Performance Modeling of Single Processor and Multi-Processor Computer Architectures

by

Hormazd P. Commissariat

Thesis submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

_____

Dr. F. G. Gray, Chairman

_____ _____

Dr. J. R. Armstrong Dr. N.J.Davis, IV

June 1995

Blacksburg, Virginia

Performance Modeling of Single Processor and Multi-Processor Computer Architectures

by

Hormazd. P. Commissariat

Dr. F. G. Gray

Electrical Engineering

(ABSTRACT)

Determining the optimum computer architecture configuration for a specific application or a generic algorithm is a difficult task. The complexity involved in today's computer architectures and systems makes it more difficult and expensive to easily and economically implement and test full functional prototypes of computer architectures. High level VHDL performance modeling of architectures is an efficient way to rapidly prototype and evaluate computer architectures.

Determining the architecture configuration is fixed, one would like to know the tolerance and expected performance of individual/critical components and also what would be the best way to map the software tasks onto the processor(s). Trade-offs and engineering compromises can be analyzed and the effects of certain component failures and communication bottle-necks can be studied.

A part of the research work done for the RASSP (Rapid Prototyping of Application Specific Signal Processors) project funded by Department of Defense contracts is documented in this thesis. The architectures modeled include a single-processor, single-global-bus system; a four processor, single-global-bus system; a four processor, multiple-local-bus, single-global-bus system; and finally, a four processor multiple-local-bus system interconnected by a crossbar interconnection switch. The

hardware models used are mostly legacy/inherited models from an earlier project and they were upgraded, modified and customized to suit the current research needs and requirements. The software tasks that are run on the processors are pieces of the signal and image processing algorithm run on the Synthetic Aperture Radar (SAR).

The communication between components/devices is achieved in the form of tokens which are record structures. The output is a trace file which tracks the passage of the tokens through various components of the architecture. The output trace file is post-processed to obtain activity plots and latency plots for individual components of the architecture.

To my mom and dad

Acknowledgments

Numerous people were responsible for the success of this research work. I take this opportunity to thank them.

I would like to thank my advisor **Dr.F.Gail Gray** for giving me the opportunity to work on this interesting research topic. Moreover his constant encouragement, advice and support were invaluable to me.

I would like to thank my co-advisor **Dr.J.R.Armstrong**, for his valuable ideas and suggestions.

I appreciate the efforts of **Dr.N.J.Davis, IV** for consenting to be on my graduate committee on a very short notice and for reviewing my thesis.

Special thanks to **Dr. Geoffrey Frank** of Research Triangle Institute for his invaluable ideas, suggestions, encouragement and motivation, without which the work documented in this thesis would not have been completed.

A big thank-you to **Todd Carpenter**, **John Shackelton**, and **Fred Rose** of Honeywell Inc. for answering all my questions about their model library and providing support.

Thanks are due to **Mr. Bud Clark** of Research Triangle Institute.

Thanks are also due to my **colleagues and team members** on the RASSP (Rapid Prototyping of Application Specific Signal Processors) project.

I would like to express my gratitude to the **students of the Computer Research Laboratory** at Virginia Tech, for their company, help and friendship.

This thesis is dedicated to my **parents** to whom I will forever be grateful for everything in my life.

Last but not least, I would like to thank **God** for his divine help, support and guidance.

Table of Contents

Chapter 3. System Development and Results

List of Figures

Figures in Chapter 1

Figures in Chapter 2

Figures in Chapter 3

Chapter 1. Introduction

1.1 Motivation

Determining the optimum computer architecture configuration for a specific application or a generic algorithm is a difficult task. The complexity involved in today's computer architectures and systems makes it more and more difficult to easily and economically implement and test full functional prototypes of computer architectures. High level VHDL performance modeling of architectures is a more efficient to rapidly compare architectures.

1.2 Task Description

The research work done as part of the RASSP (Rapid Prototyping of Application Specific Signal Processors) project funded by the ARPA/TriServices contract is documented in this thesis. The objective is to develop an abstract, high-level VHDL library of computer architecture hardware models and to use them to do performance analysis of single processor and multi-processor architectures. The factors we are interested in monitoring

are communication bottle-necks and overheads. Also we would like to determine optimum selection of device parameters for a particular task mapping and architecture schematic. Figure 1 shows the organization of the RASSP project at Virginia Tech.

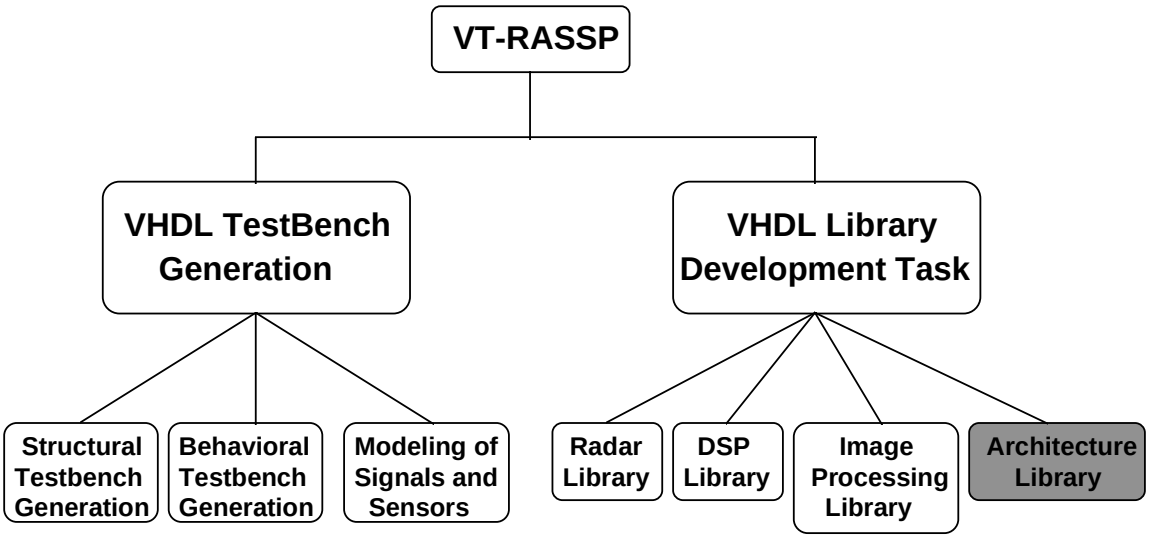

Fig. 1.1: The Virginia Tech RASSP effort.

1.3 Research Approach

The primitives used in the development of the architecture models were derived from an earlier research project. They were modified and customized to meet the current research needs. Components, tools and devices that were not pre-existent were custom designed to be compatible with the existing library models.

An approach for the development and design of performance models of architectures follows:

- Perform a cursory study of all the components and resources that are available in the existing libraries. The purpose of this step is to become familiar with the available components and resources and to identify strengths and weaknesses in preparation for customization.

- Translate the task description to hardware and software requirements specifications.
- Based on the specifications, salvage as many reusable components from the existing libraries as possible.
- Modify the library components if necessary to meet the current research specifications.
- Develop new components and tools to supplement the library.
- Build architectures from the components using a tentative but reasonable choice for device parameters and characteristics.
- Run simulations and analyze results.
- If the results are unsatisfactory due to resource contention or speed considerations then modify device characteristics as needed.

This thesis uses the approach described above to construct architectural schematics in an evolutionary manner, starting with a simple single-processor architecture and then successively upgrading the architecture by adding more complexity, components and processors.

1.4 Thesis Organization:

Chapter 2, "Background" gives an introduction to the concept of performance modeling using VHDL. The performance statistics are defined and a section on reusability of VHDL models is included.

Chapter 3, "System Development and Results" demonstrates the methodology developed to use the model library. The development of the various architectures and their performance analysis is explained.

Chapter 4, "Conclusions and Future Work" is the concluding chapter of the thesis and it explains the contributions made to the state of the art by the work documented in this thesis. The section on future work suggests areas that could be explored for further refinements and additions to the present work.

Chapter 2. Background

2.1 Why VHDL ?

The following is abridged from reference [1].

"**Public Availability:** VHDL (VHSIC Hardware Description Language) was developed under a Government contract and is now an IEEE standard.

Wide Range of Descriptive Capability: VHDL supports behavioral descriptions of hardware from the digital system (e.g., box) level to the gate level.

Design Methodology and Design Technology Support: VHDL was designed to support many different design methodologies (e.g., top-down versus library-based) and design technologies (e.g., synchronous versus asynchronous, or PLA versus random logic).

Technology and Process Independence: By this it is meant that VHDL does not have embedded within it an understanding of particular technologies or processes. However, such information can be written using VHDL."

Modifiable Code: The advantage of using VHDL over some other ready made high level modeling tool is that users can alter and customize the VHDL code for the component primitives as well as the architecture more effectively and to a greater depth of fidelity than they could if they were using a ready made CAD tool.

2.2 Performance Modeling using VHDL:

VHDL is a rich language which supports a variety of model types a few of which are shown in figure 2.1. So the modeller can use VHDL to develop models from the system block level down to the gate level. Within the complete spectrum of design hierarchy, shown in figure 2.2, performance modeling is more suited to modeling of the system behavior. At this level of abstraction the design is in the block-diagram stage with blocks representing system functions. Performance modeling is used at this stage to do analysis of the system by checking system timing, determining component throughput requirements, component utilization, analyzing trade offs, etc.

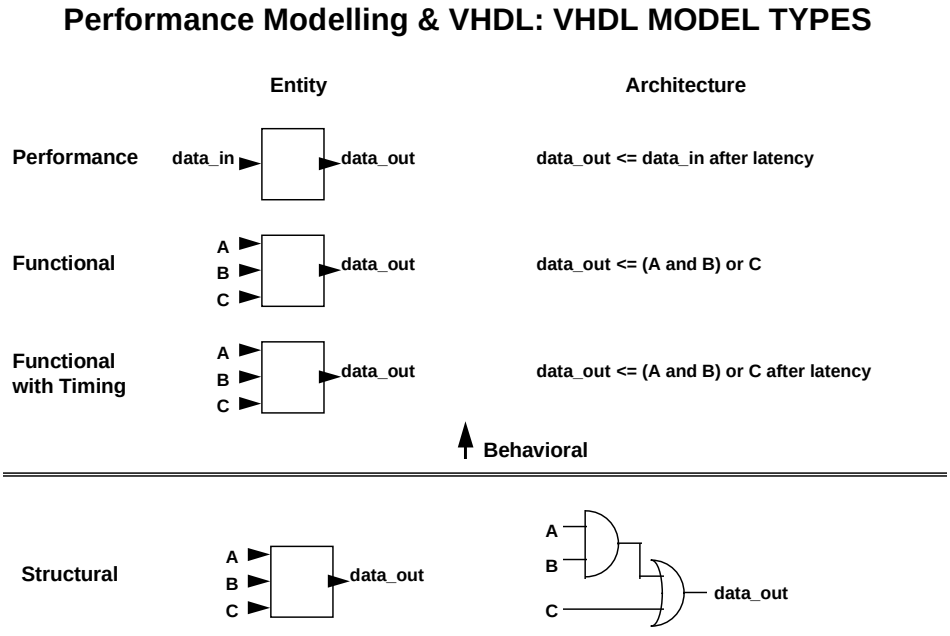

Figure 2.1 : VHDL Model Types [2]

The results of the performance modeling determine the architecture and configuration of the system. The functionality of the components within the architecture first enter into the design procedure at the Hardware Schematic Capture stage. The model at this stage can be a high-level behavioral model representing the entire device or the system may be represented as a structural model of interconnected sub-systems.

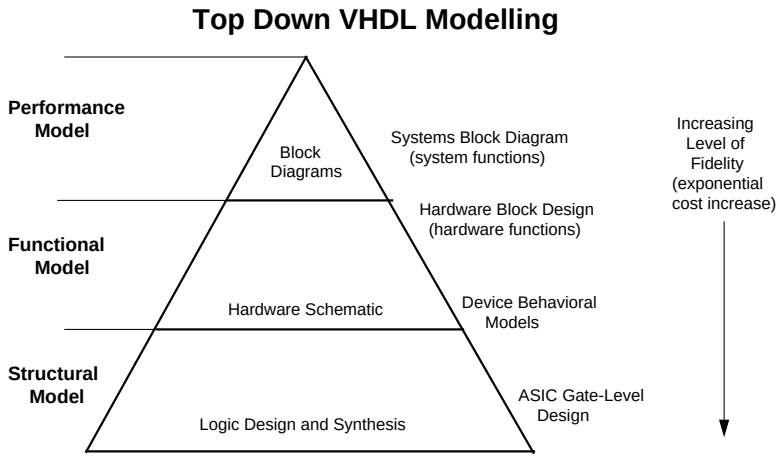

Figure 2.2 Design Hierarchy / Abstraction Levels [2]

The last stage of the design hierarchy is the ASIC (Application Specific Integrated Circuit) Gate-Level stage. At this stage, the design is mostly in the form of a synthesizeable structural model.

As we progress down the design hierarchy or design abstraction level, the fidelity, size, cost and effort required to model the system increases exponentially. But this is to be expected as we approach fabrication of the final design.

2.3 The Performance Statistics [2]

This section describes the performance data produced by the simulation and how it is collected. There are three basic statistics: latency, utilization, and throughput.

Performance studies and architectural trade-off analyses are interpreted with respect to these metrics.

Latency - The time it takes for a token to get from point A to point B.

Utilization - Time busy / total simulation time

Throughput - Data processed / time period

2.3.1 Latency

This attribute is the most difficult to obtain from the simulation. This is because latency can be specified in so many different fashions and over a wide range of locations. For instance, the driving requirement might be that the arrival of a radar pulse results in the firing of a certain task on a particular processor. In terms of the simulation, a single token representing that radar pulse must be generated, and the time of its creation recorded. That single event might trigger hundreds or thousands of other events in the simulation, which might happen to be concurrently handling other events. Once the effects of that initial token propagate through the system and reach the processor, that time must be recorded. If every event were recorded, the simulator performance would degrade due to memory and file I/O limitations. Means must be identified to track only particular types of information through the system. This situation is even more complex when a single event results in multiple events. A single file is kept containing information such as token id, source node, intermediate stops, final destination, and times associated with each. A post-processor is written to extract the desired latency values.

2.3.2 Utilization

A procedure is added to every architecture to monitor utilization. It is essentially a counter which keeps track of the amount of time the element is busy. At the end of simulation, each module outputs this information. A post-processor can sum it to any level within the hierarchy to provide a conglomerate utilization number. Such post-processing can even be augmented by such parameters as size, weight and power, to give a utilization per dollar, or throughput per area measure, or whatever measure is deemed appropriate and meaningful by the system architects.

