VHDL Extensions For Complex Transmission Line Simulation

Peter Walker

Division of Engineering
Brown University
Providence , RI 02906
paw@cs.brown.edu

Sumit Ghosh

Computer Science Dept
Arizona State University
Tempe, AZ 85287
sumit.ghosh@asu.edu

Abstract

This paper proposes extensions to the VHDL grammar and defines new semantics in the language to model the timing behavior of high frequency buses and clock lines with multiple, distinct taps in a VHDL description. The proposed language constructs utilize transmission line analysis to model the timing behavior but avoids the continuous time simulation by using line-events*.*

1 Introduction

The VHDL language design [1] incorporates two timing constructs – "inertial" and "transport" delay. The semantics of inertial delay allows for both recognition and preemption of inconsistent events, as outlined in [2], that may be generated during the simulation of the circuit models. In contrast, the transport delay semantics assumes that a signal, asserted by a source at one end of a hardware device, will be propagated to the receiver and is not subject to preemption. In VHDL, the transport delay semantics extends to hardware devices such as buses and interconnect lines that exhibit infinite frequency response, i.e. any pulse is transmitted regardless of its duration. Transport delay semantics however lack the ability to capture effectively the behavior of transmission lines. Perturbation due to reflection at the intermediate taps on a bus is ignored and this results in incorrect timing behavior. Higher clock speeds and newer bus design techniques, such as Intel's PCI [3] bus, increasingly behave as transmission lines and effective simulation of such systems with VHDL requires extensions in its capability.

1.1 Weaknesses In VHDL

Consider a one-bit digital bus to which three digital devices **A**, **B** and **C** are connected at the locations. Assume that the propagation delay from **A** to **B** is 10ns while that from **C** to **B** is $5ns$. Also, assume that at time $t = 0$, **A** asserts a pulse of width 2ns on the line. According to the VHDL transport semantics, the leading edge of the pulse is scheduled to arrive at **B** at time $t = 10ns$. Assume that at time $t = 2ns$, **C** introduces a pulse also of width 2ns on the line. The VHDL transport semantics will schedule the leading edge of the pulse at **B** at $t = 7ns$. Representing this scenario in VHDL requires elaborate coding to capture the function of this system. Furthermore, as the number nodes on the bus increases the complication of representation increases and ability to capture subtle line behavior is reduced.

Furthermore, VHDL allows an arbitrary mix of transport and inertial assignments to be made on signals which may lead to errors in simulation. Also, the semantics of transport assignment is such that when transactions are asserted on the signals all components that read the bus sees the event simultaneously. Discrete timing between taps cannot be easily accommodated.

VHDL permits a designer to define a signal of type bus and an associated resolution function, both of which are intended to define a bus to which multiple components of the system are connected. However, resolution functions do not consider the inter-node timing along the line, only the logical value.

2 Modeling a Digital Bus

The subject of transmission line analysis is a well established engineering science [4]. A typical digital bus consists of multiple drivers and loads placed at appropriate intervals along a conductor. Figure 1 presents a digital bus with drivers placed at locations i, j, and k along the length of the conductor. The source voltages are labeled $v_s^i(t)$, $v_s^j(t)$, and $v_s^k(t)$, while the source impedances are represented through Z_i, Z_j, and Z_{ki}. The characteristic impedances of the conductor segments between locations $\{i, j\}$ and $\{j, k\}$ are represented through $Z_{o_{ij}}$ and $Z_{o_{jk}}$.

In Figure 1, when a wave, $v_{ij}^+(t)$, traveling from i to j, reaches j, the contribution by the incident wave to

ED&TC '97 on CD-ROM

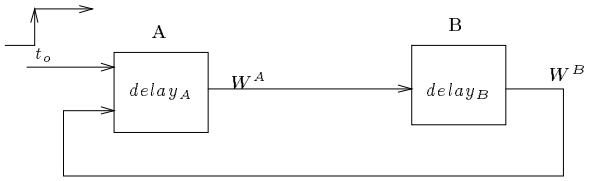

Figure 1: Cyclic Circuit

to outputs on which events are pending.

Optimistic simulation uses both concurrency by assuming all events are concurrent. P^2EDAS is designed to exploit both types of simulation concurrency using conservative methods.

