A Top-down Design Environment for Developing Pipelined Datapaths

Robert McGraw
RAM Laboratories
119 N. El Camino Real, Suite 175
Encinitas,CA 92024
rmcgraw@adnc.com

James H. Aylor
Department of Electrical Engineering
University of Virginia
Charlottesville, VA 22903
jha@virginia.edu

Robert H. Klenke
Department of Electrical Engineering
University of Virginia
Charlottesville, VA 22903
rhk2j@virginia.edu

1. Abstract

This paper presents a design environment for cycle-based systems, such as microprocessors, that permits modeling of these systems at various levels, from the abstract system level, through the detailed RTL level, to an actual implementation. The environment allows the models to be refined to lower levels in a step-wise manner. The environment provides the ability to obtain meaningful metrics from abstract models of a processor's architecture. This capability allows design alternatives to be evaluated earlier in the design cycle, thus eliminating costly redesign and reducing the processor time to market.

2. Introduction

Currently within the design community there is an increasing interest in the development of methodologies which reduce the time to market for a given system under development. One area of particular concern deals with the development of application specific processors [1]. With integrated circuits projected to reach the size of over 100 million transistors per die by the turn of the century [2], this increasing complexity must be handled properly so as not to adversely affect processor design time. One way to address this problem of complexity management is through the use of a top-down design methodology.

Top-down design methodologies have been used to design digital hardware design since the early 1970's [3]. A top-down design methodology follows a design from the top level, usually the specification level, of detail down to a detailed implementation. Model refinement in these methodologies works by having each level of detail serve as the design specification for the level of detail immediately below. It is acknowledged that if this hierarchical chain can be verified from one level of detail to the next, the resulting behavioral implementation will be "*right the first time*" [4]. Being able to develop systems that work on the first pass in a timely manner helps address the time to market problem. Unfortunately, there exists a lack of modeling environments which promote complete top-down design and refinement of processors from the system level.

This paper presents a timed cycle-based design environment which is geared toward the development of pipelined datapaths for processors and other synchronous systems. This cycle-based environment permits the processor designer to model and hierarchically refine pipelined processor datapaths from the system level down through the RTL level until a behavioral implementation has been developed. This paper focuses on the modeling and development of pipelined datapaths because most modern processor architectures contain considerable pipelining. The remainder of this paper is organized as follows; Section 2 presents a background of existing processor design environments. Section 3 presents an overview of the new design environment proposed herein. Section 4 describes the intermediate level modeling capability of the environment that provides a link between the abstract system level of modeling and the detailed functional level model. Finally, Section 5 presents an example of modeling a MIPS R4000 processor using the environment and Section 6 presents some conclusions.

3. Existing environments and methods

For a processor design environment to completely support top-down design and refinement, the environment must have some means of developing a system level processor model, some means of refining the system level model to the RTL level, and some means of providing abstract control to the datapath in order to obtain meaningful results from the model. At each level of design detail, different architectural analyses can be performed as detailed in Figure 1. For instance, at the system level, datapath control is often provided through the use of random distributions to exercise all model paths. Resulting analyses which can be performed include determination of cycle time and critical paths. At the RTL level, the design is very detailed and control is provided by a explicit control unit. At the RTL level, all functional and detailed performance metrics can be obtained. The need for a methodology and environment which supports the modeling and refinement of both system level and RTL level datapath models has been expressed in the literature [5,6].

Existing commercial design methodologies use a variety of tools to analyze designs at varying levels of detail. For example, Sun Microsystems uses architecture-specific simulators such as the UltraSPARC Performance Simulator (UPS) [7] to examine architectural trade-offs at a functional level. The UPS is a trace-driven simulator designed to simulate the Ultra-SPARC microarchitecture at a functional, RTL level of modeling. IBM uses several modeling tools to satisfy different parts of its design methodology during the development of its PowerPC line of processors. IBM examines architectural trade-offs at the functional level of detail by using the Basic RISC Architecture Timer (BRAT) [8]. The BRAT tool is an architecture-specific simulator. IBM also developed processor models using Verilog and their propriety Design Structure Language (DSL) which were used to analyze architectural trade-offs at the both the system and functional levels.The DEC design methodology for the 100 MHz CISC NVAX processor and the 200 MHz RISC Alpha AXP 21064 processor [9,10] included the analysis of the processor architecture starting at the RTL level using Digital's in-house hardware description