2.3.3 Throughput

Though this statistic is not tracked in this work, throughput can be handled very much like utilization. Each module can keep track of the amount of data processed. A post-processor can be devised which will handle multi-level calculation of throughput.

To monitor statistics other than the ones above, one can add such a function to the Stat_Pack package in the GEN library.

2.4 Reusability

Since many of the models used in the development of the architectures were obtained from a previous research project, a brief section on reusability is appropriate.

2.4.1 The importance of reusability:

Research is an ongoing process; the quest for more knowledge never ceases. Modern research would not have reached the stage at which it is today if researchers were unable to make use or ***reuse*** the results and the outputs of past research. "Reinventing the wheel" from scratch was not a top priority in this work; but given the wheel and a few other

parts, a "car" or a "cart" was built and in the process this work has added value to the pre-existing models, and has thereby contributed effectively to state of the art in this field.

2.4.2 Ensuring Reusability of VHDL models:

Further reusability of the component models as well as architectural configurations is ensured in the following ways:

- **Use of Generics:** For a model to be easily and readily reused, the important parameters of the model should be easily and readily modifiable. This can be achieved by using generics to give values to important device and system parameters, so that they can be modified easily at the component mapping stage. Moreover generics allow most of the behavioral details within the components to be variable. For example in the present library of components, most communications and device behavioral parameters are generics.
- **Modular Design and Implementation:** A modular design is one in which the complete design is composed of independent but interconnected blocks or modules. It's quite possible that another design in a different context or research problem may not require all of the components used in the predecessor design. By using structural models even at the high-level performance level stage, just the individual components which are required in the successor design can be easily obtained from the predecessor design. It simplifies the salvage and reuse process. In the case of the present library, a striking example of the application of this principle is the crossbar interconnection device which is structurally composed of individual crossblocks. The designer can use as many crossblocks as needed for a particular application. Thus the very same design of the crossbar can be reused by reconfiguring it appropriately. A disadvantage of using structural models is that structural models

have signals that have to be port-mapped. Signal processing increases simulation time relative to a behavioral model.

- **Ease of Extendibility:** The models should be designed in a way that makes their later modification and customization an easily achievable goal. This may require that certain presently unused features be preserved. Also, the design should not be heavily constrained in order to accommodate future modifications. For example, in the present design the basic token structure, which is a very fundamental signal, has a few fields which are not used in the present application but are preserved so as to avoid modifying all the components of the library if the token structure needs to be changed in the future.
- **Good Documentation and Coding Style:** As in case of most programming languages good documentation and coding style improves the readability of the VHDL code and can go a long way in facilitating the reusability of the model as well as the individual parts of the model. For example self-explanatory choice of variable names improves readability of the code and makes it easy to reuse the model.

The steps suggested above can go a long way in making models reusable and thereby decreasing the model design time

Chapter 3. System Development and Results

3.1 The System Development Flowgraph

This section describes the use of high level performance models to evaluate computer architectures. The steps recommended to be followed for the system design/development process are as follows *(refer to figure 3.1)* :

1. The first step would be the formulation of the specifications. The problem definition or task description must be translated into hardware and software specifications.
2. Given the software specifications create a high level behavioral model of the application software/algorithm. This step is optional but will help in understanding the application algorithm and will facilitate the mapping of tasks onto the hardware/processors.
3. To begin with select a tentative but reasonable choice of the architecture schematic and the device parameters. Select the required hardware components from the model library and interconnect them (by structural port mapping) to form the architecture.

4. Make a tentative but reasonable partitioning of the application software, and the tasks to be mapped onto the hardware/processors. Step 2 can provide valuable information for performing the partitioning by providing insight into inherent parallelism within the algorithm. The mapping information is used to write the application software architecture's VHDL code file.
5. Run simulations and analyze the results to identify changes required (if any) in the schematics or the parameters. Make changes to the architecture and component parameters and repeat steps 3 through 5 until the architecture and device parameters are satisfactory.
6. Freeze the schematic, device parameters and the task partitioning, and move to the next lower level of the design process involving increasing details.

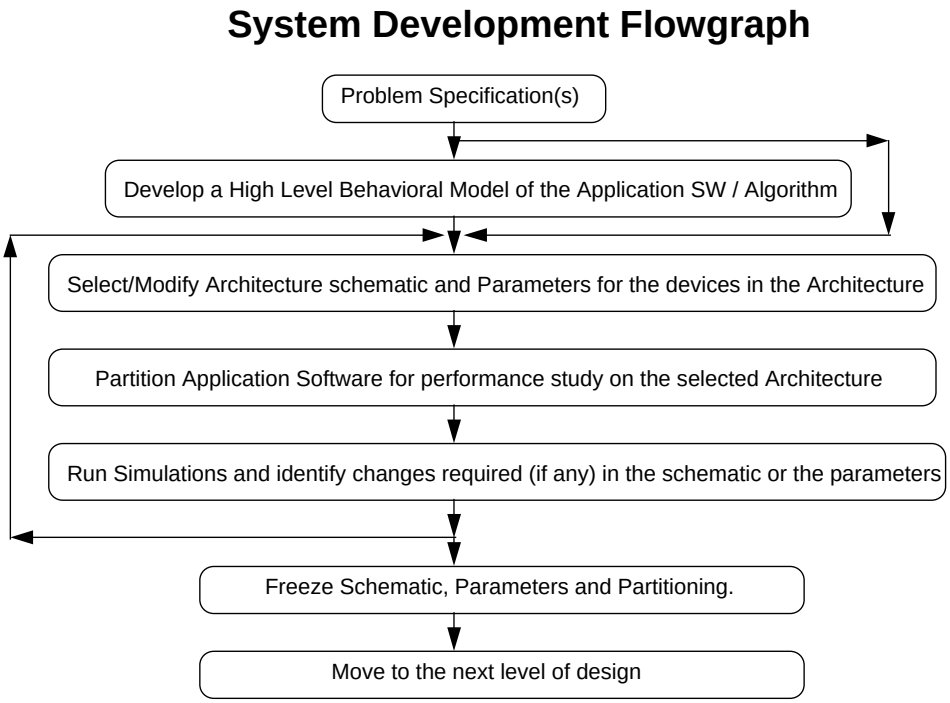

Figure 3.1: The System Development Flowgraph

As far as the applicability of the above flowgraph is concerned, the work documented in the present thesis does not address the issues of the development of higher level model behavioral model of the application software, nor does it explain the partitioning of the application software. These were performed by other persons working on the project. This thesis concentrates on steps 3 and 5 shown in the flowchart.

3.2 The Architectures

With reference to the system development flowgraph, the selection of the architecture schematics is the third step in the design process. This section considers four different system architectures: a single processor architecture, a four processor single global bus architecture, a four processor multiple local bus single global bus architecture and finally a four processor multiple local bus with crossbar architecture. The development of each of these architectures is explained and their performance compared.

3.2.1 The Single Processor Architecture

3.2.1.1 The Architecture Schematic

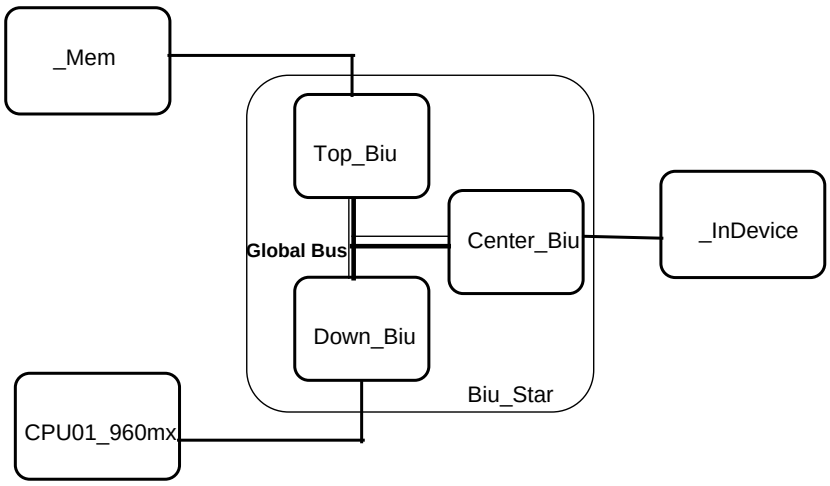

Figure 3.2: The Schematic diagram for the Single Processor architecture

Figure 3.2 shows the simplest of the architectures. This architecture is the basis for the other architectures down the line. It forms the basic processing node for the multiprocessor architectures. It consists of an input device which models the radar, the processing element which in the present case is the 960mx CPU, and the local memory of the CPU. All three components are interconnected to the bus using the bus interface units (BIUs). The star connection of the three BIUs is made into a structural (derivative) device and is called the BIU_STAR.

3.2.1.2 The Tasks Mapping

The task mapping was aided by using CAD tools such as Cadence SPW™ (Signal Processing WorkSystem). The higher level behavioral model also provided valuable insight into the software partitioning and recognition of inherent parallelism in the software tasks, especially in the case of the multi-processor architectures. The software tasks partitioning as well as the development of the behavioral model for the software were performed by other persons working on the project. Since the task mapping process is not a direct concern of this work it is not explained in great detail.

The simulation begins with the radar (which is modeled by the indevice) sending a pulse (termed Pole). The indevice writes the pulse data into the local memory of the CPU by performing a DMA(direct memory access) on the local memory. After the transfer of the pulse data is complete, the memory interrupts the processor which then "wakes up" and begins its processing. The tasks performed by the CPU (CPU01_960mx) are pieces of the Synthetic Aperture Radar algorithm. The various tasks are named Pole Read, FIR processing, Weight Read, Complex Vector Multiply & zero padding, Pulse Write, Pulse Read, and Bin Write. The sequence of simulation steps are tabulated in the following table:

No.	Step Name	Token Type	Msg Size	Msg Value XX- Don't Care	Source	Destination
1.	Pole Pulse	Write	32 Kbytes	XX	_Indevice	_Mem
2.	Pole Ack.	Interrupt	1 Kbyte	Event_1	_Mem	_CPU_960mx
3.	Pole Read	Read	1 Kbyte	32 Kbyte	_CPU_960mx	_Mem
4.	Pole Read Ack	Data	32 Kbytes	XX	_Mem	_CPU_960mx
5a.	Even FIR	(15 MPY-ADD) this is a processing step done on CPU_960mx				
5b.	Odd FIR	(15 MPY-ADD) this is a processing step done on CPU_960mx				
5c.	Form Complex	(12 FLADD) this is a processing step done on CPU_960mx				
6.	Weight Read	Read	1 Kbyte	48 Kbytes	_CPU_960mx	_Mem
7.	Weight Rd. Ack	Data	48 Kbytes	XX	_Mem	_CPU_960mx
8a.	Complex VPM	(6 Complex MPY = 24 MPY-ADDs) this is done by CPU_960mx				
8b.	Odd FIR	(8 FLADD Times) this is done by CPU_960mx				
9.	Pulse Write	Write	64 Kbytes	XX	_CPU_960mx	_Mem
10.	Pulse WriteAck	Ack	1 Kbytes	XX	_Mem	_CPU_960mx
11.	Pulse Read	Read	1 Kbytes	64 Kbytes	_CPU_960mx	_Mem
12.	Pulse Rd. Ack	Data	64 Kbytes	XX	_Mem	_CPU_960mx
13.	Bin Write	Write	128Kbytes	XX	_CPU_960mx	_Mem
14.	Bin Write Ack	Ack	1 Kbytes	XX	_Mem	_CPU_960mx

Fig: 3.3: Sequence of Simulation Steps for the single processor model

3.2.1.3 Results

Two simulations runs were performed. In the first simulation only one cycle of the radar pulse is considered. In the second simulation as many radar pulses as can be handled by the system are considered. There is a limit on how many pulses can be packed into one simulation cycle. This is determined by communication bottle-necks and processing speed. If too many radar pulses are considered within one simulation cycle then the bus interface units will time-out waiting for the bus to become idle and the radar pulse tokens will be lost.

The output trace file is postprocessed to obtain activity plots which are shown in *figures. 3.4 and 3.6* and latency plots which are shown in *figures 3.5 and 3.7*. The activity plots show periods of activity for the various devices in the architecture. It makes use of the utilization performance statistic to form the plots. Refer back to the earlier section 3.4 for more details on this. The average utilization value for each component is displayed on the right hand side of the plot. For example in the activity plot of figure 3.5 the CPU shows the highest utilization of 75 %. As can be seen in the activity plot shown in figure 3.4, in the case of the single simulation cycle, only one radar pulse is considered (starting at time = 0 ns) and all the processing gets completed by 13826 ns. Figure 3.6 shows that 4 radar pulses were sent during the simulation time of 53 ms, and the last pulse gets approximately half processed, as the simulation is halted after 53 ms.

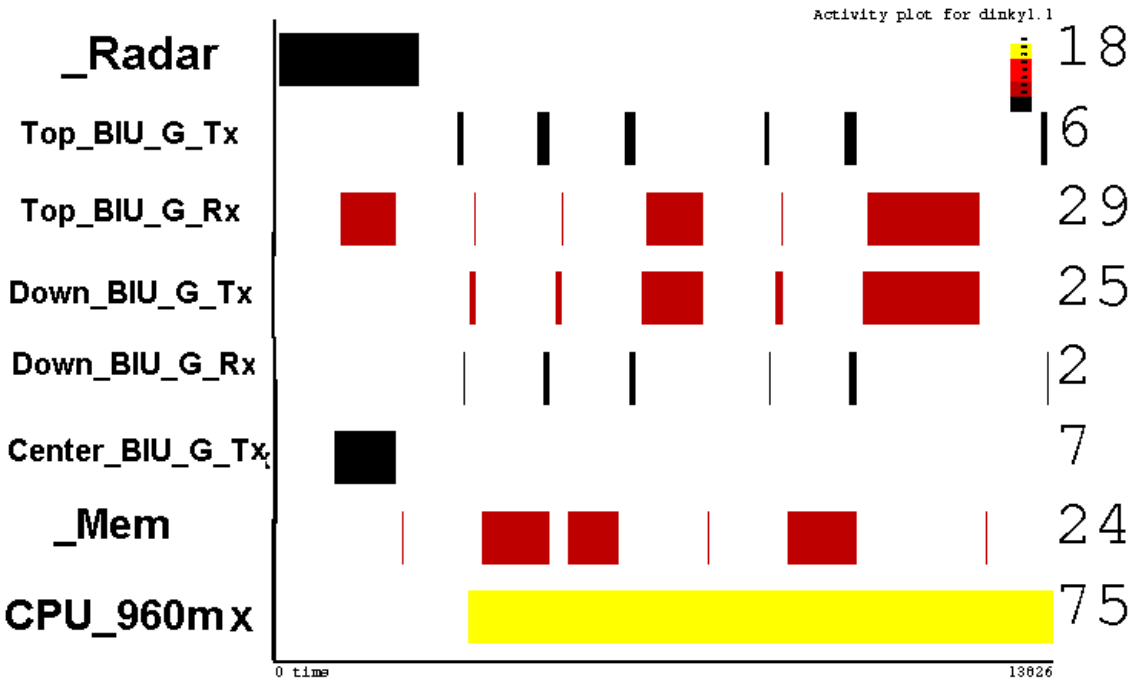

Figure 3.4: Activity Plot for one cycle of simulation of the single processor model.

