

# On the Origin of VHDL's Delta Delays\*

SUMIT GHOSH

Department of Electrical and Computer Engineering, Castle Point on Hudson,  
Stevens Institute of Technology, Hoboken, New Jersey 07030, USA. Email: sghosh2@stevens-tech.edu

*Delta delays permeate the current VHDL 1076 standard and have been a source of much confusion and serious problems. The investigation in the paper turns to history and critically examines the contemporary scientific thinkings and technical developments, especially in modeling continuous electronic systems through discrete event techniques. This paper uncovers a plausible explanation, one that is logical and self-consistent, relative to the origin of BCL which, in turn, reveals the source of the problem with delta delays. The paper explains the scientific difficulties with delta delays and offers a solution to the problem that is practically realizable and, surprisingly, straightforward and simple.*

## AUTHOR'S QUESTIONNAIRE

1. The paper discusses materials/software for a course in: (1) Hardware description languages, (2) Logic design using VHDL, and (3) VHDL.
2. Students of the following departments are taught in this course: Electrical Engineering, Computer Engineering, Computer Science & Engineering, Computer Science.
3. Level of the course (year): senior-level undergraduates, first-year graduates, doctoral students engaged in research.
4. Mode of presentation: conventional lectures or web-based
5. Is the material presented in a regular or elective course? Both regular and as elective, depending on the program and academic institution.
6. Class or hours required to cover the material: 3–4 hours in a semester, towards the end of the semester.
7. Student homework or revision hours required for the materials: N/A
8. Description of the novel aspects presented in your paper: (1) New material not covered in any of the traditional VHDL classes or books or the VHDL Language Reference Manual, (2) Material essential to undertake accurate digital designs.
9. The standard text recommended in the course, in addition to author's notes: (1) *Hardware Description Languages: Concept and Principles* published by IEEE Press and written by the author, (2) *VHDL Language Reference Manual*, an IEEE standard and published by the IEEE Press, (3) Any of 25 or so books that present the syntax and use of VHDL.
10. The core of the paper relates strongly to the material covered in a traditional classroom.

## INTRODUCTION: THE NOTION OF TIME IN HDLS

TIMING is extremely important and one of the most important concepts in hardware design. In the digital design discipline, the correct functioning of systems is critically dependent on accurately maintaining the relative occurrence of events, thereby underscoring the importance of timing. Consider that an individual takes a set of shift registers, decoders, an ALU, and other hardware devices, and interconnects them in a haphazard manner, without any regard to timing. The resulting product is hardly useful. In contrast, when the same devices are put together by an expert designer, with their interactions carefully timed, the result is a powerful and sophisticated computer. Logically, therefore, timing is critical in hardware description languages (HDLs). In fact, a key difference between HDLs and the general-purpose programming languages such as Fortran, Pascal, Algol, C, or C++ lies in HDL's ability to model relative timing accurately. Barbacci [1] observes that the behavior of computer and digital systems is marked by sequences of actions or activities while Baudet *et al.* [2] view the role of time in HDLs as an ordering concept for the concurrent computations. Timing is manifest in HDLs through inertial and transport [3] delays, as attributes to help specify constraints between two or more signals, etc. For further details, the reader is referred to [4].

## MOTIVATION UNDERLYING THE NEED TO INTRODUCE DELTA DELAY

The original VHDL architects had introduced delta delays in VHDL, motivated by the following scenario. When a digital system consists of two components  $C^1$  and  $C^2$  with delays  $d_1$  and  $d_2$  respectively, such that  $d_1 \gg d_2$ , it should be fine to treat  $d_2$  as zero in the simulation and thereby

\* Accepted 29 August 2003.