1-58113-049-x-98/0006/$3.50 DAC98 - 06/98 San Francisco, CA USA

Design Flow

Modeling Level	Datapath Control	Model Input	Type of Analyses
System Level	Distribution-based	Distribution-based	Cycle, setup and hold times, Critical Path Analysis
Intermediate System/RTL	Instruction-based Control provided through reservation tables or RTL level descriptions	Instruction Trace or Instruction Mix Driven	Cycle-time, Critical Path Analyses, Latency, Throughput, Concurrency, Register Setup and Hold Times, Determination of MIPS bounds
Register Transfer Level (functional)	Based on Modeled Control Unit or Datapath Information	Instruction Trace or Instruction Mix Driven	All Performance and Functional Analyses

Figure 1. Design flow for methodologies

language (DECSIM) based simulator. AMD's Am29000 processor was modeled at the functional level, using a specifically designed, C-based architectural language, and at the gate level to guide logic design [11]. Additional processor design methodologies currently being used deal with design starting at the functional level [12,13]. These methodologies are similar to bottom-up design strategies in that they are often based on existing architectures.

Using existing methods, the complete design of a processor cannot be performed in the same modeling and simulation environment. The current methodologies require the construction of multiple, disjoint, processor models at the system level and RTL level [14]. Typically these methodologies create system level models on a processor-by-processor basis using some type of modeling language or hardware description language. When a more detailed model of the processor is required, a new model is developed at the RTL level. There are several reasons for the creation of multiple models. First, processor modeling tools above the RTL level are relatively non-existent. Second, along with the need to address design refinement issues, existing environments do not have suitable methods for controlling an abstract datapath to produce meaningful results.

Several approaches have attempted to address processor and datapath modeling above the RTL level. Zhang and Grunbacher [5] have developed a Petri Net based design approach for pipelined processors. This approach allows for a design to be modeled at the system level through the use of Petri Nets. In addition, Razouk [6] has developed a timed Petri Net approach for detailed modeling of a processor design at the system level. Unfortunately, these methods lack the ability to link system level modeling environments to the RTL level of development.

This paper presents a timed-cycle-based design environment which allows for the modeling of processor datapaths above the RTL level. In particular, this methodology and environment provides datapath development constructs and control methods which link system level and RTL level models together through the use of an intermediate system/RTL level modeling domain. This intermediate system/RTL level domain, detailed in the shaded row of Figure 1, consists of a model of execution and datapath control methods which allow for the analysis of pipelined datapaths.

4. Environment and Model of Execution

A timed cycle-based processor design environment which specifically addresses the development of pipelined datapaths has been constructed. This environment supports system level processor modeling using abstract datapath constructs and mechanisms to control the datapaths. This environment addresses model refinement issues by providing modeling constructs and abstract control methods which bridge the system/RTL level modeling gap.

This design environment is based on the ADEPT performance modeling environment [15]. ADEPT is based on the VHSIC Hardware Description Language (VHDL) and provides a modeling environment where high-level models can be refined down to an implementation in an integrated manner. In the ADEPT environment, a system model is constructed by interconnecting a collection of *ADEPT modules*. The modules model the information flow, both data and control, through a system. Each ADEPT module is implemented in VHDL and communicates with other modules by exchanging *tokens* which represent data being transmitted in the system. The ADEPT modeling modules communicate via a four-state token passing protocol (present, acknowledged, released, removed). This protocol provides fully interlocked handshaking between elements. This type of asynchronous handshaking protocol is needed because the communications between the existing ADEPT modules is inherently asynchronous in nature. The VHDL code generated by ADEPT can be simulated using any IEEE 1076-87 compliant VHDL simulator. Facilities and programs to collect and analyze the simulation results are provided as part of the ADEPT system.

