. The table shows the values before and after the division and multiplication operation. The final result decimal ‘15’is stored at location x”0F”.

Address Location	Result before Multiplication and division (decimal)	Result after multiplication and division (decimal)
0D	10	20
0E	10	5

For the copy 2 of application, the data stored in the data memory is decimal ‘2’ from location x”11” to x”1A”.The data location after addition of first five numbers is stored in location x”1B” and similarly the result of the addition of the next five numbers is stored at x”1C”.

The value changes after the division and multiplication process takes place. The table shows the values before and after the division and multiplication operation. The final result decimal ‘15’ is stored at location x”1D”.

Address Location	Result before Multiplication and division (decimal)	Result after multiplication and division (decimal)
1B	10	20
1C	10	5

Application 2: Cyclic Process Flow Graph Model

Sets of Table Load, Table Input and Load PRT tokens to be fed in the LUT are as follows:

For CE0:

Process Number	Table Load	Table Input	Load PRT
P1	83f80003	83f04440	81d0030c
P2	83f8010D	83F08600	81D00314
P3	83F80014	83F0C406	81D0031C
P4	83f8011B	83f10A00	81D00324
P5	83F80023	83F14806	81D0032C
P6	83F8022A	83F18000	81D00334

For CE1:

Process Number	Table Load	Table Input	Load PRT
P1	82F80003	82F04440	81d0020B
P2	82F8010D	82F08600	81D00213
P3	82F80014	82F0C406	81D0021B
P4	82Ff8011B	82F10A00	81D00223
P5	82F80023	82F14806	81D0022B
P6	82F8022A	82F18000	81D00233

Contents of Instruction Memory:

Process P1:

Instruction Memory Address	Data	Operation
3	0300	INPUT MEM[R3]
4	AF00	INC R3
5	0300	INPUT MEM[R3]
6	AF00	INC R3
7	0300	INPUT MEM[R3]
8	AF00	INC R3
9	0300	INPUT MEM[R3]
A	AF00	INC R3
B	0300	INPUT MEM[R3]
C	3000	JUMP #0

Process P2:

Instruction Memory Address	Data	Operation
D	7300	LD R0, MEM[R3]
E	BF02	ADD R3, #2
F	1000	ADD MEM[R3], R0
10	Cf02	SUB R3, #2
11	2000	STORE MEM[R3], R0
12	6300	OUTPUT MEM[R3]
13	3000	JMP #0

Process P3:

Instruction Memory Address	Data	Operation
14	BF04	ADD R3, #4
15	7300	LD R0, MEM[R3]
16	CF04	SUB R3, #4
17	8000	IS R0= MEM[R3]
18	2000	STORE MEM[R3], R0
19	6300	OUTPUT MEM[R3]
1A	3000	JMP #0

Process P4:

Instruction Memory Address	Data	Operation
1b	BF01	ADD R3, #1
1C	7300	LD R0, MEM[R3]
1D	Af00	INC R3
1E	5000	SUB R0,MEM[R3]
1F	CF01	SUB R3, #1
20	2000	STORE MEM[R3], R0
21	6300	OUTPUT MEM[R3]
22	3000	JMP #0

Process P5:

Instruction Memory Address	Data	Operation
23	BF03	ADD R3, #3
24	7300	LD R0, MEM[R3]
25	CF02	SUB R3,#2
26	8000	IS R0= MEM[R3]

27	2000	STORE MEM[R3], R0
28	6300	OUTPUT MEM[R3]
29	3000	JMP #0

Process P6:

Instruction Memory Address	Data	Operation
2A	7300	LD R0, MEM[R3]
2B	6300	OUPUT MEM[R3]
2C	AF00	INCR R3
2D	6300	OUPUT MEM[R3]
2E	3000	JMP #0

Application 3: Proving the concept of Multiple Forking for the HDCA

Sets of Table Load, Table Input and Load PRT tokens to be fed in the LUT are as follows:

For CE0

Process Number	Table Load	Table Input	Load PRT
P1	83f80003	83f04430	81d0030c
P2	83f8000F	83F08E00	81D00314
P3	83F80016	83F0C850	81D0031C
P4	83F80118	83F11000	81D00324
P5	83F80120	83F15000	81d0032C
P8	83F80328	83F20C00	81D00344
P6	83F80230	83F18000	81D00334

For CE1

Process Number	Table Load	Table Input	Load PRT
P1	82f80003	82f04430	81d0020B
P2	82f8000F	82F08E00	81D00213
P3	82F80016	82F0C850	81D0021C
P4	82F80118	82F11000	81D00223
P5	82F80120	82F15000	81d0022B
P8	82F80328	82F20C00	81D00243
P6	82F80230	82F18000	81D00234

For Multiplier CE

Process Number	Table Load	Table Input	Load PRT
P7	85f80104	85F1CC00	81d0053C

Process P1:

Instruction Memory Address	Data	Operation
3	0300	INPUT MEM[R3]
4	AF00	INC R3
5	0300	INPUT MEM[R3]
6	AF00	INC R3
7	0300	INPUT MEM[R3]
8	AF00	INC R3
9	0300	INPUT MEM[R3]
A	AF00	INC R3
B	0300	INPUT MEM[R3]
C	AF00	INC R3
D	0300	INPUT MEM[R3]
E	3000	JUMP #0

Process P2:

Instruction Memory Address	Data	Operation
F	7300	LD R0, MEM[R3]
10	AF00	INC R3
11	1000	ADD R0, MEM[R3]
12	BF06	ADD R3, #6
13	2000	STORE MEM[R3], R0
14	6300	OUTPUT MEM[R3]
15	3000	JMP #0

Process P3:

Instruction Memory Address	Data	Operation
16	D300	DELAY
17	3000	JMP #0

Process P4:

Instruction Memory Address	Data	Operation
18	BF02	ADD R3, #2
19	7300	LD R0, MEM[R3]
1A	AF00	INC R3
1B	1000	ADD MEM[R3], R0
1C	BF0E	ADD R3,#14
1D	2000	STORE MEM[R3], R0
1E	6300	OUTPUT MEM[R3]
1F	3000	JMP #0

Process P5:

Instruction Memory Address	Data	Operation
20	BF04	ADD R3, #4
21	7300	LD R0, MEM[R3]
22	AF00	INC R3
23	1000	ADD MEM[R3], R0
24	BF16	ADD R3, #22
25	2000	STORE MEM[R3], R0
26	6300	OUTPUT MEM[R3]
27	3000	JMP #0

Process P8:

Instruction Memory Address	Data	Operation
28	BF11	ADD R3, #17
29	7300	LD R0, MEM[R3]
2A	BF0A	ADD R3, #10
2B	5000	SUB MEM[R3], R0
2C	BF0A	ADD R3, #10
2D	2000	STORE MEM[R3], R0
2E	6300	OUTPUT MEM[R3]
2F	3000	JMP #0

Process P6:

Instruction Memory Address	Data	Operation
30	BF07	ADD R3, #7

31	7300	LD R0, MEM[R3]
32	BF1E	ADD R3, #30
33	1000	ADD MEM[R3], R0
34	BF14	ADD R3, #20
35	2000	STORE MEM[R3], R0
36	6300	OUTPUT MEM[R3]
37	3000	JMP #0

Process P7:

Contents of Instruction Memory for Multiplier CE

Instruction Memory Address	**Data**	**Operation**
04	0007	OFFSET ADDITION
05	0004	MULTIPLICAND VALUE

One command token was entered for the test bench and its value was x"01010003"

Final Results in the Shared Data Memory is "16" at location "60".