enhance the simulation throughput. Thus, the user was encouraged to ignore the delay for  $C^2$  and assumed that the simulation would still generate accurate results. The architects soon discovered, upon implementation, that zero delays lead to uncertainty in simulation, especially for sequential systems, a fact that had already been well documented in the simulation literature. To resolve the problem, they introduced the notion of delta delays wherein the VHDL compiler, at compile time, first detects and then automatically replaces every instance of zero delay usage in a VHDL description, with a delta delay. While the VHDL 1076 language reference manual (LRM) does not provide much detail, a delta delay value is a non-zero time interval, selected by the VHDL compiler, such that an indefinite number of such intervals may be accommodated within a simulation timestep. The simulation timestep is clearly defined by those delays of components, such as  $d_1$ , that are significant. According to the architects, this new notion of delta delays is guaranteed to yield accurate simulation results, despite ignoring component delays such as  $d_2$  and without any further user intervention. In essence, the notion of delta delay imparted a false sense of trust among users in that VHDL had somehow solved the problem with zero delays and this tempted designers to use zero delays freely and without appropriate restraint. Not surprisingly, this has led to spurious errors that not only confront designers with inconsistent simulation results but defy insight into the cause of the error. To address these problems, a number of VHDL handbooks have included special sections on how to get around such problems in VHDL and, at leading international HDL conferences, during presentations, authors and members of the audience exchange information on how to avoid pitfalls in VHDL by writing tricky code. All of these clearly contradict the spirit of objectivity and logic in science and is of great concern since ICs and computers designed in VHDL will continue to fly planes, control nuclear reactors and power stations, and run the nation's financial system. The greatest conceptual difficulty with delta delay is that it constitutes a significant impediment to the concurrent execution of VHDL models on parallel processors in the future. While the error with delta delay is undeniable and seems readily

visible, it is intellectually puzzling how and when delta delay successfully bypassed a critical and rigorous scientific analysis. Since VHDL architects claim to have borrowed the concept directly from Conlan's BCL model of time, one must examine BCL to uncover the truth. Unfortunately, even the literature on BCL offers no direct reasons underlying the need to introduce it.

In the course of designing VHDL, the architects became aware that in a VHDL description of a digital system, the delays of different constituent components at different levels of hierarchy could differ by a significant margin. Thus, assuming two components  $C^1$  and  $C^2$  with delays  $d_1$  and  $d_2$  respectively, conceivably  $d_1 \gg d_2$ . Driven by the perception that the use of zero delays will limit the scheduler's workload and improve simulation throughput, they argued that it should be fine to treat  $d_2$  as zero. Thus, users were encouraged to ignore relatively insignificant delays of components such as  $C^2$  and were assured that the simulation would still generate accurate results. Examination of the language reference manual (LRM) [3] reveals, even today, that no restrictions are imposed, syntactically or semantically, on the use of zero values for inertial and transport delays. This was a misunderstanding, a serious error, and the problem of zero delay in simulation, especially in sequential systems, was well documented in the contemporary literature. In fact, the nature of the error is subtle and the misconception so serious that even as recently as 1996, Becker [5] states that the use of zero delays improves simulation performance.

It is not surprising, upon implementation, that VHDL architects were confronted with inconsistencies stemming from the use of zero delays. To understand the hows and whys, consider the RS latch in Fig. 1. Assume that both NANDs are 0 ns delay gates and that they execute concurrently in a structural description. Assume that the initial value at Q and Qb are both 1 and that the value at set and reset input ports are both 1. At simulation time 0, both gates execute and generate the following assignments: (1) a value 0 is assigned at Q at time  $0 + 0 = 0$  ns and (2) a value 0 is assigned at Qb at time  $0 + 0 = 0$  ns. Assume that there is a race between (1) and (2) and that (1) is executed infinitesimally earlier. As a result, the lower NAND gate is stimulated and it generates an

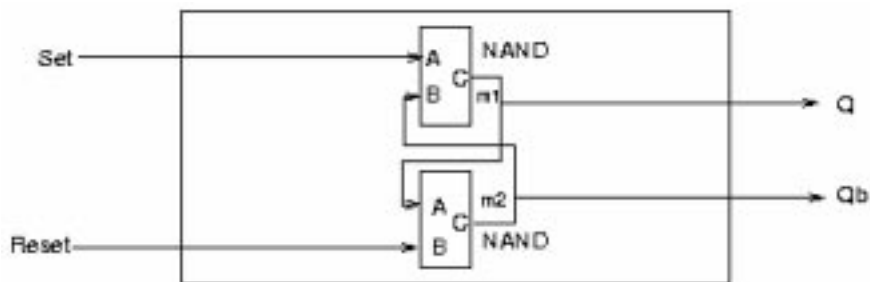


Fig. 1. Simulating a sequential circuit with zero-delay gates, in VHDL.

assignment: (3) a value 1 is assigned at Qb at  $0+0=0$  ns. Upon examining (2) and (3), both assignments are scheduled to affect the same port, Qb, at the same exact time 0 ns, one armed with a '0' and another armed with a value '1'.