4.1 Design Flow and Datapath Control

The timed cycle-based environment augments the existing ADEPT environment to allow for the modeling of cycle-based systems, while still including the concept of asynchronous delay for combinational elements. The design flow using this timed cycle-based environment takes an instruction set architecture and refines it using modeling constructs of increasing detail through the RTL level down to a behavioral implementation. The cycle-based modeling constructs support datapath modeling at, or above, the RTL level. In addition, the existing capabilities for mixed-level modeling in ADEPT [15] allow RTL level models to be refined to an actual implementation in a step-wise manner. The modeling levels which are supported by the cycle-based modeling constructs include an abstract system level, an intermediate system/RTL level, and an RTL level. These modeling levels are unique in that they are exercised using different means of datapath control as more detail is entered into the datapath model. Figure 1 denotes the modeling levels along with methods for controlling those levels, means of exercising models developed at those levels, and the types of functional and performance analyses which can be performed at each level.

The system level modeling domain supports a very high-level of abstraction (almost all data and control have been abstracted away). This particular level of design can be equated to a "block-diagram" level of design detail. At this particular level, all of the clocked elements (registers, memories) are required to be present in the design. These elements receive or source the information tokens on every cycle. In addition, the combinational elements, between the clocked elements, are modeled simply as delay elements. The system level modeling constructs currently included in the environment include clocked register constructs and various routing elements which mainly deal with value-less (uninterpreted) tokens. An example of a system level model of a four-stage pipeline is shown in Figure 2(a). Datapath routing at this level is accomplished by using various stochastic methods. At this level, datapath control is provided by using stochastic distributions to make routing decisions as tokens arrive *at the routing elements*. The main goal of modeling at such a level is to ensure that the information flow between clocked elements meets the cycle time requirements.

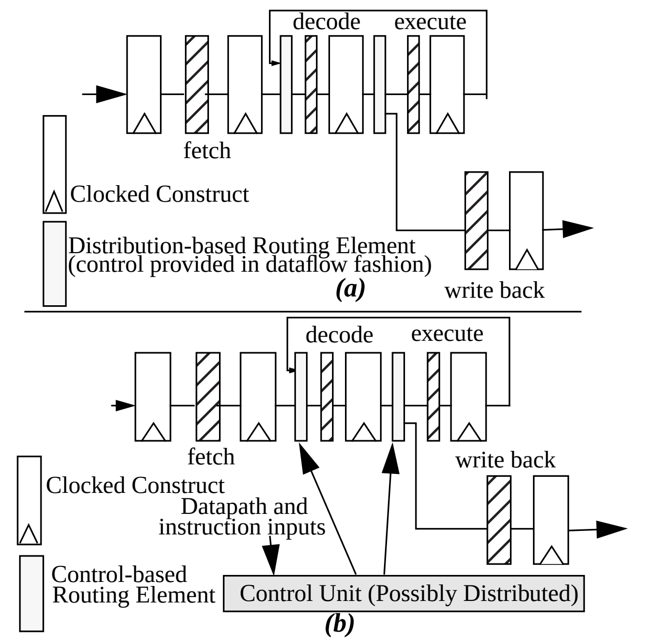

Figure 2. 4-stage pipeline (a) system level, (b) intermediate

The RTL level of modeling is much more detailed than the system modeling level. The RTL level modeling constructs include various clocked memory and register elements along with processor routing elements. An example of a RTL level model is shown in Figure 2(b). These RTL level modeling constructs model existing hardware elements such as multiplexers, demultiplexers and combinational logic using a one to one mapping of hardware signals to tokens or token values. The RTL level constructs are value-based. The constructs at the RTL modeling level route tokens based primarily on token values. The control for the RTL level datapaths is typically provided through some modeled control unit. In addition to having the responsibility of routing tokens from register to register, the RTL modeling level unclocked constructs also have the capability to operate on data (found on the token color fields).

The intermediate system/RTL level modeling constructs are the key to the environment in that they provide a link, through refinement, between the system level of modeling and RTL level of modeling. This intermediate system/RTL level modeling constructs and control methods are discussed in Section 4. By providing constructs which gradually incorporate more detail, the cycle-based design environment facilitates step-wise refinement from the system level to the RTL level.

4.2 Models of Execution

In order to communicate between various cycle-based modeling constructs, each construct must have a consistent model of execution. The model of execution refers to the way in which the modeling constructs of this environment communicate with each other. Because the modeling constructs must actually represent real systems or elements in a synchronous environment, models of execution for two types of elements are needed: clocked constructs (for synchronous elements) and unclocked constructs (for combinational elements).