The latency plot shows the latency distributions of the various devices in the architecture. The slower the device the larger will be it's latency bar profile. In the case of figures 3.5 and 3.7, the memory shows up as the slowest device.

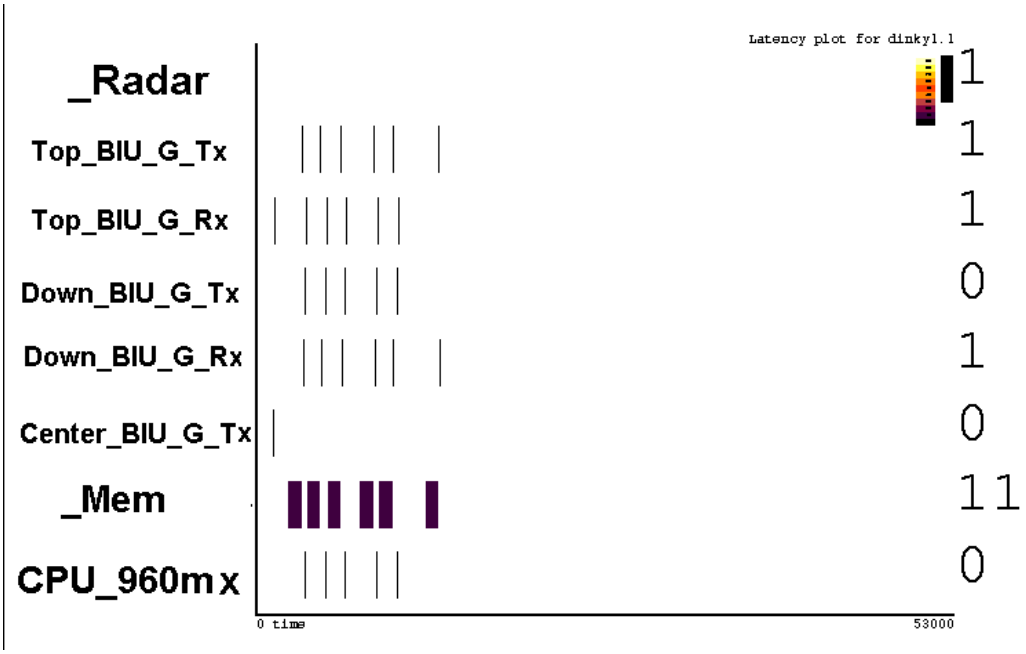

Figure 3.5: Latency Plot for one cycle of simulation of the single processor model.

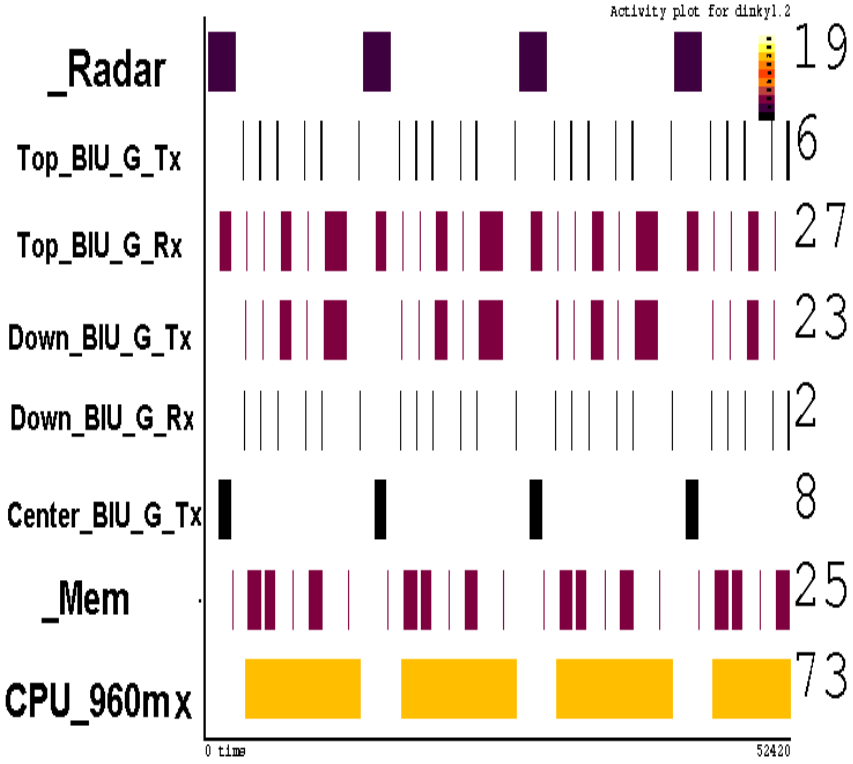

Fig. 3.6: Activity Plot for 3 & 1/2 cycles of simulation of the single processor model.

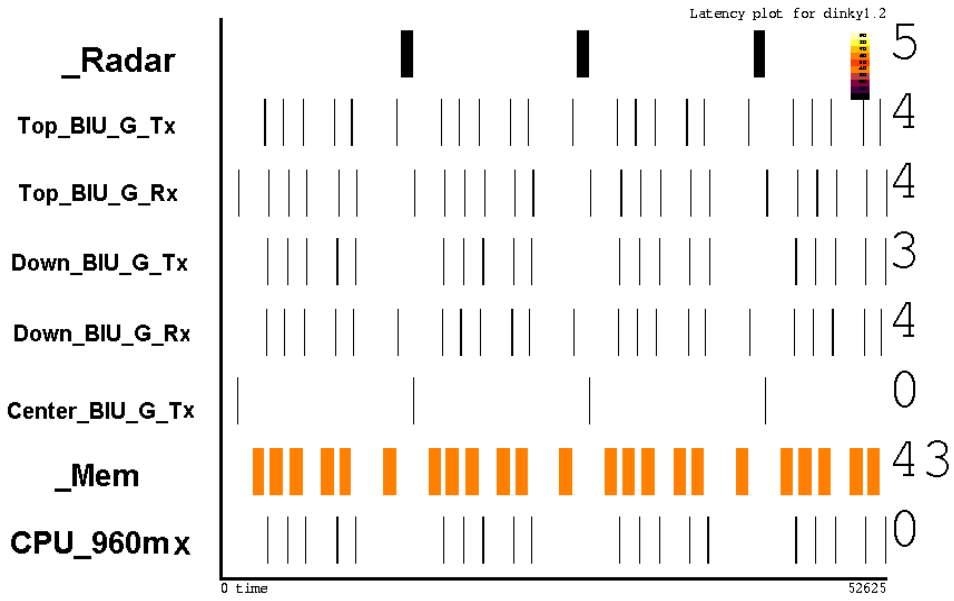

Figure 3.7: Latency Plot for 3&1/2 cycles of simulation of the single processor model.

3.2.1.4 Simulation statistics

The following explanation of the simulation statistics is valid for all the architectures and hence will not be repeated. The simulation statistics are tracked automatically by the Vantage™ Simulator by specifying the track statistic option of the batch simulator.

The first part of the statistics are obtained by the static analysis of the system. It provides information about the name of the (testbench) entity, the architecture, and the configuration for the complete system. The number of components portmapped within the structural model, the number of processes, the number of drivers etc. are also tracked. Other than the number of access objects all of the above statistics remain same for one cycle of simulation or for more than one cycle of simulation. The number of access

objects is the linked list record structure used to queue events and is approximately multiplied by the number of simulation cycles.

The run-time statistics appear in the lower part. They are explained as follows:

Current Simulation Time: The total simulation time for which the system was simulated

Current Simulation Delta: The number of delta delays totaled at the end of the simulation time.

Elaboration CPU time: The amount of CPU time the simulator spent in preparing the system for simulation. This includes initialization of variables and processes, creation of the trace file etc.

Simulation CPU time: The amount of CPU time the simulator spent in simulating the circuit after it was ready for simulation.

Transaction Count: The total number of signal transactions which took place during the completed simulation time. This figure when divided by the simulation time gives **Transacts / SimulationCpuSecond.**

Event Count: The total number of signal events which took place during the completed simulation time. This figure when divided by the simulation time gives **Events / SimulationCpuSecond.**

The circuit and simulation statistics are shown in the two figures below.

```
------------------ CIRCUIT STATISTICS ----------------------
Entity Name                          :  Dinky_testbench
Architecture Name                    :  testbench
Configuration Name                   :  DTB_01
Number of Components                 :  88
Number of Processes                  :  29
Number of Signals                    :  25784
Number of Drivers                    :  17549
Number of Access Objects             :  289 (85764 bytes)
```

```
---------------- SIMULATION STATISTICS --------------------
Current simulation time              :  53 ms
Current simulation delta             :  512
Elaboration CPU time (second)        :  1.383333e+00
Simulation CPU time (second)         :  2.166667e+00
Transaction Count                    :  351337
Event Count                          :  21643
Transacts / SimulationCpuSecond      :  1.621555e+05
Events / SimulationCpuSecond         :  9.989077e+03
```

Figure 3.8: Simulation Statistics for one cycle of simulation of the single processor model.

```
------------------- CIRCUIT STATISTICS --------------------
Entity Name                          :  Dinky_testbench
Architecture Name                    :  testbench
Configuration Name                   :  DTB_01
Number of Components                 :  88
Number of Processes                  :  29
Number of Signals                    :  25784
Number of Drivers                    :  17549
Number of Access Objects             :  930 (307888 bytes)

---------------- SIMULATION STATISTICS --------------------
Current simulation time              :  53 ms
Current simulation delta             :  1905
Elaboration CPU time (second)        :  1.383333e+00
Simulation CPU time (second)         :  6.816667e+00
Transaction Count                    :  1250238
Event Count                          :  78584
Transacts / SimulationCpuSecond      :  1.834090e+05
Events / SimulationCpuSecond         :  1.152822e+04
```

Figure 3.9: Simulation Statistics for 3&1/2 cycles of simulation of the single processor model.

3.2.1.5 Comments

In all the architectures a fixed common simulation time interval of 53 ms is considered for both the simulation-runs. This allows a fair comparison of the results of all the

architectures, both in terms of how long a single simulation cycle takes as well as in terms of how many simulation cycles can fitted within the simulation time interval of 53 ms.

It is important to keep in mind that the results of the single processor architecture cannot be directly compared with those of the multi-processor architectures because the task mapping is different in the case of the single processor architecture compared to the 4 processor architectures explained in the next few sections. Moreover the output device is not considered in the single processor architecture. Nevertheless the single processor architecture was used as a small testbench for the components and forms the building block for the next architectures down the line.

The results of all the architectures are compared at the end of this chapter in section 3.3.

3.2.2 The 4 Processor Single Global Bus Architecture

3.2.2.1 The Architecture Schematic

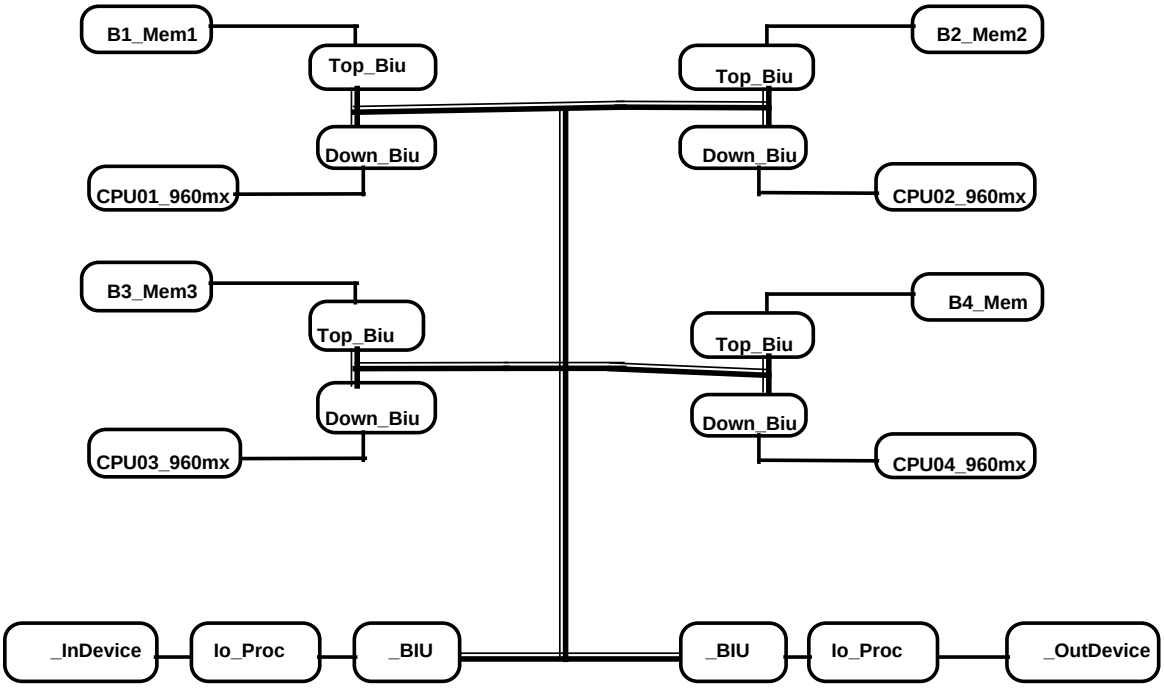

Figure 3.10: Schematic of the 4 processor single global bus architecture

This architecture consists of four single processor nodes interconnected in a star connection onto a single global bus. The local memories of the processors are also connected onto the global bus. This architecture has an outdevice component which models a slow mass storage media, typically a disk drive. The indevice models the input radar. The IO_Processors are modeled as pipelines. The IO_Processor attached to the indevice performs the routing of the radar pole tokens to the various local memories (B1_Mem1 through B3_Mem3).

3.2.2.2 The Tasks Mapping

In this architecture, parallel processing is achieved by dividing the Range processing among three processors while the fourth processor does Bin processing and storing. The Radar (_InDevice) sends pole pulses in succession to B1_Mem1, B2_Mem2 and then B3_Mem3. The pole pulse data is written to the corresponding memory by DMA. After the DMA is complete the local memory (B1_Mem1 through B3_Mem3) interrupts its corresponding local processor (CPU01 through CPU03) which begin the range processing. After the 3 CPUs (CPU01 through CPU03) finish the range processing they perform a pulse store into the local memory (B4_Mem4) of CPU4. They interrupt CPU4 after they have finished storing the pulse data into B4_Mem4. CPU4 then performs the BIN processing and BIN store to the Output device.