References

1. George Broomell and J. Robert Heath, "Classification Categories and Historical Development of Circuit Switching Topologies", *ACM Computing Surveys*, Vol.15, No.2, pp. 95-133, June 1983.
2. J. Robert Heath, Paul Maxwell, Andrew Tan, and Chameera Fernando, "Modeling, Design, and Experimental Verification of Major Functional Units of a Scalable Run-Time Reconfigurable and Dynamic Hybrid Data/Command Driven Single-Chip Multiprocessor Architecture and Development and Testing of a First-Phase Prototype", Private Communication, 2002.
3. J. Robert Heath, George Broomell, Andrew Hurt, Jim Cochran, Liem Le, "A Dynamic Pipeline Computer Architecture for Data Driven Systems", *Research Project Report*, Contract No. DASG60-79-C-0052, University of Kentucky Research Foundation, Lexington, KY 40506, Feb., 1982
4. Abd- El-Barr and El- Rewini, *Computer Design and Architecture*, draft manuscript, John Wiley 2004. http://engr.smu.edu/~rewini/8380/
5. Xiaohui Zhao, "Hardware Description Language Simulation and Experimental Hardware Prototype Validation of a First Phase Prototype of a Hybrid/Data Command Driven Architecture",*Masters Project*,University of Kentucky, Lexington,KY,Feb. 2003.
6. Venugopal Duvvuri, "Design, Development, and Simulation/Experimental Validation of a Crossbar Interconnection Network for a Single – Chip Shared Memory Multiprocessor Architecture", *Masters Project*, University of Kentucky, June 2002.
7. Sridhar Hegde, " Functional Enhancement and Applications Development for a Hybrid Heterogenous Single-Chip Multiprocessor Architecture", *Masters Thesis*, University of Kentucky, Lexington, KY, December 2004.
8. http://www.ee.uwa.edu.au/~maceyb/aca319-2002/handouts/Section4.pdf
9. http://www3.informatik.uni-erlangen.de/Lehre/RA/SS2001/Skript/05a-interconn1.pdf
10. http://www.cosc.brocku.ca/Offerings/3P93/notes/4-Interconnect.doc
11. www.comp.nus.edu.sg/~cs4231/lec/l01.pdf
12. B.Monien, R.Feldmann, R.Klasing, R.Luling, "Parallel Architectures: Design and Efficient Use." Department of Computer Science, University of Paderborn, Germany.

13. www.cs.ucsd.edu/classes/fa01/cs260
14. www.**cs**.colostate.edu/~**cs575**dl/lects/lec3.ppt
15. Paul Maxwell, "Design Enhancement, Synthesis and Field Programmable Gate Array Post Implementation Simulation Verification of a Hybrid Data/Command Driven Architecture.", *Masters Project*,Department of Electrical Engineering, University of Kentucky, Lexington KY, 2001.
16. J.R.Heath, S.Ramamoorthy, C.E.Stroud and A.Hurt, " Modeling, Design and Performance Analysis of a Parallel Hybrid Data/Command Driven Architecture System and its Scalable Dynamic Load Balancing Circuit", *IEEE Trans. On Circuits and Systems, II: Analog and Digital Signal Processing*, Vol. 44, No.1, pp. 22 -40, Januar, 1997.
17. J.R. Heath and B. Sivanesa, "Development, Analysis, and Verification of a Parallel Hybrid Data-flow Computer Architectural Framework and Associated Load Balancing Strategies and Algorithms via Parallel Simulation", *SIMULATION*, Vol. 69, No. 1, pp. 7-25, July, 1997.
18. Xiaohui Zhao, J. Robert Heath, Paul Maxwell, Andrew Tan, and Chameera Fernando, "Development and First-Phase Experimental Prototype Validation of a Single- Chip Hybrid and Reconfigurable Multiprocessor Signal Processor System", *Proceedings of the 2004 IEEE Southeastern Symposium on System Theory,* Atlanta, GA, 5pps, March 14-16, 2004.

Vita

Author's Name – Kanchan P.Bhide
Birthplace - Mumbai, India
Birthdate - September 14, 1978

Education

Bachelor of Science in Electrical Engineering
Sardar Patel College of Engineering,
University of Mumbai, India
June - 2000

Honors, Awards and Activities

Recipient of BIACS (**B**lueGrass **I**ndo **A**merican **C**ivic **S**ociety) Shashi Sathaye Memorial Scholarship for excellence in academics (2002) at University of Kentucky.