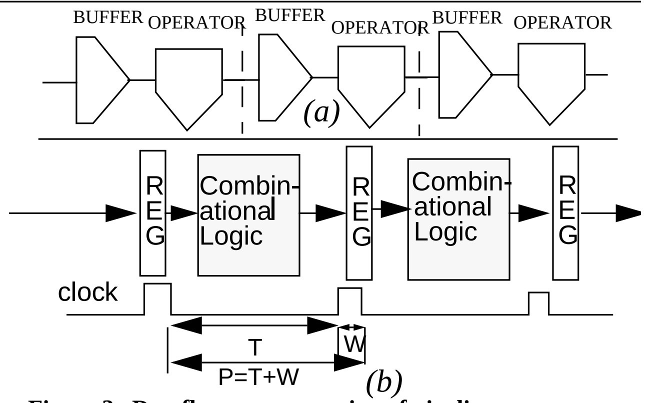

Figure 3. Dataflow representation of pipeline

Typically, existing processor datapaths can be represented using pipelined stages in a manner similar to Figure 3 [16]. Such a pipelined architecture is often implemented through stages of clocked elements (registers) followed by unclocked elements (combinational elements) as shown in Figure 3(b). Existing cycle-based environments typically map this pipeline architecture into the representation of Figure 3(a). Figure 3(a) shows each pipeline stage as being comprised of buffer elements followed by some type of operator element. The concept of buffering of information between modules is important because it is this buffering which separates the pipeline stages. In terms of the ADEPT four-way handshaking protocol, the buffer is the element that acknowledges the receipt of the token at the next stage of the pipeline. The operator element is viewed as an element which simply “operates” on arriving information before passing it on to subsequent pipeline stages. The operator modules do not buffer or acknowledge the receipt of information. The operator elements have an asynchronous delay representing combinational blocks and are known as the unclocked elements. In addition, the buffer elements are only allowed to acknowledge receipt of information on cycle boundaries. These are known as the clocked elements.

The model of execution for the unclocked elements is fairly straightforward. The unclocked constructs operate via the four-way interlocking handshake for asynchronous elements. These constructs map their inputs to their outputs using some type of control mechanism. This control mechanism may require inputs to be joined, synchronized, or forked in order to map them to the outputs. These constructs are also unbuffered in that they do not generate an acknowledgment upon the receipt of information. These constructs simply operate on arriving information and pass the information to the next construct.

The model of execution for clocked constructs is more complicated. The clocked elements are synchronized by some clock signal (to identify the cycle boundaries), yet these constructs must maintain a four-way interlocking handshake so they can communicate with the unclocked elements. In addition, these elements must contain buffering in order to acknowledge receipt of information at the cycle boundaries for each pipeline stage. For this reason, the model of execution for the clocked elements handles the four-way handshake, the buffering and acknowledgment of information, and the synchronizing of the inputs and outputs with respect to some type of clock signal.