Sequence of Processing Steps for CPU0i (where i = 1,2,3) :

No.	Step Name	Token Type	Msg Size	Msg Value XX - Don't Care	Source	Destination
1.	Pole Pulse	Write	32 Kbytes	XX	_Indevice	_Mem_i
2.	Pole Ack.	Interrupt	1 Kbyte	Event_1	_Mem_i	_CPU_0i
3.	Pole Read	Read	1 Kbyte	32 Kbyte	_CPU_0i	_Mem_i

4.	Pole Read Ack	Data	32 Kbytes	XX	_Mem_i	_CPU_0i
5a.	Even FIR	(15 MPY-ADD) this is a processing step done on _CPU_0I				
5b.	Odd FIR	(15 MPY-ADD) this is a processing step done on _CPU_0i				
5c.	Form Complex	(12 FLADD) this is a processing step done on _CPU_0i				
6.	Weight Read	Read	1 Kbyte	48 Kbytes	_CPU_0i	_Mem_i
7.	Weight Rd. Ack	Data	48 Kbytes	XX	_Mem_i	_CPU_0i
8a.	Complex VPM	(6 Complex MPY = 24 MPY-ADDs) this is done by _CPU_0I				
8b.	Odd FIR	(8 FLADD Times) this is done by _CPU_0i				
9.	Pulse Write	Write	64 Kbytes	XX	_CPU_0i	_Mem_i
10.	Pulse Write Ack	Ack	1 Kbytes	XX	_Mem_i	_CPU_0i
11.	Pulse Read	Read	1 Kbytes	64 Kbytes	_CPU_0i	_Mem_i
12.	Pulse Rd. Ack	Data	64 Kbytes	XX	_Mem_i	_CPU_0i
13.	Ext. Pulse Write	Write	64 Kbytes	XX	_CPU_0i	_Mem_i
14.	Ext. Pulse Ack	Ack	1 Kbytes	XX	_Mem_i	_CPU_0i
15.	SEND EVENT	Interrupt	1 Kbytes	EVENT_2	_CPU_0i	_CPU_04

Fig: 3.11: Sequence of Processing Steps for the first 3 CPUs

Sequence of Processing Steps for CPU04 ˝

No.	Step Name	Token Type	Msg Size	Msg Value XX- Don't Care	Source	Destination
1.	WAIT FOR 3 Consecutive EVENT_2's					
For I in 1 to 4 LOOP						
2.	Pulse Read	Read	1 Kbyte	64 Kbyte	_CPU_04	_Mem_4

3.	Pulse Read Ack	Data	64 Kbytes	XX	_Mem_4	_CPU_04

END LOOP;

For I in 1 to 4 LOOP

4.	Bin Write	Write	64 Kbytes	XX	_CPU_04	_Mem_4
5.	Bin Write Ack	Ack	1 Kbytes	XX	_Mem_4	_CPU_04

END LOOP;

For I in 1 to 4 LOOP

6.	Bin Store	Write	64 Kbytes	XX	_Mem_4	_Outdevice

END LOOP;

Fig: 3.12: Sequence of Processing Steps for the fourth CPU

3.2.2.3 Results

For the 4 processor architectures one simulation cycle consists of three consecutive radar pole pulses sent for range processing to 3 CPUs. The fourth CPU takes as input the results of the first three CPUs and does the bin processing. In the first simulation only one cycle consisting of three radar pulses is considered. In the second simulation as many radar pulses as can be handled by the system are considered. There will be a limit on how many pulses can be packed into one simulation cycle. This is determined by the communication bottle-necks and processing speed. If too many radar pulses are considered within one simulation cycle then the bus interface units time-out waiting for the bus to become idle and the radar pulse tokens get lost.

The output trace file is postprocessed to obtain activity plots *(refer to figs. 3.13 and 3.15)* and latency plots *(refer to figs. 3.14 and 3.16).* The activity plot shows the period of activity of the various devices in the architecture. It makes use of the utilization

performance statistic to form the plots. One cycle of simulation is defined as the difference between the time of arrival of the first radar pulse to the time of completion of the writing to the output device. As can be seen in the activity plot *shown in figure 3.13*, in the case of the single simulation cycle, a succession of 3 radar pole pulses are sent (to B1_Mem1 through B2_Mem3). All the processing gets completed by 32790 ns. Thus the single cycle time in this case is approximately 32.79 ms. It is important to note that this cannot be directly compared to the single processor architecture where only one single radar pulse is processed in a simulation cycle.

Figure 3.15 shows that 2 groups of 3 radar pulses were sent during the simulation time of 53ms, and the second group of pulses get approximately half processed in that the Bin_Store operation on the Outdevice does not take place for the second cycle. Thus approximately one and a half cycles can be accommodated within 53 ms of simulation time.

The latency plots shows the latency distributions of the various devices in the architecture.

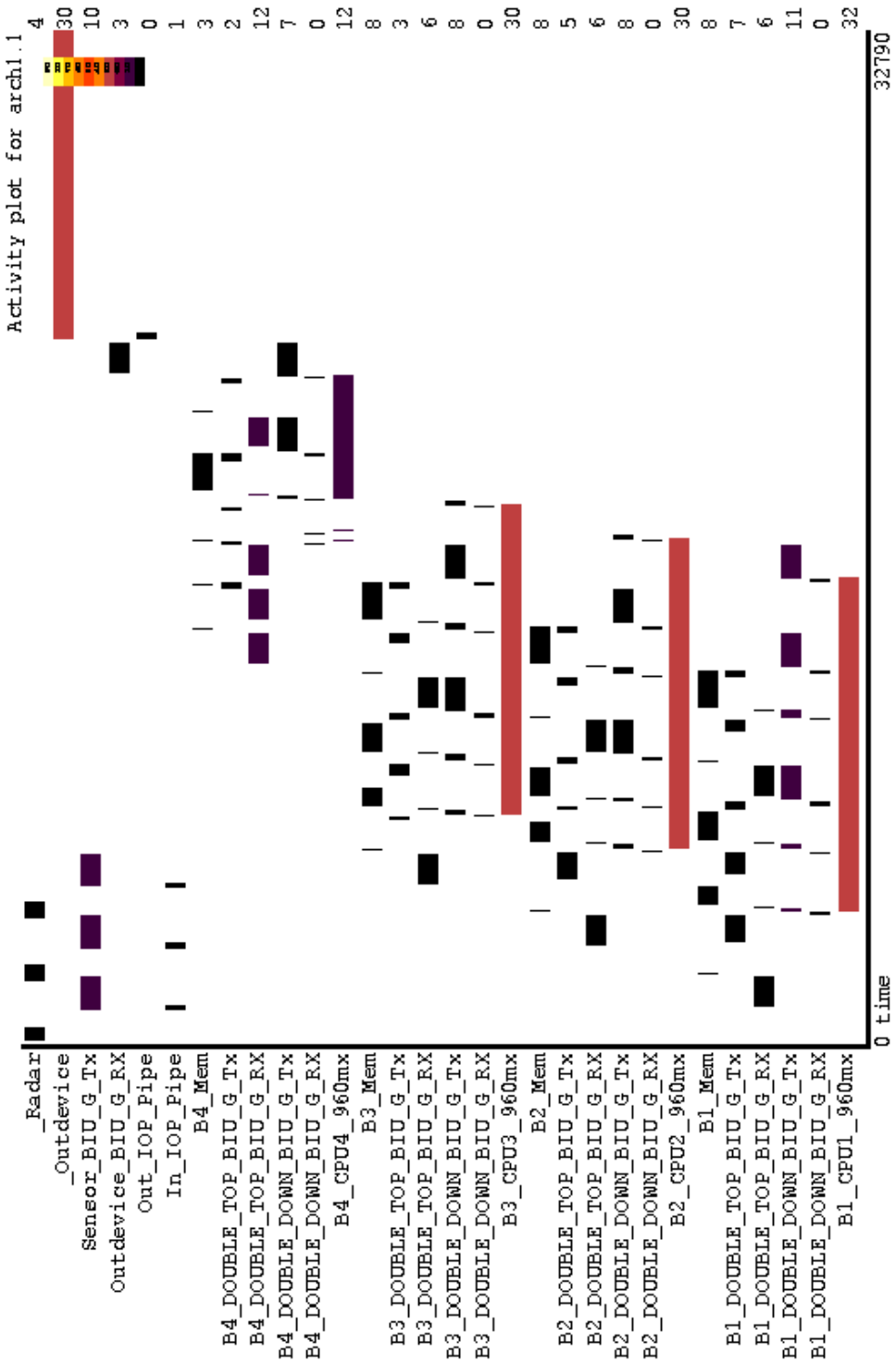

Figure 3.13: Activity Plot for one cycle of simulation.

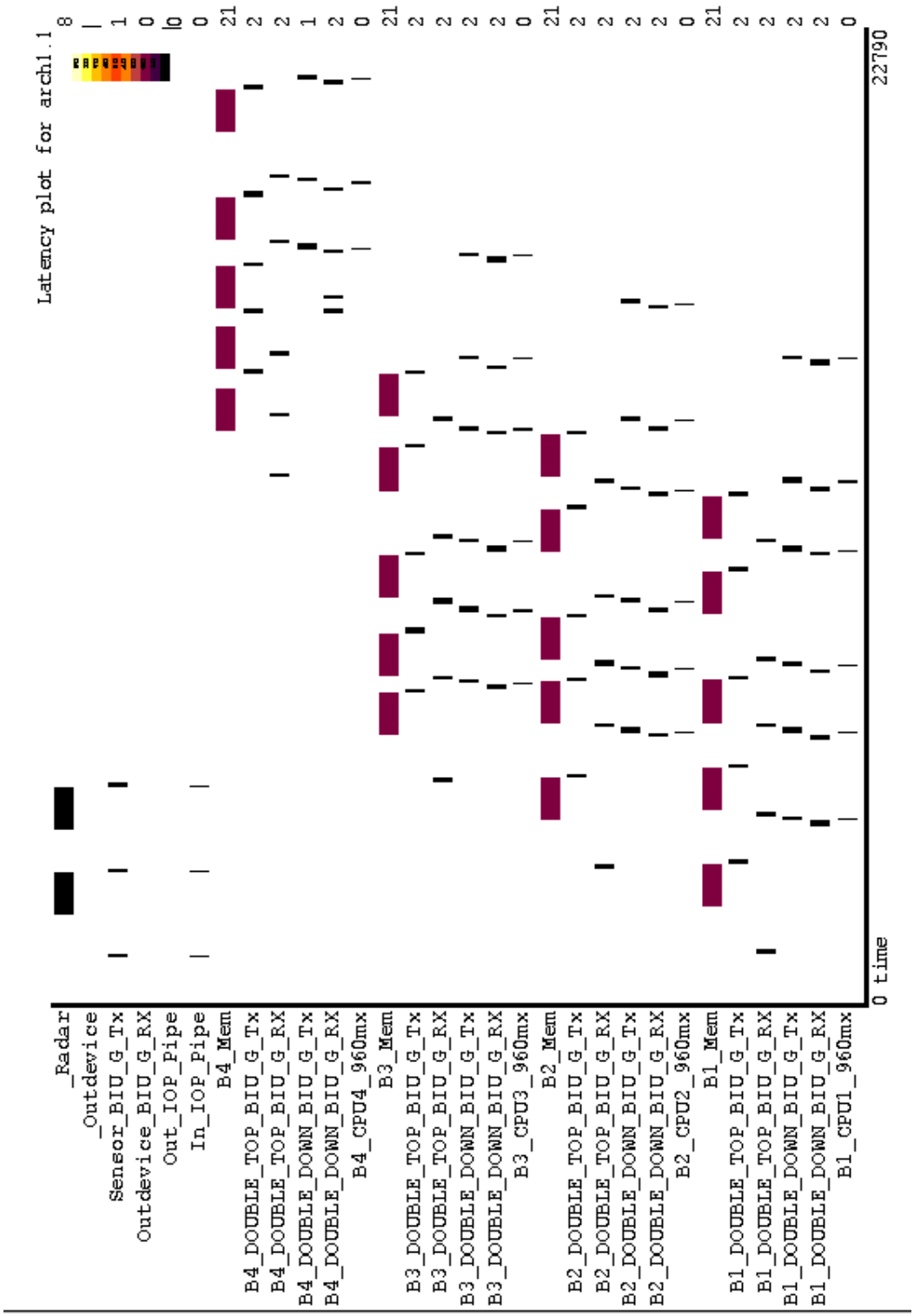

Figure 3.14: Latency Plot for one cycle of simulation.

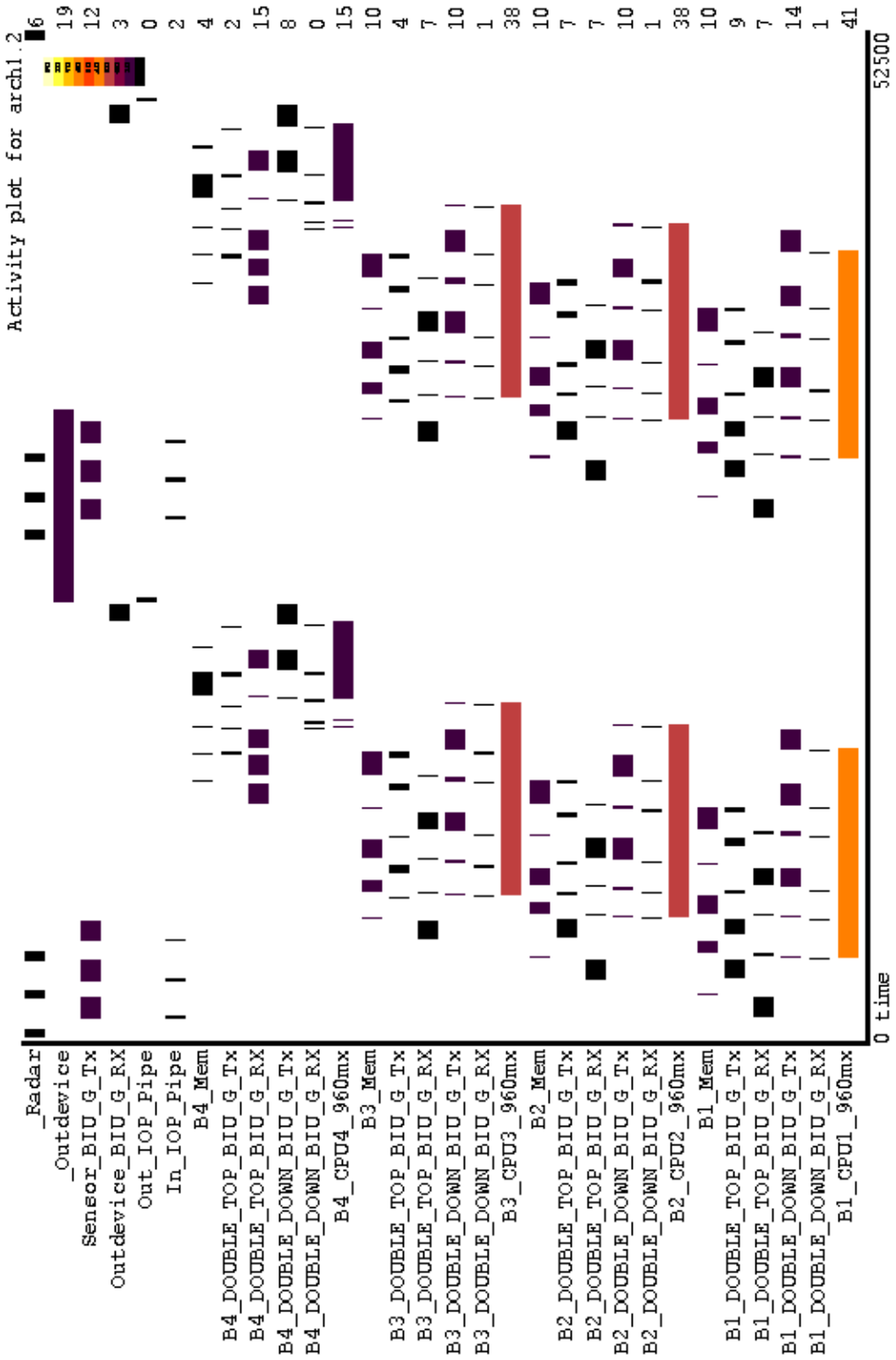

Figure 3.15: Activity Plot for one and half cycles of simulation.