2.1 Simulation Via Event Prediction

Consider a distributed simulation in which each circuit component is simulated within independent simulation processes. Given the absence of the global event queue, to ensure correctness in simulation the following is required. A *simulation clock value*, $clock^N$, is computed for every component N and is defined as the time up to which the model has been simulated. The clock is manipulated locally and without regard to other components clock. In the simulation of digital systems often events are not generated at the output of a component following its execution. Under such circumstance, events are not propagated over to the inputs of components that are connected to the output and consequently their simulation clock value stagnates periodically or deadlock occurs in the simulation. By computing the time of next event in the circuit and propagating it, the simulation clock can be kept active.

Consider the cyclic circuit shown in Figure 1. Propagation delay $delay_A, delay_B$ are associated with components **A** and **B** respectively. Assume event at time t_o is such that $t_o \gg (delay_A + delay_B)$. The circuit is to be simulated by computing the time of next event at each component input so as to determine the instances of temporal independence concurrency. At initialization, the predicted time of next event at output of **A** and **B** is respectively $W^A = 0$ and $W^B = 0$. Assume a sequentially ordered computation of prediction values starting at **A**. Thus computing the initial prediction computation yields $W^A = min(W^B + delay_A, t_0 + delay_A) = min(delay_A, t_0 + delay_A) = delay_A$. Similarly at **B** we have $W^B = min(W^A + delay_B) = delay_A + delay_B$. The new value of W^B is now used in equation to compute W^A. Thus the computation iterates over the loop until it is determined that $W^B > t_0$ at which point **A** realizes that it may execute the event at time t_o. The number of iterations before the executable event in the circuit is determined is given by $N_{iter} = \lceil \frac{t_o}{delay_A + delay_B} \rceil$. If the cost of computing the time of next event at a component is $C_i, \quad i \in \{1..n\}$, the prediction cost in given by $N_{iter} \sum_{i=1}^{n} C_i$. In general, if the time difference to the next executable event in a loop is T and loop delay is $d_l = \sum_{i=1}^{n} delay_i$, n the number of components in the loop, the number of iterations to discover the executable event is given by $N_{iter} = \lceil \frac{T}{d_l} \rceil$, which is highly inefficient.

3 The P^2EDAS Algorithm

P^2EDAS performs event prediction via an event prediction network. The network is synthesized for a given digital system to be simulated and executes concurrently with the simulation of the behavior descriptions. It consists of mathematical entities, termed *pseudo components*, that generate predicted event time. A *predicted event time*, defined at an output signal of a pseudo component, is the earliest time at which an event is expected to be generated at that component output. The predicted event time is computed separately from the actual execution of the behavior descriptions. The simulation utilizes the predicted event times to determine when execution of models are possible. Components in a cyclic subsystem have *head* and *tail* pseudo components. Acyclic components have only head pseudo components. The minimum input predicted event time to a head pseudo component defines the time up to which the component can be executed without violating the causality constraints. This time is called the *temporal independence limit*, t_{win}. All events at the input of the simulation model with time less than the temporal independence limit may be executed by the component model.

3.1 Event Prediction Network(EPN) Synthesis

In constructing the event prediction network for the digital system under simulation, first the combinational and sequential subcircuits are identified. This is done by determining the strongly connected components and synthesizing the EPN for each section. The individual EPN sections are then connected to make a complete EPN of the circuit.

For a cyclic subcircuit a feedback edge set [7] given by $S = \{E_1, E_2, \cdots E_n\}$ of a directed graph corresponding to the subcircuit is identified such that the graph may be rendered acyclic following the removal of all of the edges E_1 through E_n. Correctness is not affected by the identification of the minimal feedback edge set which is a computationally demanding problem for large circuits [8]. For each $E_j, \quad \forall j \in \{1, 2, \ldots, n\}$, in the original directed graph, a new acyclic directed graph is

constructed by replacing E_i with two unconnected edges E_j^{in} and E_j^{out}.