The model of execution for both the unclocked and clocked

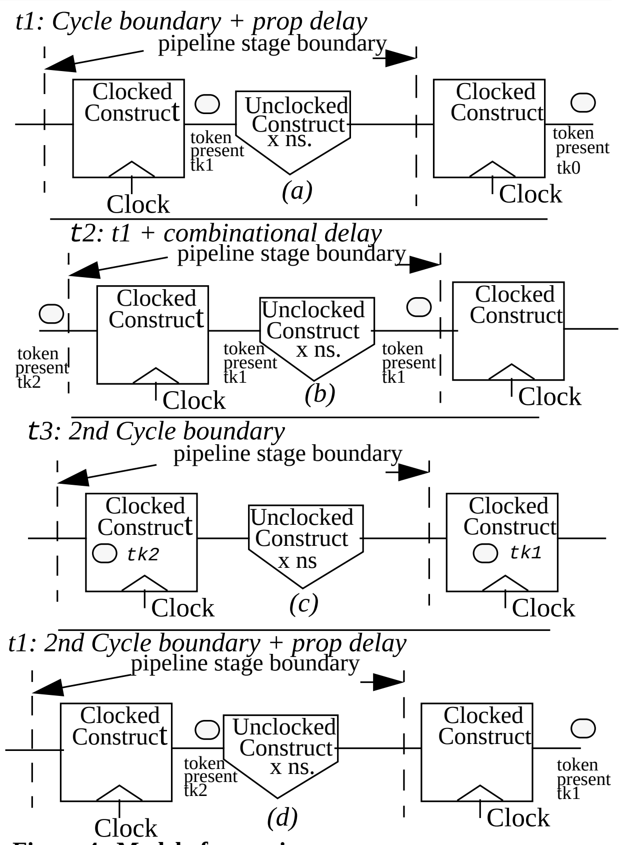

Figure 4. Model of execution

constructs for the cycle-based design environment is demonstrated in Figure 4 using a single pipeline stage. Figure 4(a) shows the clocked constructs outputting a token (*tk0*,*tk1*) at the cycle boundary after the propagation delay of the clocked constructs. This is represented by the token being present at the outputs of the clocked constructs. Figure 4(b) shows the token (*tk1*) propagating through the unclocked constructs after a delay of ***X*** (equal to the unclocked combinational delay). This results in the token, *tk1*, being present at the output of the unclocked construct. Because the unclocked constructs are unbuffered elements and do not generate their own acknowledgment, the input of the unclocked constructs still has token, *tk1*, present. The tokens remain in these “present” states until the cycle boundary is reached.

At the cycle boundary (determined when the clock signal is enabled), the clocked constructs copy the values on their input tokens to an internal token and finish the four-state handshake on their inputs. The clocked constructs then place their internal tokens on their outputs after accounting for the propagation delay *only* if those outputs are clear. This is the normal operating model of execution for the clocked constructs.

The models of execution for the clocked and unclocked constructs were verified using several basic architecture configurations. These basic configurations included linear pipelines, linear pipelines with single feedback loops, and linear pipelines with multiple feedback loops. In addition, each model of execution’s ability to handle a stalled pipeline (due to resource contention issues or multiple cycle delay stages) has also been examined and verified.

5. Intermediate System/RTL Level Modeling

This new environment is set apart from the existing environments in that it provides an intermediate system/RTL level of modeling constructs which bridges the system level to RTL level modeling gap for abstract processor datapaths.

5.1 Intermediate System/RTL Level Modeling Constructs

Datapaths developed using the intermediate system/RTL level modeling constructs can provide the designer with a more detailed datapath analysis than can be found using only a system level model. While continuing to allow the designer to perform cycle-time and critical path analyses, datapaths which are developed using the more detailed intermediate system/RTL level modeling constructs also allow the designer to examine concurrency issues and perform latency and throughput analyses. Also, the system/RTL modeling level can permit the designer to obtain an estimated value for *instructions per second* before a detailed design or a complete compiler for the processor are developed.

The intermediate system/RTL modeling level constructs route the datapath information based on the desired datapath routes needed to satisfy a particular instruction. Typically these datapaths will be exercised using a statistical instruction mix, although an instruction trace can also be used. Each element of the system/RTL level datapath receives the active instruction, or instructions, for the current cycle. Because a modeled control unit is typically absent at early stages of the design cycle, the datapath control must be solely based on this instruction and its associated instruction fields. The current instruction is provided through the use of a colored information token. The control for the system/RTL level datapaths is dependent upon this current instruction and provided in two ways, depending upon the type of datapath and analyses required by the designer. Control for un-pipelined datapaths is provided based on the register transfer description for each instruction. Control for pipelined datapaths is provided using the reservation tables which describe the stage to stage information flow for each instruction. The reservation table-based control methods and modeling constructs are described in Section 5.2.

5.2 Reservation Table Control Methods

The goals of analyzing such a pipelined datapath would be to obtain latency and throughput information as well as a bounds for *instructions per second* for a pipelined execution unit under a given instruction mix or workload. One way of providing the control information for pipelined units is to make use of design methods concerning the design of pipelined execution units. In order to analyze the operation of pipelined execution units (such as integer pipelines and floating point units) system designers often use reservation tables [17,18]. Reservation tables are used to specify the use of given resources used by a instruction as it proceeds through the pipeline. Reservation tables can be used to determine instruction latency, or how long an instruction has to wait at the “head” of the pipeline before entering without causing resource contentions. These reservation tables can be used to give the designer a rough idea of attainable throughput and latency metrics concerning any pipelined unit.