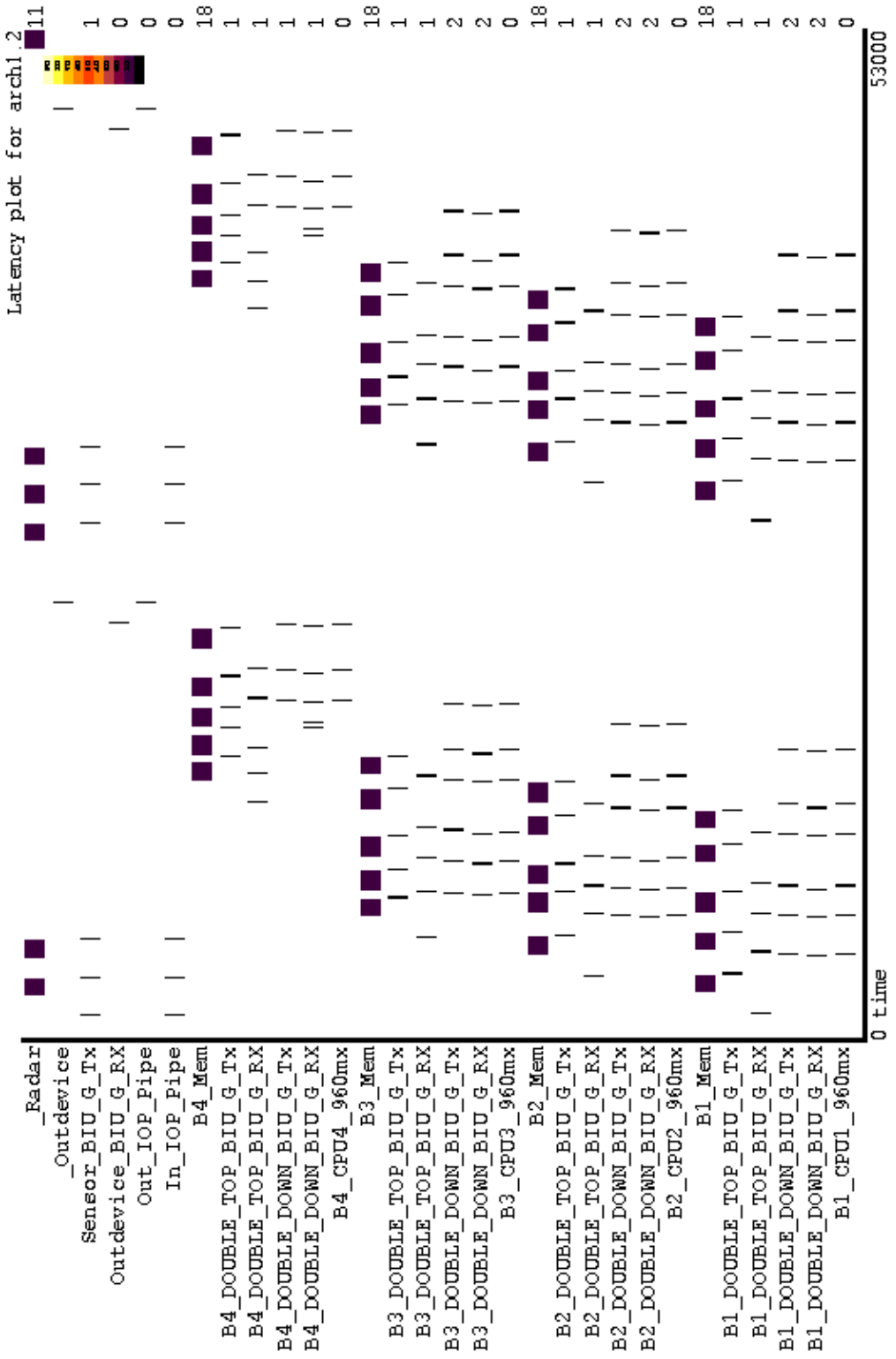

Figure 3.16: Latency Plot for one and half cycles of simulation.

3.2.2.4 Simulation Statistics

The circuit and simulation statistics are shown in the two figures below.

```
------------------- CIRCUIT STATISTICS -------------------
Entity Name                         :  Arch1_testbench
Architecture Name                   :  testbench
Configuration Name                  :  STP_01
Number of Components                :  268
Number of Processes                 :  101
Number of Signals                   :  95388
Number of Drivers                   :  66064
Number of Access Objects            :  1122 (343200 bytes)

----------------- SIMULATION STATISTICS ------------------
Current simulation time             :  53 ms
Current simulation delta            :  1778
Elaboration CPU time (second)       :  4.716667e+00
Simulation CPU time (second)        :  1.151667e+01
Transaction Count                   :  1434695
Event Count                         :  85847
Transacts / SimulationCpuSecond     :  1.245755e+05
Events / SimulationCpuSecond        :  7.454153e+03
```

Figure 3.17 : Simulation Statistics for one cycle of simulation.

```
------------------- CIRCUIT STATISTICS -------------------
Entity Name                         :  Arch1_testbench
Architecture Name                   :  testbench
Configuration Name                  :  STP_01
Number of Components                :  268
Number of Processes                 :  101
Number of Signals                   :  95388
Number of Drivers                   :  66064
Number of Access Objects            :  1987 (656656 bytes)

----------------- SIMULATION STATISTICS ------------------
Current simulation time             :  53 ms
Current simulation delta            :  3523
Elaboration CPU time (second)       :  4.600000e+00
Simulation CPU time (second)        :  1.810000e+01
```

```
Transaction Count                    :  2706641
Event Count                          :  166770
Transacts / SimulationCpuSecond      :  1.495382e+05
Events / SimulationCpuSecond         :  9.213812e+03
```

Figure 3.18 : Simulation Statistics for one and half cycle of simulation.

3.2.2.5 Comments

For this particular architecture, only one and a half cycles of simulation could be completed within the fixed simulation time-frame of 53 ns. The single global bus readily identifies itself as the bottle neck in the architecture. This is because even traffic between the CPUs and their local memories has to go through the global bus and hence keeps it busy when it is required by the input/output devices. This shows up in the simulations as assertion warnings from the Bus Interface Units complaining that the bus is busy when they try to access the global bus. This problem is alleviated by the next architecture.

This architecture would be recommended only for processor intensive tasks where not much of simultaneous bus traffic is involved.

The results of all the architectures are compared at the end of this chapter in section 3.3.

3.2.3 The 4 Processor Single Global Bus, Multiple Local Bus Architecture

3.2.3.1 The Architecture Schematic

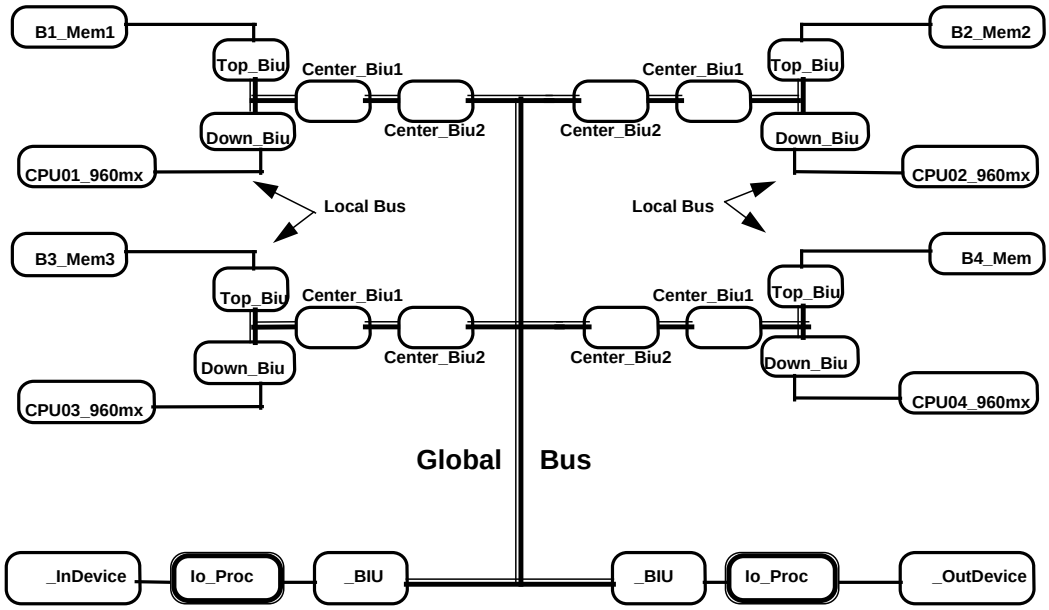

Figure 3.19: The 4 processor single global bus multiple local bus architecture

This architecture shown in *the figure above* is a refinement of the previous one, in that it has a two hierarchy bus system. A local bus system connects the CPUs to their local memory and this forms the basic processing node. The four processing nodes, the indevice and the outdevice are interconnected by the global bus.

3.2.3.2 The Tasks Mapping

The task mapping is identical to the earlier architecture.

3.2.3.3 Results

For the 4 processor architectures one simulation cycle consists of three radar pole pulses sent for range processing to 3 CPUs. The fourth CPU takes as input the results of the first

three CPUs and does the bin processing. In the first simulation only one cycle consisting of three radar pulses is considered. In the second simulation as many radar pulses as can be handled by the system are considered. There will be a limit on how many pulses can be packed into one simulation cycle. This is determined by the communication bottle-necks and processing speed. If too many radar pulses are considered within one simulation cycle then the bus interface units time-out waiting for bus to become idle and the radar pulse tokens get lost.

The output trace file is postprocessed to obtain activity plots *(refer to figs. 3.20 and 3.22)* and latency plots *(refer to figs. 3.21 and 3.23).* The activity plot shows the period of activity of the various devices in the architecture. It makes use of the utilization performance statistic to form the plots.

One cycle of simulation is defined as the difference between the time of arrival of the first radar pulse to the time of completion of the writing to the output device. As can be seen in the activity plot *shown in figure 3.20,* in the case of the single simulation cycle, a succession of 3 radar pole pulses are sent (to B1_Mem1 through B2_Mem3). All the processing gets completed by 37749 ns. Thus the single cycle time in this case is approximately 37.75 ms.

Figure 3.22 shows that 3 groups of 3 radar pulses were sent during the simulation time of 53ms, and the third group of pulses get approximately half processed by the CPUs. Thus approximately two cycles can be accommodated within 53 ms of simulation time.

The latency plots shows the latency distributions of the various devices in the architecture.

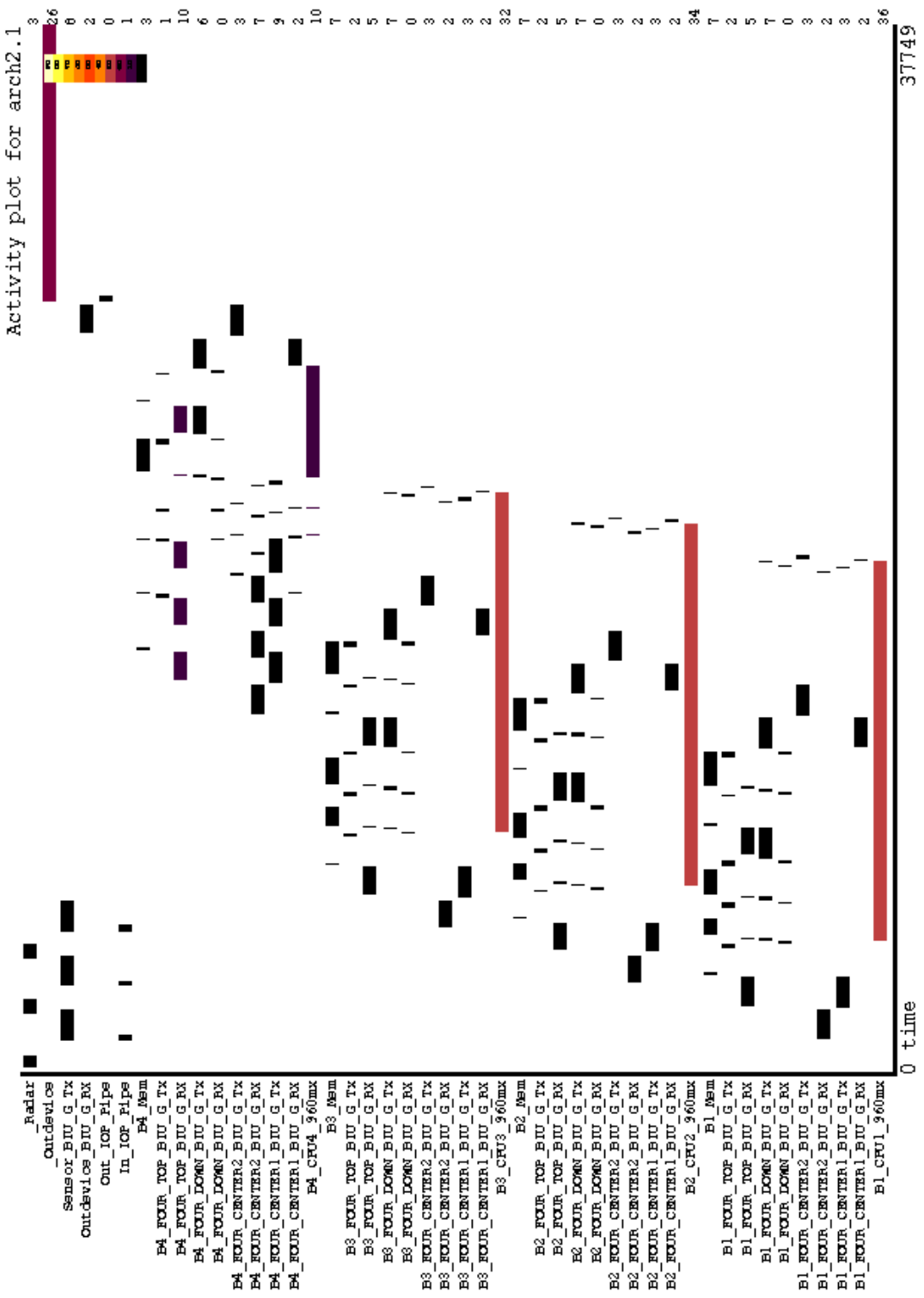

Figure 3.20: Activity Plot for one cycle of simulation.