The inconsistencies reflected a conceptual problem that was quite severe. The literature contained no scientific solution. Realizing that the entire VHDL effort was in jeopardy, the VHDL architects engaged in a desperate search and believed that they found their answer in Conlan's BCL model of time [6]. They imported BCL directly from Conlan and into VHDL, as acknowledged in [7] and recently reconfirmed [8] by Menchini [9]. In the process of importing BCL into VHDL, it is not clear whether the language designers performed any critical analysis or scientifically assessed the consequences. At least, none is reported in the literature.

The BCL time model was renamed delta delay in VHDL. Under the delta delay concept, the user is encouraged to specify a zero delay value for components whose delays are significantly small, relative to others in the simulation. At compile time, the VHDL compiler first detects and then automatically replaces every instance of zero delay usage in a VHDL description, with a delta delay. While the VHDL LRM does not provide much detail, a delta delay value is a non-zero time interval, selected by the VHDL compiler, such that an indefinite number of such intervals may be accommodated within a simulation timestep. The simulation timestep is clearly defined by those delays of components, such as  $d_1$ , that are significant. The user is guaranteed accurate simulation results despite ignoring component delays such as  $d_2$  and without any further user intervention. In essence, the notion of delta delay in VHDL appears to have mysteriously solved the problem of zero delays in simulation. The claim is critically examined subsequently in this paper.

### THE ORIGIN OF THE BCL TIME MODEL

The BCL model of time in Conlan [6] is dual-scale. While the 'real' time is organized into discrete instants separated by intervals of width equal to a single time unit, at the beginning of each time unit, there exists an indefinite number of computation 'steps' identified with integers greater than zero. BCL is non-intuitive, is not accompanied by any logical reasoning, and does not appear in any of the contemporary HDLs. To understand its origin, we are forced to play detective and turn to history to examine the contemporary thinkings and technical developments.

The 1960s witnessed an intense effort to adapt the traditional continuous simulation techniques for execution on the fast digital computers that were rapidly replacing analog computers. At first, the natural desire to reuse traditional practices resulted in models that described continuous

systems in the form of analog computer diagrams. Towards the end of the decade, there were several breakthroughs and a number of new mechanisms were developed, under the discrete event paradigm, for different application areas. One of these mechanisms, termed discrete event specification system (DEVS), was pioneered by Zeigler [10]. Consider a complex dynamic continuous system, that accepts discrete input values at its input ports from the external world at discrete time intervals,  $T_1, T_2, \dots, T_n$ . Corresponding to an input asserted at time  $T_j$ , the system may generate an output as a complex function of time. To accurately capture the dynamic output behavior, DEVS proposed organizing the output into a finer set of explicit time intervals,  $t_1, t_2, \dots, t_r$ , defined over the interval specified by  $T_j$  and  $T_{j+1}$ , and approximating the output segments over each of the finer time intervals through constant, piecewise trajectories. DEVS viewed the overall system behavior as follows. An external input event triggers a series of internal state changes that, in turn, generate a series of piecewise constant output values. The functions that define the internal state changes and the output segments in terms of the external input and internal state values, constitute the DEVS formalism [10]. Clearly, while the time intervals delimited by  $T_1, T_2, \dots$  are defined by the system, the choice of  $t_1, t_2, \dots$  is up to the DEVS model developer and dictated by the desired accuracy in the system characterization.

Contemporaneously, in the digital systems discipline, researchers observed that for some digital circuits, a single external input signal triggers multiple changes in the output, caused by the racing of signals through gates. The subject matter was classified under hazards [11]. Both static and dynamic hazards were demonstrated and while combinatorial and sequential systems were equally susceptible, the latter triggered particularly complex behavior stemming from unequal gate delays, etc.