The EPN for the cyclic subcircuit, is synthesized from connecting two identical copies of the acyclic circuit through a *prediction switch*. The EPN section to the left and right of the prediction switch are referred to as the *tail EPN* and *head EPN* respectively. The pseudo components in the EPN are identified as X_t and X_h respectively, where X refers to the corresponding simulation model. The prediction time at every input port of a pseudo component X_t that has a label of the form E_j^{in}, is permanently held at a large predicted event-time value represented by the symbol ∞. An output port of every X_t that has the form E_j^{out} is linked to the input port of pseudo component Y_h in the head network that has a label of the form E_j^{in} via the prediction switch. The second input of prediction switch is the output labeled E_j^{out} at the component X_h. Thus, the prediction switch may be used to select prediction values from the tail EPN, hence making totally acyclic EPN, or it may restore the cycle in head network such that the prediction computations are circulated in head EPN.

For acyclic subcircuits, the prediction networks for those circuits constitute only head components. No prediction switch is required for such circuits either. An EPN of the circuit in figure 1 is as shown if figure 2.

3.2 Simulation Algorithm

A model can execute events up to the value of t_{win}. The event prediction network is responsible for computing values of t_{win} efficiently. Every behavior model, C^n, has a private simulation clock , $clock^n$. S_o^n, represents the logical value of the most recent event asserted at the output o of C^n. Assume that $t_{e_o}^n$ represents the time of the earliest event in the output event queue corresponding to the output o of C^n. Where the output event queue is empty, $t_{e_o}^n$ is set to ∞. Assume also, that t_{e_i} represents the time of the earliest event at input i of C^n. Our discussion of P^2EDAS is for cyclic subcircuits since their simulation is more complex. Thus, assuming that C^n is included in a cyclic subcircuit, the event prediction network will consist of C_t^n and C_h^n, representing the tail and head pseudo components respectively. Corresponding to every head and tail pseudo component of C^n, the predicted event time W_i^n and W_o^n are associated with input port i and output port o respectively. The simulation steps are as follows.

Initialization: Set all predictions values, W_i^n/W_o^n, $clock^n$, t_{win}^n to zero and appropriate inputs of tail EPN set to infinity.

Simulation Process at the Model: has the following steps

- Model execution: For a given component, C^n, identify any and all events, t_{e_i}, at the input ports such that $t_{e_i} < t_{win}^n$. The behavior model is executed for all such events, starting with the earliest event. For every event executed, the $clock^n$ value is advanced to the time of earliest next event that can be processed. The value of $clock^n$ will always be less than t_{win}^n. The newly generated output events, if any, are included in the output event queue of the affected outputs.

- Preempt output events using the semantics as defined in [9].

- Propagation of asserted events: For each output assert events , $t_{e_o}^n$, if the relationship $t_{e_o}^n \leq clock^n$ is true. These events are no longer preemptable.

- Updating the EPN parameters, namely the new time of events at inputs and logic value at outputs.

- The above steps are continually executed until $clock^n$ exceeds the simulation time of interest.

Execution of pseudo component: For each pseudo component, input port i, output port o with logical value S_o, the function $min\_delay(i, o, S_o)$ is the minimum of transition time to any other logical value that may be asserted at o. Every pseudo component computes output predicted event times, W_o^n at every output o, as

$$W_o = min\{t_{e_o}, (min(W_i, t_{e_i}) + min\_delay(i, o, S_o))\} \quad \forall i. \tag{1}$$

If the value is changed, the newly computed W_o values is propagated to the dependent pseudo components. The head pseudo component also computes t_{win}^n values as

$$t_{win}^n = minimum\{W_i\} \quad \forall i. \tag{2}$$

where i is the set of inputs to the component. When i is a primary input port, W_i is replaced by maximum simulation time in the computation of t_{win}^n. If t_{win}^n changes the behavior model is alerted. Correctness in the simulation requires that t_{win}^n be monotonic. Hence if W_i values in equation 2 have transient non-monotonic values, the event prediction network must wait until the W_i values are settled before they are used.