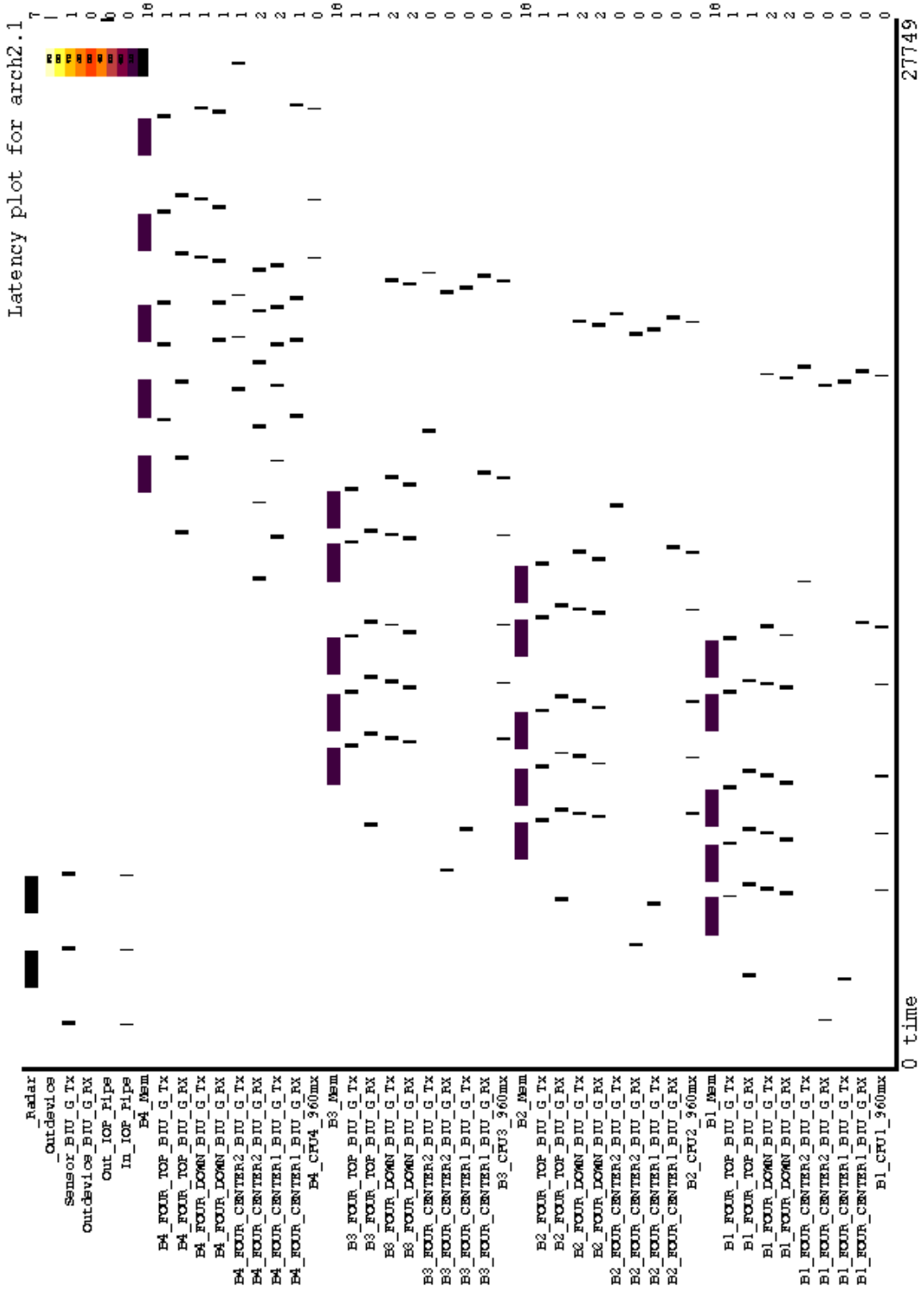

Figure 3.21: Latency Plot for one cycle of simulation.

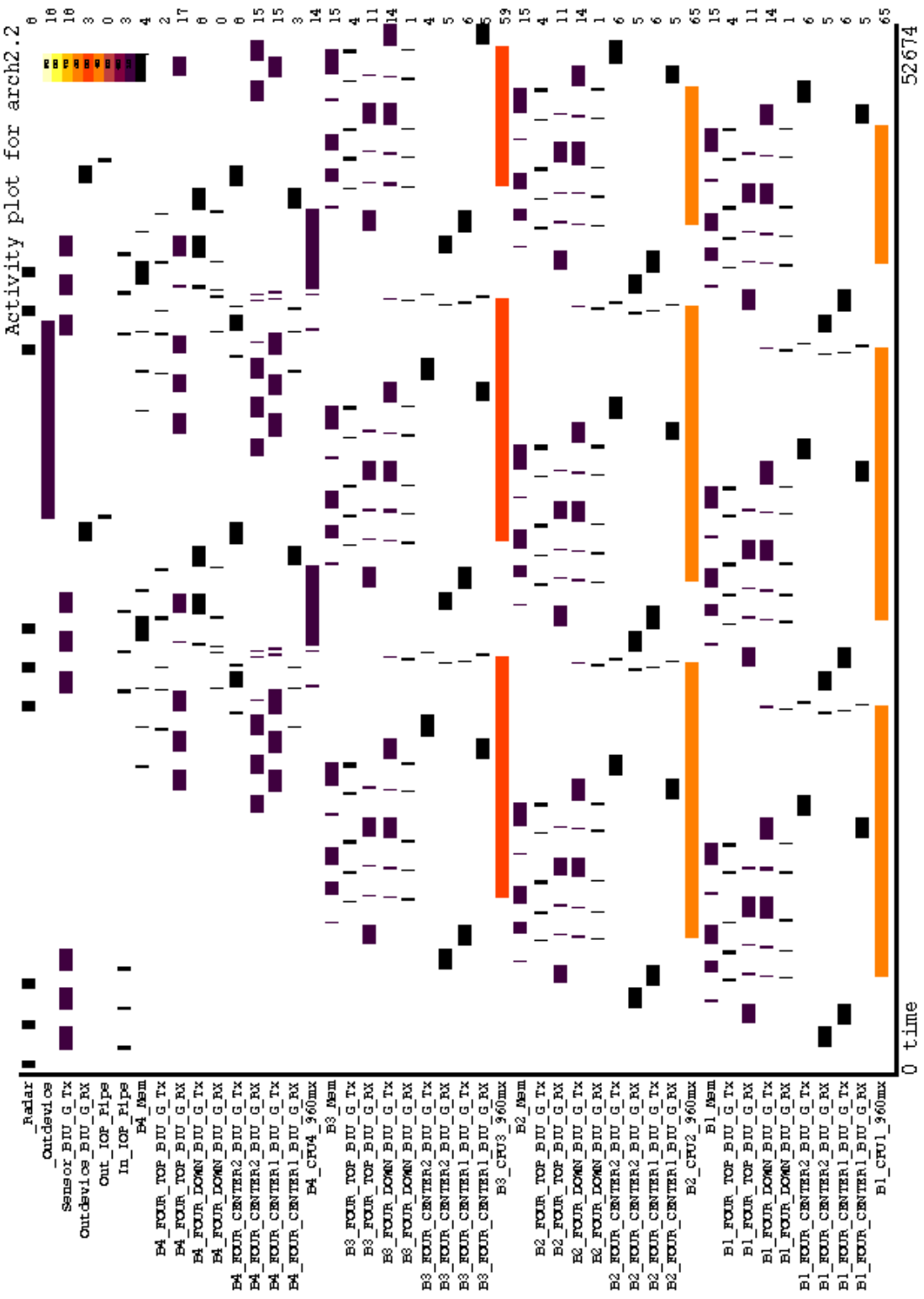

Figure 3.22: Activity Plot for two cycles of simulation.

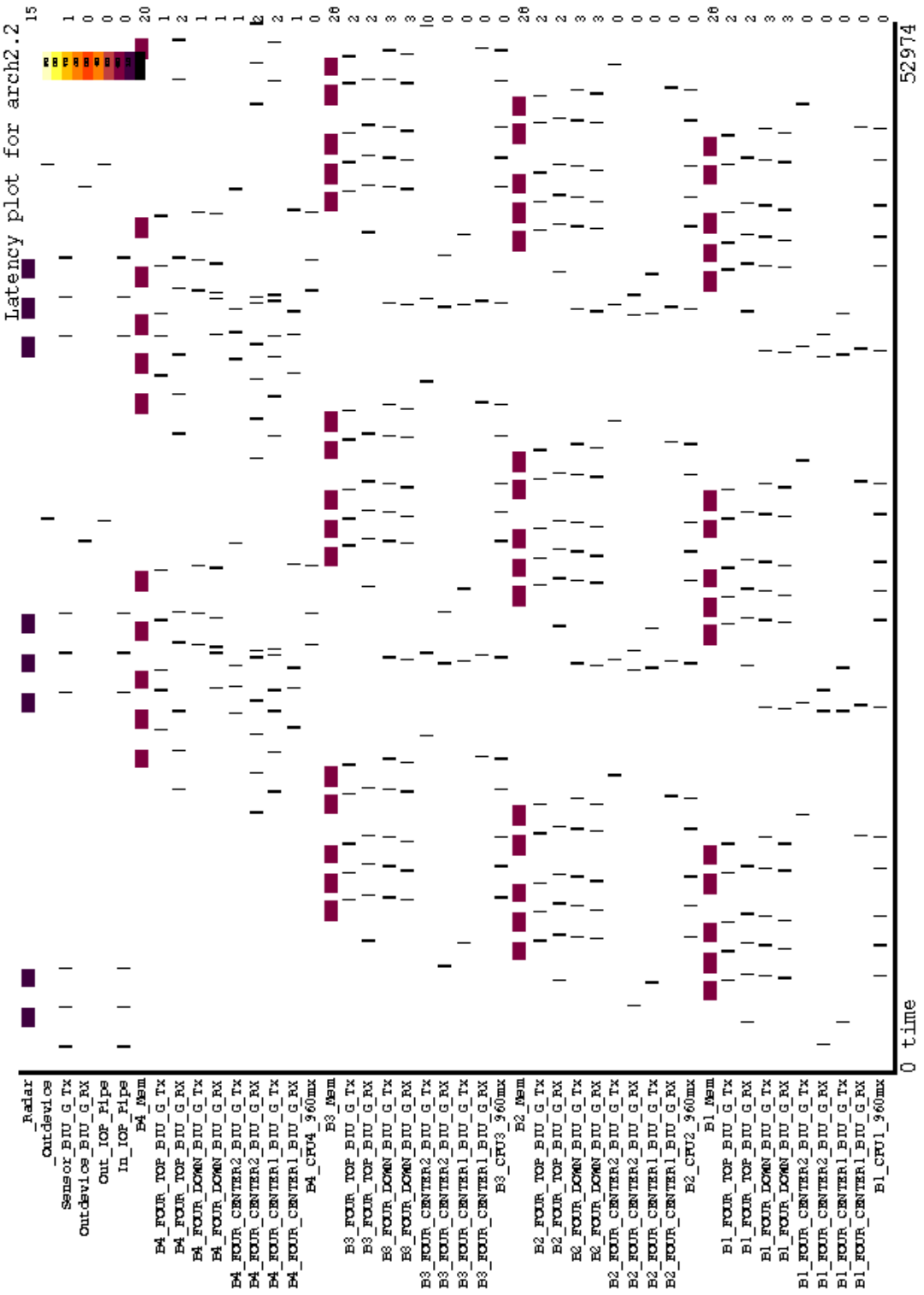

Figure 3.23: Latency Plot for two cycles of simulation.

3.2.3.4 Simulation Statistics

The circuit and simulation statistics are shown in the two figures below.

```
------------------- CIRCUIT STATISTICS --------------------
Entity Name                         :  Arch2_testbench
Architecture Name                   :  testbench
Configuration Name                  :  STP_03
Number of Components                :  300
Number of Processes                 :  133
Number of Signals                   :  115964
Number of Drivers                   :  80472
Number of Access Objects            :  1270 (396188 bytes)

----------------- SIMULATION STATISTICS -------------------
Current simulation time             :  53 ms
Current simulation delta            :  2111
Elaboration CPU time (second)       :  5.083333e+00
Simulation CPU time (second)        :  1.046667e+01
Transaction Count                   :  1609844
Event Count                         :  102583
Transacts / SimulationCpuSecond     :  1.538068e+05
Events / SimulationCpuSecond        :  9.800924e+03
```

Figure 3.24 : Simulation Statistics for one cycle of simulation.

```
------------------- CIRCUIT STATISTICS --------------------
Entity Name                         :  Arch2_testbench
Architecture Name                   :  testbench
Configuration Name                  :  STP_03
Number of Components                :  300
Number of Processes                 :  133
Number of Signals                   :  115964
Number of Drivers                   :  80472
Number of Access Objects            :  2932 (994240 bytes)

----------------- SIMULATION STATISTICS -------------------
Current simulation time             :  53 ms
Current simulation delta            :  5644
Elaboration CPU time (second)       :  4.900000e+00
Simulation CPU time (second)        :  2.316667e+01
Transaction Count                   :  3981084
Event Count                         :  258518
```

```
Transacts / SimulationCpuSecond     :  1.718454e+05
Events / SimulationCpuSecond        :  1.115905e+04
```

Figure 3.25 : Simulation Statistics for two cycles of simulation.