Thus, the invention of BCL may have been motivated by either one or both of two possibilities. First, BCL may have been a generalization of DEVS, intended to serve as a common timing framework to facilitate the simulation of both discrete and continuous sub-systems in Conlan. While the discrete sub-system would be simulated utilizing standard discrete event simulation technique, simulation of the continuous sub-system would fall under the jurisdiction of DEVS. Perhaps, Piloty and Borriore had in mind an environment to simulate a system wherein control would alternate between discrete event simulation of the discrete sub-system and DEVS execution of the continuous sub-system. The dual time-scale of BCL would provide a logical and structural mechanism to relate the signals, in time, between the two sub-systems. Intuitively, this is most likely the truth. Piloty and Borriore's thinking may have been as follows. The time scale for the discrete sub-system is the 'real' time and it also corresponds to

that of the external input to the continuous sub-system. The time-scale for the continuous sub-system is much finer and are termed 'steps'. At the time intervals equal to 'step', the internal state and output functions are executed as outlined in DEVS. Clearly, the model developer is responsible for determining the 'step' value, based on the desired accuracy.

However unlikely, there is a second possibility. Piloty and Borrione may have wanted to extend the DEVS idea to model pure discrete digital systems, intending to capture complex hazard behaviors. This appears unlikely for simple analysis reveals that the time-scale for characterizing a hazard is not significantly different from that of the external input stimuli. Piloty and Borrione would not have chosen to use the phrase 'indefinite number of steps'.

If the inventors of BCL indeed intended the finer time-scale namely 'steps' to enable the simulation of continuous sub-systems, clearly its use to simulate discrete components with zero delays in VHDL constitutes an inappropriate adaptation. In BCL, the continuous sub-system under DEVS and the discrete sub-system simulations were assumed to be distinct. In contrast, in VHDL, the delta delay micro time-scale is erroneously forced to coexist and cooperate with the macro time-scale within a single simulation system. Last, in BCL, the user is required to determine the 'step' value based on the desired accuracy. In contrast, VHDL architects encourage users to abandon the small delay values in favor of zero delay, and attempt to incorporate in the VHDL compiler an automated means of determining a value for the micro time-step. The irony is that, immediately after the user has abandoned  $d_2$  type values, the VHDL compiler must turn around and reinvent delta delay values that are comparable to  $d_2$ , but without the benefit of any scientific basis. Clearly, there is mismatch between the philosophy underlying BCL and VHDL's needs under delta delay.

It is pointed out that the lack of a principle to unify the traditional discrete event simulation of discrete sub-systems with the DEVS approach for continuous sub-systems, continues to resist to this day a uniform simulation approach to a hybrid system consisting of both discrete and continuous sub-systems. However, recent research has led to the discovery of a new principle—generalized discrete event specification system (GDEVS) [12–15], that promises to achieve this unification in the very near future.

### DIFFICULTIES WITH VHDL'S DELTA DELAYS

According to the original VHDL architects [7] and as recently reconfirmed [8] by Menchini [9], VHDL's model of time is derived from the BCL time model in Conlan [6]. In the BCL time model, the real time is organized into discrete instants

separated by a single time unit and the beginning of each time unit contains an indefinite number of computation 'steps' identified with integers greater than zero. The discrete instants and computation steps correspond to the macro- and a micro-time scale in VHDL, constituting the notion of delta delay. VHDL permits signal assignments with zero delays, i.e. the value is assigned to the signal in zero macro-time units but some finite, delta, micro-time units. The actual value of delta is inserted by the VHDL compiler, transparent to the user.

The first difficulty with delta delay is conceptual. Given that a host computer is a discrete digital system, it cannot accommodate an indefinite number of steps within a finite time unit. Although the individual computation 'steps' must imply some hardware operation, they do not correspond to discrete time instants that are utilized by the underlying discrete event simulator to schedule and execute the hardware operations. Thus, the computation 'steps' may not be executed by the simulator and, as a result, they may not serve any useful purpose. It is also noted that, fundamentally, in any discrete event simulation, the timestep or the smallest unit through which the simulation proceeds, is determined by the fastest sub-system or process. For accuracy, this requirement is absolute and cannot be transcended. Assume that this timestep is  $T_m$ . If, instead of  $T_m$ , a timestep  $T$  is used deliberately ( $T > T_m$ ), the contributions of the fastest sub-system or process cannot be captured in the simulation, leading to errors in interactions, and eventually incorrect results.

The second difficulty with delta delay is that the VHDL language reference manual [3] does not state how a value for the delta is selected. This is an important question since VHDL may return different results corresponding to different choices of the delta, as illustrated through Fig. 2.