EPN prediction phases: Prediction values propagate from the tail EPN to the head. In the head EPN, the network filters the prediction values further by selecting and propagating minimum values. Transitivity in the cyclic subsystem requires that all paths that affect the input of components be considered in determining the correct event prediction time values. This

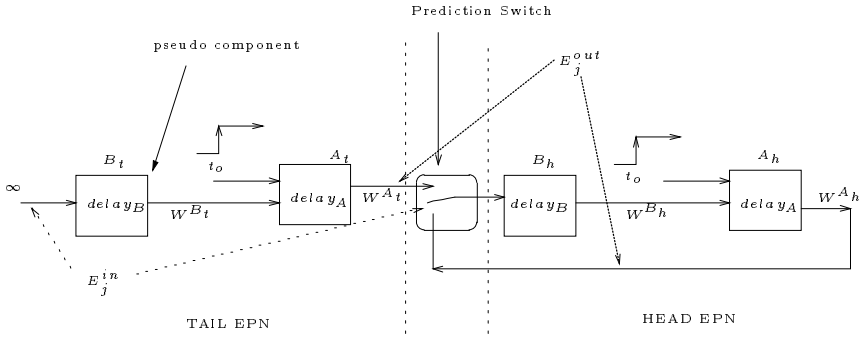

Figure 2: An EPN for Cyclic Circuit of figure 1

is realized by doing prediction on the head EPN with the cycle closed. Further, the prediction values in the head network goes through transient values before they settle to the correct prediction values (or steady state).

By restoring the cyclic structure of the circuit the prediction process may now behave as the inefficient prediction method described previously. This occurs when *stale head prediction values* are used i.e., prediction values not related to existing events. Stale prediction values exist when output preemption, scheduling and asserting of new events occurs, requiring that predicted event time on those outputs be recomputed. When prediction in the head EPN starts, the prediction values that can be trusted as being related to an actual event are those that are propagated from the tail network. To avoid the inefficient prediction through the use of stale head prediction values, the prediction in the head EPN is done in two steps.

- **Acyclic Head Prediction:** First, prediction with the prediction switch set to select tail EPN outputs. This prediction process is acyclic as occurs the tail network. Head prediction values are updated to reflect predicted event time based on actual events.
- **Cyclic Head Prediction:** A second prediction with prediction switch set to select cyclic head EPN values. In this phase the transitive dependence in the circuit is correctly captured. Thus the network restores its cyclic structure but using prediction values related to events, and proceeds to compute a stable state. Transitivity and non-zero loop delay, ensures that stable state with correct values will always be arrived at without deadlock.

As such, the prediction simulation in P^2EDAS uses the following distinct phases, namely:

1. **Tail Prediction Phase:** The tail network process changes in prediction values until no more changes occur.
2. **Head Prediction Phase:** Performed with the sub-phases as described above.
3. **Decision and Execution Phase:** When the head prediction is settled, all head pseudo components concurrently does the following:
   (a) Compute the t_{win}, determines if executable events exist and if so schedule the corresponding simulation model for execution.
   (b) Propagate head prediction values to pseudo components outside the subcycle.

3.2.1 P^2EDAS Prediction Monitors

The phase of a given pseudo component is determined by the phase associated with the subcycle. Thus each cyclic subsystem operates in its private phase without regard to the phase of another subcycle, implying an asynchronous simulation process. As such, a per subcycle *prediction monitor* is used to determine when the simulation in a given cyclic subsystem has reached steady state and to initiate transition to the next phase.

The prediction process uses the monitor as follows: For each prediction message sent between pseudo components in a given cyclic subcircuit, the sender informs the prediction monitor. For each message received and processed from pseudo component in the same subcycle, the pseudo component subcycle prediction monitor is informed. During a given prediction phase, the monitor determines steady state as, when all messages sent have been received and processed by the pseudo components. It then initiates transfer to the subsequent phase. Thus the prediction monitor non-intrusively determines the

Code Name	Circuit	Components	Signals
CPU	DLX Processor	19092	23749
TIMER	Timer	1063	1612
UROM	User ROM	852	984
SROM	System ROM	1502	1634
URAM	User RAM	12480	12581
SRAM	System RAM	12480	12581
Decoder	Address Decoder	168	266
Resolver	Distributed Resolver	32	259

Table 1: Subcircuits size in DLX computer

completion of prediction phases in the simulation process.