3.2.3.5 Comments

It is most interesting to note that for the given algorithm mapping, the single cycle time is slightly longer (37.75 ms) than the one observed in the pervious architecture (32.8 ms). This is interesting because one would have expected the two level bus hierarchy to speed up the simulation by alleviating the contention for the global bus. But this is achieved at a price. More BIUs are required in this architecture which add to the bus interface overhead. Hence it increases the single cycle time.

But when it comes to parallel processing, this architecture has a slight edge over the previous one in that it can process more simulation cycles within the set simulation time of 53 ms. Thus this architecture would be recommended for algorithms that exhibit parallelism which is suited to a two hierarchy bus.

The results of all the architectures are compared at the end of this chapter in section 3.3.

3.2.4 The 4 Processor Multiple Local Bus with Crossbar Architecture

3.2.4.1 The Architecture Schematic

The major difference between this architecture *(refer to figure 3.7)* and the earlier one (section 3.2.3) is that this architecture has a crossbar interconnection device added to replace the single global bus. This prevents the congestion of the global bus and facilitates faster inter-device communication.

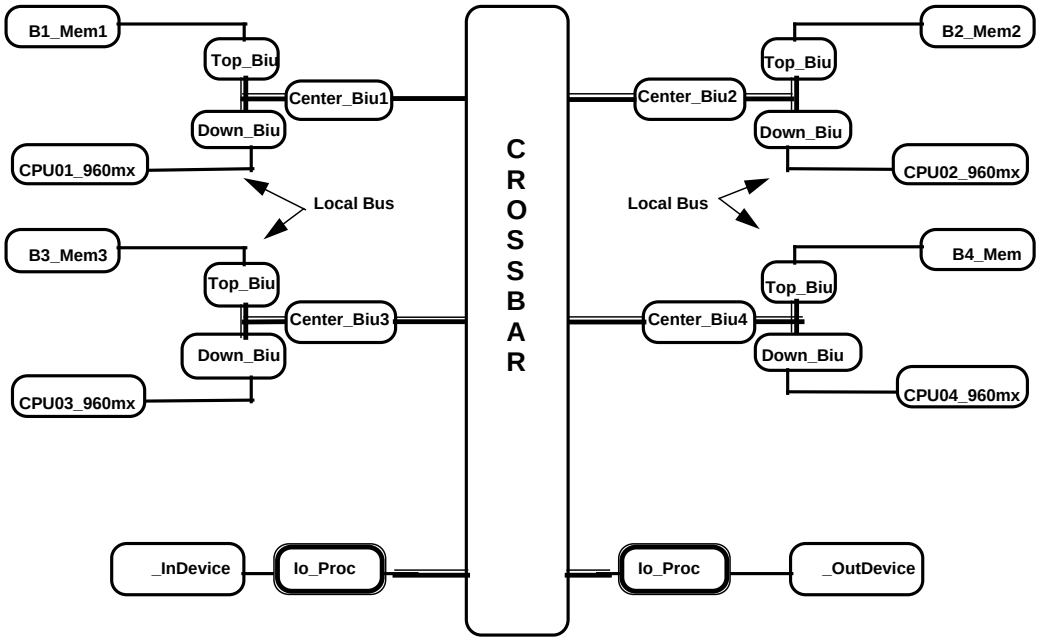

Figure 3.26: The 4 processor architecture with a crossbar interconnection device

3.2.4.2 The Tasks Mapping

The tasks mapping is identical to the one for the earlier architecture (Section 3.2.3).

3.2.4.3 Results

The output trace file is postprocessed to obtain activity plots *(refer to figs. 3.27 and 3.29)* and latency plots *(refer to figs. 3.28 and 3.30).*

One cycle of simulation is defined as the difference between the time of arrival of the first radar pulse to the time of completion of the writing to the output device. As can be seen in the activity plot *shown in figure 3.27*, in the case of the single simulation cycle, a succession of 3 radar pole pulses are sent (to B1_Mem1 through B2_Mem3). All the processing gets completed by 33802 ns. Thus the single cycle time in this case is approximately 33.8 ms.

Figure 3.29 shows that 4 groups of 3 radar pulses were sent during the simulation time of 53ms, and the third group of pulses get approximately half processed by the CPUs. Thus

approximately two and half cycles can be accommodated within 53 ms of simulation time.

The latency plots shows the latency distributions of the various devices in the architecture.

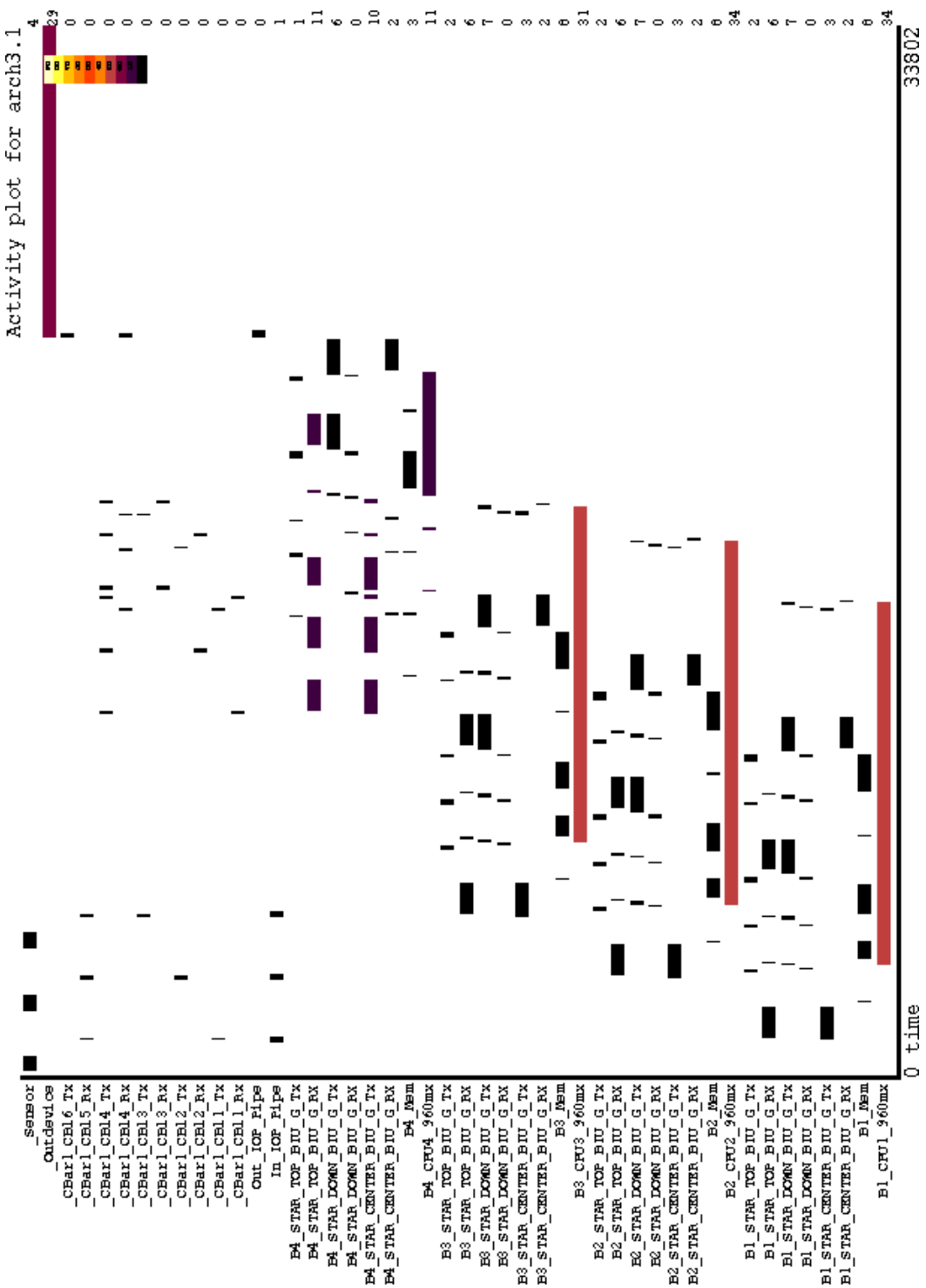

Figure 3.27: Activity Plot for one cycle of simulation.

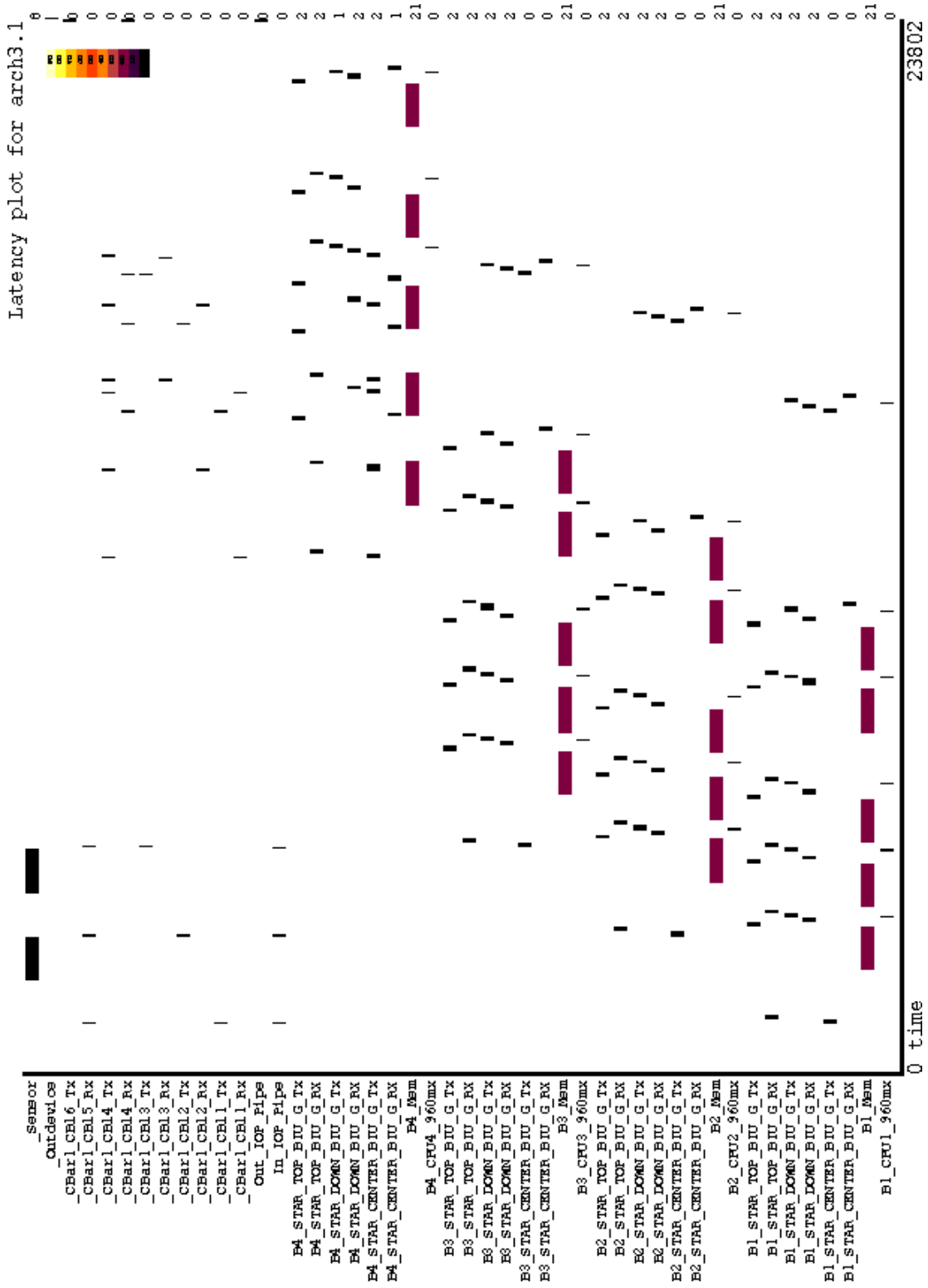

Figure 3.28: Latency Plot for one cycle of simulation.

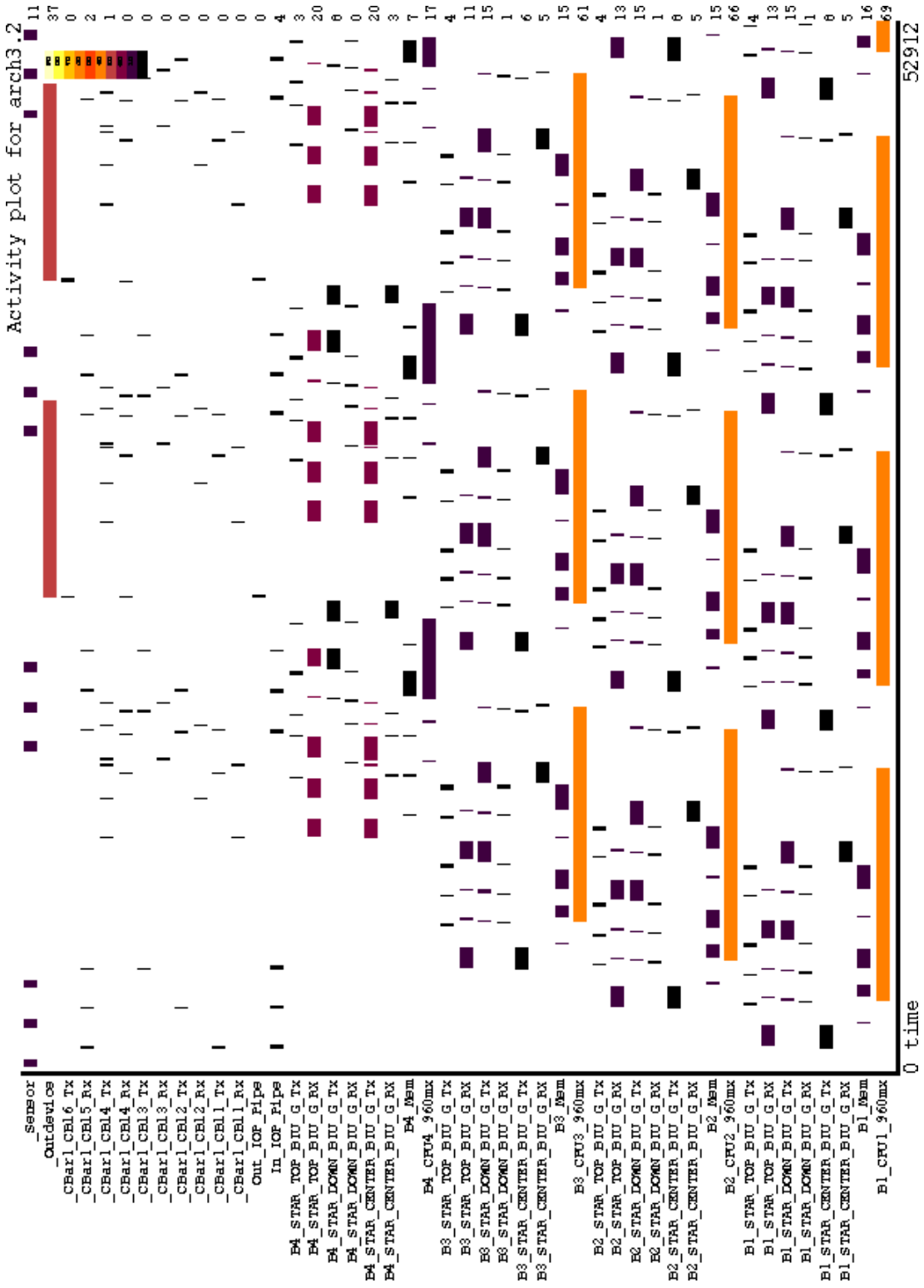

Figure 3.29: Activity Plot for two and half cycles of simulation.