Figure 2 presents a signal waveform. Assume that the value of delta is 1 ps. When the current simulation time is either 1 ns or 2 ns, VHDL safely returns the value 0 for the signal value. However, where the delta value is 5 ps, VHDL will return the value 0 corresponding to the current simulation time of 2 ns but fail to return a definite value corresponding to the current simulation time of 1 ns. Since the signal waveform is realized at

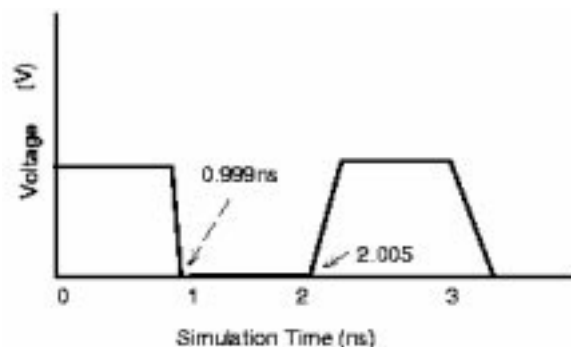


Fig. 2. Impact of the choice of delta value on simulation results.

runtime, i.e. as the entities execute during simulation, and as the VHDL compiler must select a value for the delta delay at compile time, it is difficult to ensure the absence of ambiguous results.

The third difficulty is that, fundamentally, any attempt to simulate an asynchronous circuit with zero-delay components, under discrete event simulation, is likely to lead into ambiguity. In its aim to simulate digital designs with zero-delay components through delta delays, VHDL incurs the same limitation. Consider, for example, the RS latch shown earlier in Fig. 1 and assume that both NANDs are 0ns delay gates and that they execute concurrently in a VHDL structural description. Assume that the initial value at Q and Qb are both 1 and that the value at set and reset input ports are both 1. At simulation time 0, both gates execute and generate the following assignments: (1) a value 0 is assigned at Q at time  $0+0=0$ ns and (2) a value 0 is assigned at Qb at time  $0+0=0$ ns. Assume that there is a race between (1) and (2) and that (1) is executed infinitesimally earlier. As a result, the lower NAND gate is stimulated and it generates an assignment. (3) a value 1 is assigned at Qb at  $0+0=0$ ns. Upon examining (2) and (3), both assignments are scheduled to affect the same port, Qb, at the same exact time 0ns, one armed with a '0' and another armed with a value '1.'

The fourth difficulty is that one can construct any number of example scenarios in VHDL where the result is inconsistency and error. Consider the process, PROC1, shown below which has an inconsistency with delta delays in VHDL. While not critical to this discussion, it is pointed out that the process PROC1 does not include a sensitivity list which is permitted by the VHDL language [3]. As an example usage of a process without a sensitivity list, the reader is referred to page 57 of [3].

```
architecture X of Y is
  signal a, b: resolve BIT .. ;
begin
  PROC1: process
    variable c: int;
  begin
    for i in 0 to 1000 loop
S1:    a <= b + c;
S2:    b <= a + c;
    end loop;
  end process;

  PROC2: process
    variable d: int;
  begin
    for i in 0 to 7 loop
S3:    a <= b + d;
S4:    b <= a + d;
    end loop;
  end process;
end X;
```

The statements S1 and S2 are both zero delay signal assignments. While S1 updates the signal 'a' using the value of signal 'b' and the variable, 'c,' the statement S2 updates the signal 'b' using the value of the signal 'a' and the variable 'c.' To prevent ambiguity of assignments to the signals 'a' and 'b,' the VHDL compiler inserts, at compile time, a delta delay of value delta1 say, to each of S1 and S2. Thus, S1 is modified to:  $a \leftarrow b + c$  after delta1, and S2 is modified to  $b \leftarrow a + c$  after delta1. For every iteration, the subsequent assignments to 'a' and 'b' are realized in increments of delta1. That is, first (NOW + delta1), then (NOW + delta1 + delta1), and so on. These are the micro time steps in the micro-time scale and we will refer to them as delta points. Between the two consecutive macro time steps, the VHDL scheduler may only allocate a maximum but finite number of delta points which is a compile time decision. Conceivably, the designer may choose a value for the number of iterations such that, eventually, the VHDL scheduler runs out of delta points. Under these circumstances, VHDL will fail. Thus, the idea of signal deltas, transparent to the user, is not implementable.