3.3 P^2EDAS Simulation

Consider that the simulation of the circuit in Figure 1 will be done using P^2EDAS. A prediction network of circuit is shown in Figure 2. Assuming an ordered evaluation from left to right, the computation at B_t is given as $W^{B_t} = min(\infty + delay_B, t^B_{e_i} + delay_B, t^B_{e_o})$. Ignoring the $\infty + delay_B$ term in equation (since it is constant and does not compete with the other terms) the equation for W^{B_t} may be written as $W^{B_t} = min(t^B_{e_i} + delay_B, t^B_{e_o})$.

This equation reveals that prediction values are not considered, only the time of events that exist at the model. As no events exist at the input and output of **B**, $t^B_{e_i} = t^B_{e_o} = \infty$ and hence $W^{B_t} = \infty$. At component A_t the prediction value is computed as $W^{A_t} = min(W^{B_t} + delay_A, t_0 + delay_A, t^A_{e_o})$. With $t^A_{e_o} = \infty$ the equation yields $W^{A_t} = t_o + delay_A$. This value is passed into the head network where computation yields $W^{B_h} = (delay_A + delay_B + t_0)$ and hence, $W^{B_h} > t_0$. For the cyclic prediction phase, this condition remains unchanged. Thus in a single pass over the prediction network it is determined that the event at **A** is executable, versus N_{iter} in the iterative approach.

4 Simulation Results and Conclusion

Distributed simulation of circuits was done on network of Pentium 90 workstations, linked by a 100MBit Ethernet network and running the Linux operating system. Each machine was equipped with 16 Megabyte of RAM and 100 Megabyte of swap space. Simulation of a DLX [10] computer system was performed. Table 1 shows the description of each major partition of the system.

Table 2 shows the distributed simulation time of the system for 3, 5, and 8 processors. The results shows reduction in the simulation time between 3 and 5 processors but an increase between 5 and 8 processors. This is due to the increase cost of communication as the number of compute nodes participating in the simulation increases. The single processor simulation runs for a significantly longer time. The reason is that, with the circuit loaded the simulator consumes 37 Megabytes of memory. Thus as the simulation progresses a significant amount of swapping occurs as the machine is equipped with only 16 Megabytes of RAM. This hi-lights an important benefit of distributed simulation. By using multiple machines, the resource needed for the simulation at each machine is greatly reduced.

Processors	1	3	5	8
Time (sec)	43200	4920	1800	2160

Table 2: Distributed simulation time for DLX Computer

In conclusion, we have introduce a new distributed, conservative, asynchronous simulation algorithm which shows scalable performance. The algorithm simulates arbitrary digital systems. It supports preemption, as occurs in digital circuit simulation, is deadlock free and is capable of exploiting all available deterministic parallelism in the simulation.

References

[1] J. Misra. Distributed Discrete-Event Simulation. *Computing Surveys*, 18(1):39–65, March 1986.

[2] D. Jefferson. Virtual Time. *ACM Transactions on Programming Languages*, 7(3):404–425, July 1985.

[3] J.V. Briner, J.L. Ellis, and G. Kedem. Breaking the Barrier of Parallel Simulation of Digital Systems. In *Proceedings of the 28th Design Automation Conference*, San Francisco, June 1991.

[4] K.M. Chandy and J. Misra. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Communcications of the ACM*, 24(4):198–206, April 1981.

[5] J.K. Peacock and Wang and Manning. Distributed Simulation Using a Network of Processors. *Computer Networks*, 3(1):44–56, 1979.

[6] Eric Debenedictis, Sumit Ghosh, and Meng-Lin Yu. An Asynchronous Distributed Discrete Event Simulation Algorithm for Cyclic Circuits using Data-flow Network. *IEEE Computer*, 24(6):21–33, June 1991.

[7] Narsingh Deo. *Graph Theory with Applications to Engineering and Computer Science*. Prentice Hall Inc, 1974.

[8] S. Chakradhar, A. Balakrishnan, and V. D. Agrawal. An exact algorithm for selecting partial scan flip-flops. In *Proc. 31st ACM/IEEE Design Automation Conference*, pages 81–86, June 1994.

[9] Sumit Ghosh and Meng-Lin Yu. A Preemptive Scheduling Mechanism for Accurate Behavioral Simulation of Digital Designs. *IEEE Computer*, 38(11):1595–1600, November 1989.

[10] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 1990.