The system/RTL level modeling constructs allow the designer to encode these reservation tables in a file. Figure 5 shows the reservation table for an integer instruction for the four-stage DLX pipeline of Figure 6[19]. The intermediate system/RTL level model of the four-stage pipeline is shown in Figure 6. The coded reservation tables are accessed by the intermediate system/RTL level routing elements and used to control the pipelined datapaths on a cycle-by-cycle basis.

The reservation tables are employed at the pipeline’s clocked constructs to control instruction initiation within the pipeline. The pipehead_cyc element, shown in Figure 7, is the clocked element which has been developed to govern instruction initiations. The pipehead_cyc modeling construct requires four generic properties: *i_tag*, *trig_tag*, *delay*, and *filename1*. The *i_tag* property specifies the token color tag on which the instruction information is contained. The *delay* property specifies the propagation delay tokens encounter while passing through the pipehead_cyc element. The *filename1* property specifies the file which contains the names of the coded reservation table data files and reference numbers for all pipeline reservation tables which are used in the pipeline model.

The pipehead_cyc element is placed at the head of the top-level pipeline. When instructions arrive at the pipehead_cyc construct, they are checked for resource conflicts with all resources in the pipeline. First, the pipeline status reservation table is accessed. This pipeline status reservation table contains the status information (stage and cycle markings) for the pipeline referenced by the pipehead_cyc construct. This pipeline status reservation table is intersected with the reservation table of the incoming instruction to determine if a resource contention will occur if that instruction is initiated. If a resource contention will occur if the instruction is initiated, then no initiation is made for that cycle and the instruction is left on the pipehead_cyc element's input. This allows the same instruction to be presented on the subsequent cycle.

The reservation tables are also utilized at the pipeline's unclocked routing elements to control the stage-to-stage routing for each instruction within the pipeline. The unclocked routing units have their outputs bound (using defined properties and net interconnections) to different stages of the modeled pipeline. Tokens arriving at the unclocked elements are routed by accessing their reservation tables, and identifying the stage(s) to which they should be routed for that cycle. An example of such a routing element is the piperoute2 element, which is shown in Figure 8. The piperoute2 is used to route tokens internal to the pipeline execution units using reservation tables. This element requires several generics in order use reservation tables to assist in routing tokens. The *i_tag* and *filename1* properties are the same as those found in the pipehead_cyc element. In addition to these properties, the piperoute2 element also has output binding properties *Outbindings1* and *Outbindings2*, a *max_inst* property, a *maxclkcyc* property, a *pipelength* property and a *cyc_no_tag* property. The *max_inst* property specifies the maximum number of instructions to be handled by the pipeline containing this particular element. The *maxclkcyc* property specifies the maximum number of cycles required to complete any instruction. The *pipelength* property specifies the number of stages in the pipeline. The *cyc_no_tag* property specifies the token color tag field which contains the cycle count for each instruction. The *Outbindings* properties specify the stage connectivity for each output. These properties are arrays which list the stages which connect to the current stage. For example, the four-stage DLX pipeline of Figure 6 contains a piperoute2 construct after its decode stage (stage 2). This routing element is required because information needs to be routed to the write back (stage 4) or execution (stage 3) stages after the decode stage. For this reason, the *Outbindings* properties of the piperoute2 construct are assigned to stages 3 and 4 respectively. When the token arrives at the piperoute2 element, its reservation table is accessed and the token is routed to the output which is referenced in the reservation table for that cycle.

5.3 Hierarchical Modeling Using Reservation Tables

In order to facilitate top-down design and refinement, the timed cycle-based environment has the capability of modeling hierarchical pipelines using the reservation table control methods. The control methods developed allow for the insertion of a "low-level" pipeline into a top-level pipelined datapath.