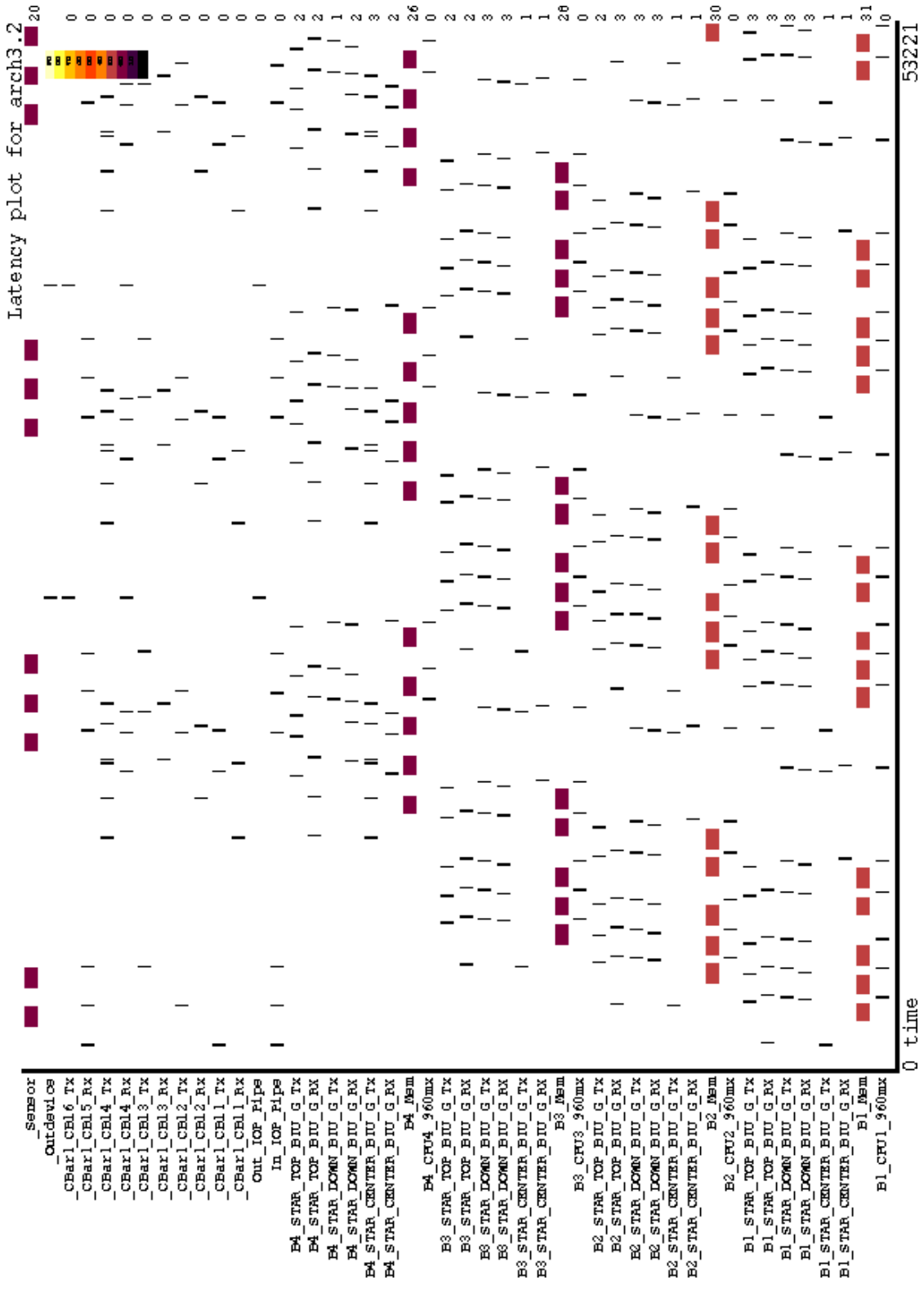

Figure 3.30: Latency Plot for two and half cycles of simulation.

3.2.4.4 Simulation Statistics

The circuit and simulation statistics are shown in the two figures below.

```
------------------ CIRCUIT STATISTICS ----------------------
Entity Name                          :  Arch3_testbench
Architecture Name                    :  testbench
Configuration Name                   :  STP_04
Number of Components                 :  289
Number of Processes                  :  127
Number of Signals                    :  138912
Number of Drivers                    :  95886
Number of Access Objects             :  1270 (392184 bytes)

------------------ SIMULATION STATISTICS -------------------
Current simulation time              :  53 ms
Current simulation delta             :  2106
Elaboration CPU time (second)        :  5.816667e+00
Simulation CPU time (second)         :  1.116667e+01
Transaction Count                    :  1918436
Event Count                          :  107549
Transacts / SimulationCpuSecond      :  1.718002e+05
Events / SimulationCpuSecond         :  9.631254e+03
```

Figure 3.32: Simulation Statistics for one cycle of simulation.

```
------------------- CIRCUIT STATISTICS --------------------
Entity Name                          :  Arch3_testbench
Architecture Name                    :  testbench
Configuration Name                   :  STP_04
Number of Components                 :  289
Number of Processes                  :  127
Number of Signals                    :  138912
Number of Drivers                    :  95886
Number of Access Objects             :  3359 (1148272 bytes)

------------------ SIMULATION STATISTICS -------------------
Current simulation time              :  53 ms
Current simulation delta             :  6471
Elaboration CPU time (second)        :  5.966667e+00
```

```
Simulation CPU time (second)        :  3.143333e+01
Transaction Count                   :  5503674
Event Count                         :  315314
Transacts / SimulationCpuSecond     :  1.750904e+05
Events / SimulationCpuSecond        :  1.003120e+04
```

Figure 3.32: Simulation Statistics for two and half cycles of simulation.

3.2.4.5 Comments

For the task mapping considered, this architecture gives the best overall performance in terms of single cycle speed as well as number of simulation cycles completed within the fixed simulation time. The cycle time of 33.8 ms for this architecture is nearly as good as the cycle time of 32.8 ms obtained for the single global bus architecture and is better than the 37.75 ms obtained for the 2-hierarchy bus architecture.

The parallel processing capability of this architecture is much superior to that of the single global bus architecture. The crossbar architecture can process approximately two and a half simulation cycles within a time frame of 53 ms, whereas the single global bus architecture can process only one and a half cycle. The 2-hierarchy bus architecture can process two cycles of simulation within a time frame of 53ms.

Thus this architecture would be recommended for applications which exhibit a high degree of parallelism which requires access to a common communication medium simultaneously.

The results of all the architectures are compared at the end of this chapter in section 3.3.

3.3 Comparison of the Results

The simulation results of the 4 architectures are compared in the following figure:

Architecture Name	***Single Proc.***	***4 Procs. global bus***	***4 Procs.2-level bus***	***4 Procs. with Crossbar***
No. of Components	88	268	300	289
No. of Processes	29	101	133	127
No. of Signals	25784	95388	115964	138912
No. of Drivers	17549	66064	80472	95886
No. of Access Objs	289 (85764 bytes)	1122 (343200 bytes)	1270 (396188 bytes)	1270 (392184 bytes)
simulation time	53 ms	53 ms	53 ms	53 ms
simulation delta	512	1778	2111	2106
Elab. CPU time(sec)	1.383333e+00	4.716667e+00	5.083333e+00	5.816667e+00
Sim. CPU time (sec)	2.166667e+00	1.151667e+01	1.046667e+01	1.116667e+01
Total CPU time	3.55 secs.	16.24 secs.	15.55 secs.	16.99 secs.
Sim. Efficiency. (%)	1.492	0.326	0.34	0.312
Trans Count	351337	1434695	1609844	1918436
Event Count	21643	85847	102583	107549
Trans/SimCpuSec	1.621555e+05	1.245755e+05	1.538068e+05	1.718002e+05
Events/SimCpuSec	9.989077e+03	7.454153e+03	9.8000924e+03	9.631254e+03

Figure 3.33: Comparison of Simulation Statistics for the 4 architectures for one cycle.

Simulation efficiency is taken as the ratio of the real logic time (i.e. simulation time) to the total CPU time. The ratio is expressed as a percentage (i.e. multiplied by 100).

Thus, Sim. Efficiency = (Simulation Time / Total CPU time) * 100

As can be seen in the above figure the complexity of the models increases from left to right, though the 4 processor 2-level bus architecture **is as complex as** the 4 processor

crossbar architecture. To consider the complexity of the model all factors such as number of components, number of processes, number of drivers, access object types, etc. have to be considered. As far as total simulation time goes (Elab. time + Sim. time), there isn't too much of a difference between the architectures, but it increases from left to right. It is interesting to note that the elaboration time (i.e. the time taken by the simulator to prepare the model for simulation) **is more for** the 2-level bus architecture than the single global-bus architecture where as the simulation time of the 2-level bus architecture **is less than** that of the single global-bus architecture. This can be explained as follows. The elaboration time for the single global-bus architecture will be less as the architecture has less number of components (BIUs in particular). But because the 2-level bus architecture prevents bus contention it simulates faster.

Architecture Name	***Procs. global bus***	***4 Procs.2-level bus***	***4 Procs. with Crossbar***
Cycle time (ms)	32.79 ms	37.75 ms	33.80 ms
No. of Cycles completed in 53ms	1.5 cycles	2.0 cycles	2.5 cycles

Figure 3.34: Comparison of cycle time and number of cycles completed within the benchmark simulation time.

The number of cycles an architecture can complete within the simulation time gives an indication of the throughput of the architecture. From the study of the above figure, the crossbar architecture has an edge over the other two architectures. It has a moderate cycle time, but can process more cycles per given time than the other two devices. Nevertheless it is a highly complex architecture.

Chapter 4. Conclusions and Future Work

4.1 Conclusions

The following contributions were made to the state of the art:

1. Developed a library of reusable models.

The original library of models was modified, customized and redesigned to meet the present research needs. This was the most time consuming and tedious task. But nevertheless the most important step. For the architecture analysis to be successful it is necessary to have a robust and bug free model components library. The modification and customization wasn't a simple task. The sheer huge size of the components library as well as the fact that the original library was intended for use in a different scenario added to the difficulty. A lack of documentation was also problematic.

2. Developed a Methodology for using the above library

A methodology was developed and is recommended for using the model library (Chapter 3, section 3.1). Part of the implementation of the methodology was done by other persons working on the project. Hence the complete application of the methodology to the

development of the architectures is not discussed in great depth in this thesis. The methodology suggested is not specific to this work and can be applied to the design of varied architecture systems.

3. Constructed sample architectures

Four different architectures with similar tasks mapping were constructed and simulated to demonstrate the proof of concept of the methodology as well as to test the model library. A sample architectural analysis was also presented.

4.2 Future Work:

The following areas could be explored for further refinements and additions to the present work:

1. Complexity

More complexity could be added to the architectures by adding more processors and devices. Also more complexity could be added to the models themselves by modeling the functionality of the models with a greater degree of fidelity.

2. Automation

The component mapping, task mapping, generation of the application software models, simulations, and postprocessing could be automated.

3. User-Friendliness

The user-friendliness of the library coupled with the design methodology needs to be improved. This would need to be done in conjunction with the automation step. Graphical user interfaces (GUI) would definitely help in making the library and the design methodology easy to use for the user/designer.

4. Light-Weight processor model

As mentioned earlier the processor model is one of the most complex models in the whole library. It has a large amount of intricate hardware and software modeling detail which may not always be required. This is especially true in very high level abstract models of system architectures. Hence a simpler processor model would be recommended to improve simulation speed.

References

[1] Roger Lipsett, Carl Schaefer, Cary Ussery, "VHDL Hardware Description and Design", Kluwer Academic Publishers, 1989.

[2] GPD VHDL Performance Modeling Methodology Document, Honeywell, Systems Research Center, SDRL 12, AF Contract No. F33615-90-C-3800, January 17, 1992.

[3] Comments from the VHDL code provided by Honeywell Inc. GOVERNMENT PURPOSE LICENSE RIGHTS LEGEND CONTRACT NUMBERS: F33615-94-C-1501 (Omniview PMW), DAAL01-93-C-3380 (Martin Marietta RASSP), F33615-94-C-1495 (VHDL Hybrid Models), F33615-92-C-3802 (CDG), CONTRACTOR: Honeywell Inc. 27327 .

[4] James R. Armstrong, F. Gail Gray, "Structured Logic Design with VHDL", PTR Prentice Hall Inc., 1993.

[5] David A. Patterson, John L. Hennessy, "Computer Architecture - A Quantitative Approach", Morgan Kaufmann Publishers Inc., 1990.

[6] Kai Hwang, Faye A. Briggs, "Computer Architecture and Parallel Processing", McGraw Hill, 1984.

Vita

Hormazd P. Commissariat was born on February 9th 1971, in the metropolis of Bombay, India. From kindergarten to the tenth grade he studied at St.Mary's High School in Bombay and then went to K.C.College, Bombay for his high school education (eleventh and twelfth grade). He received his Bachelor of Engineering (B.E.) in Electronics Engineering in May 1992 from the University of Bombay.

On completion of his BE he was employed by Godrej and Boyce Manufacturing Company as a Hardware Engineer for a period of six months in Bombay, India. He left Godrej and Boyce to join MASTEK (Management and Software Technology), Bombay, India to work as a Software Engineer. After working for a period of five months for MASTEK he decided to pursue his MS in the Electrical Engineering Department at Virginia Polytechnic Institute and State University (Virginia Tech) beginning from the Fall'93 term.

After completing his MSEE in the Summer'95 term, he will be working with Intel Corporation, Santa Clara, California as Components Design Engineer for the next generation Itanium (ia64) processor.