The pipelined cycle-based constructs were used to model a five-stage DLX pipeline with multiple execution units. Figure 9 shows the five-stage DLX pipeline (fetch, decode, execute, memory access, write-back) where execution unit 2 (Ex-2) represents a multi-function floating point unit that has its own underlying reservation table. Ideally, once this pipelined stage has been designed, it would be desired to hierarchically replace the single Ex-2 stage in the top-level pipeline with the 3-stage multifunction pipeline. In addition, it is also necessary to perform all routing at this "lower-level" by routing this 3-stage pipeline locally using its own reservation table. The original top-level reservation table is then altered to reflect the extra cycles spent in the multi-function

<u>cycle</u> stage	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>
Fetch	X				
Decode		X		X	
Ex			X		
WB					X

Coded Reservation Table

```
i1
1 0 0 0 0
0 1 0 1 0
0 0 1 0 0
0 0 0 0 1
```

Figure 5. Reservation Table and Coded File for Figure 6

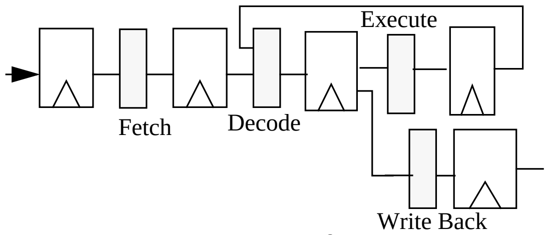

Figure 6. Block Diagram of Pipelined Datapath

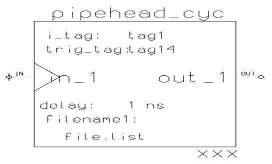

Figure 7. Pipehead_cyc Modeling Construct

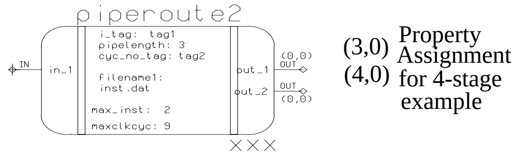

Figure 8. Piperoute2

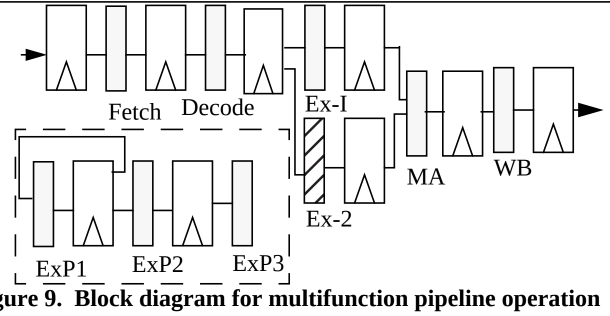

Figure 9. Block diagram for multifunction pipeline operation

Ex-2 unit.

The pipehead_cyc modeling constructs have the capability to examine all levels of pipeline hierarchy to decide when instruction initiations can be performed at the head of the top-level pipeline. By allowing the pipehead_cyc modeling construct at the head of the top-level pipeline to examine the reservation tables for all levels of pipeline hierarchy, refined lower-level pipelines can be simply "plugged in" to the top-level datapath model.

6. Example --- MIPS R4000 Processor

The timed cycle-based modeling environment's intermediate system/RTL modeling level has been verified through the use of several examples. One such example involves the modeling of a MIPS R-4000 processor [20]. The MIPS R-4000 is a 9-stage pipelined processor. These stages include: instruction fetch 1, instruction fetch 2, register fetch, execution, data fetch 1, data fetch 2, tag check and, write back. The execution units used for the execution pipeline stage include both an single stage integer execution unit and a floating point unit consisting of 8 pipelined stages. The MIPS R-4000 was modeled in hierarchical fashion with separate reservation tables developed for the integer and floating point execution units.

The reservation tables for both the top-level MIPS DLX pipeline and the internal pipeline execution unit were developed by hand for each instruction. During simulation, each modeling element accesses these reservation tables to determine if an instruction can be initiated and where information needs to be routed to. The MIPS R-4000 model was exercised using the SPEC95 benchmarks [21]. The benchmark instruction traces were obtained by compiling the SPEC95 source code on a Silicon Graphics MIPS R4000 machine and outputting the symbolic assembly instruction trace. This assembly language instruction trace was then mapped to instruction tokens entering the DLX pipeline. The MIPS R-4000 DLX model was simulated using a Mentor Graphics QuickVHDL simulator on a Sun Sparc-10 workstation. The simulation showed that the R-4000 processor executing the SPEC benchmark tomcatv.f, had a millions of instructions per second (MIPS) value of 75.76 using a 100 MHz clock. The published performance rating for the 100 MHz SGI R4000 was 76.5 indicating the abstract model was useful in obtaining a ballpark performance metric for millions of instructions per second. The model simulates at 16.4 cycles per minute of CPU time. It should be noted that this model did not take into account such issues as cache hits and misses and interrupts. Because this model was constructed at the intermediate system/RTL modeling level, statistical probabilities were used to help predict instruction branching. Exact branching values were not used because this model is an abstract model and does not contain the detail required to obtain those values. By using distributions to predict when branches could by taken, the model was able to use the SPEC benchmark traces in sequence to obtain a representative workload.

7. Summary and Conclusions

This paper presented a timed cycle-based design environment which provides a means for modeling and simulating processor datapaths at high levels of design abstraction. This environment was made possible by developing modeling constructs and abstract control methods which facilitate the modeling and control of processor datapaths above the RTL level. The methods for controlling the abstract processor datapath models are rooted in existing processor design methods and have been extended to assist in exercising meaningful processor models at early stages of the design. By obtaining meaningful metrics from abstract models of the processor's architecture, design decisions can be evaluated earlier in the design cycle, thus eliminating costly redesign and reducing the processor time to market.

8. References

[1] Proceedings, First Annual RASSP Conference, August 1994.

[2] Heaton, J., "Simulation - A Key to Smart Design,". Proceedings, Institution of Electrical Engineers, 1995.

[3] Rose, Charles. "The What and How of Top-Down System Design" TD Technologies, 1993.

[4] Transcend Promotional Document, TD Technologies, Inc.

[5] Zhang, Q.and H. Grunbacher. "Petri Nets Modeling in Pipelined Microprocessor Design," Applications of Theory of Petri Nets, pp. 582-591. 1993.

[6] Razouk, Rami R. "The Use of Petri Nets for Modeling Pipelined Processors," 25th ACM/IEEE Design Automation Conference. pp. 548-553.

[7] Tremblay, Maturana, Inoue, and Kohn. "A Fast and Flexible Simulator for Micro-architecture trade-off analysis on Ultra-Sparc-I," 32nd Design Automation Conference, 1995. pp. 2-6.

[8] Poursepanj, et al."The PowerPC 603 Microprocessor: Performance Analysis and Design Trade-offs," IEEE Spring COMPCON 1994 pp. 316-323.

[9] Dutton, Todd A. "The Design of the DEC 3000 Model 500 AXP Workstation," IEEE Digest of Papers, COMPCON, Spring 1993. pp 449-454.

[10] Peng, Donchin, Yen. "Design Methodology and CAD Tools for the NVAX Microprocessor," IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1992 pp. 310-313.

[11] "Simulation in the design of the Am29000 microprocessor," Electronic Engineering. November 1987. pp. 44-52.

[12] Taylor, Rekow, Radke, and Thompson. "A 100 MHz Floating Point/ Integer Processor," IEEE 1990 Custom Integrated Circuits Conference. pp. 24.5.1-24.5.4.

[13] Narita, Arakawa, Uchiyama, and Kawasaki. "Design Methodology for GMicro/500 TRON Microprocessor," IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1993 pp. 253-257.

[14] Franke, D., Purvis, M., "Hardware/Software Codesign: A Perspective", 13th International Conference on Software Engineering, May 1991, pp. 344-352.

[15] Klenke, R. H., M. Meyassed, J. H. Aylor, B. W. Johnson, R. Rao, A. Ghosh, "An Integrated Design Environment for Performance and Dependability Analysis," Proceedings of the ACM Design Automation Conference, June 1997 pp. 184-189.

[16] Kogge, Peter M. The Architecture of Pipelined Computers. Hemisphere Publishing Corporation, 1981.

[17] Stone, Harold S. High-Performance Computer Architecture. Addison-Wesley Publishing. 1993.

[18] Hayes, John P. Digital System Design and Microprocessors. McGraw-Hill, Inc. 1984.

[19] Hennessey, John L. and David A. Patterson. Computer Architecture: A Quantitative Approach. Morgan Kaufmann, Publishers, San Francisco, Ca. 1996.

[20] MIPS R-4000 User's Manual, Silicon Graphics, Inc.

[21] Standard Performance Evaluation Corporation. 1995 Benchmarks.