Fifth, the notion of delta delays, in its current form, poses a serious inconsistency with VHDL's design philosophy of concurrency. Consider the case where the processes PROC1 and PROC2, by definition, are concurrent with respect to one other. The two sets of statements—{S1, S2} in PROC1 and {S3, S4} in PROC2, both affect the signals 'a' and 'b' and 'resolve' constitutes the resolution function, as required by VHDL. The statements S1, S2, S3, and S4 are all zero delay signal assignments, so delta delays must be invoked by the VHDL compiler. Since the dynamic execution behavior of processes are unknown *a priori*, the VHDL compiler may face difficulty in choosing appropriate values for the delta delay in each of the processes. In this example, however, logically, the VHDL compiler is likely to assign a very small value for the delta delay (say delta1) in PROC1, given that it has to accommodate 1001 delta points. In contrast, the VHDL compiler may assign a modest value for the delta delay (say delta2 where  $\text{delta2} \gg \text{delta1}$ ) in PROC2, given that only 8 delta points need to be accommodated. As stated earlier, assignments to the signals 'a' and 'b' will occur from within PROC1 at (NOW + delta1), (NOW + delta1 + delta1), and so on. From within PROC2, assignments to the signals 'a' and 'b' will occur at (NOW + delta2), (NOW + delta2 + delta2), etc. By definition, a resolution function resolves the values assigned to a signal by two or more drivers at the same instant. Therefore, here, 'resolve' will be invoked only when  $(\text{NOW} + m * \text{delta1}) = (\text{NOW} + n * \text{delta2})$ , for some integer values 'm' and 'n.' In all other cases, assignments to the signals 'a' and 'b' will occur either from within PROC1 or PROC2. Thus, the values of the signals 'a' and 'b,' from the

perspectives of processes PROC1 and PROC2 are uncoordinated, implying ambiguity and error.

There is a misconception that, during VHDL simulation, one can turn off the advancement of time, i.e. macro-time step, then execute a number of micro-steps, and then resume the normal advancement of time. This is incorrect and explained as follows. Fundamentally, a simulator is a sophisticated program whose sole objective is to execute events. A simulation terminates when all events have been executed and the number of outstanding events is nil. Simulation, in turn, critically depends on the notion of time to order the events for, without order, causality may be violated leading to erroneous results. Therefore, any suggestion to turn off the advancement of time, even temporarily, is equivalent to terminating the simulation. In fact, in any simulation, every activity, whether labeled macro-event or micro-event, must be logically ordered by time such that the causal dependence remains in effect and correct results are produced. In essence, there is no computational savings of any kind through the use of dual time-scales.

#### A SIMPLE SOLUTION TO THE PROBLEM OF DELTA DELAYS IN VHDL

The solution to the original problem that set the VHDL architects on their journey to delta delay, comes in two parts. The first is the elimination of delta delay in its present form from the VHDL LRM and prohibiting the use of zero delays. Delta delays are unnatural, unnecessary, and error prone. Second, users must simply be encouraged to specify the exact delays of components at any level of hierarchy, in terms of the universal time, regardless of whether it is relatively too small or too large. The VHDL execution environment, like any other event driven simulation system [16], will generate correct results without any additional resource or external intervention. Of course, the dynamic range of the delays, i.e. the ratio of the largest to the smallest delay value, must be accommodated by the VHDL compiler and correctly interpreted in the simulation environment. This may require a simple reorganization of the integer representation in the host computer, perhaps concatenate two or more 32-bit words or 64-bit words for use in representing simulation time. If  $d_1$  and  $d_2$  represent large and small propagation delay values respectively, it does not matter how large  $d_1$  is or how small  $d_2$  is, relative to each other. For, in event driven simulation, the scheduler jumps from one event to the next logical event, following the causal dependence chain, regardless of the amount of jump, measured in terms of time. Thus, the simulation execution time is influenced primarily by the causal dependence chain, reflected in the differences of the time values of the events, and not by the absolute values of  $d_1$  and  $d_2$ .

To gain insight into the philosophy behind the

simple solution requires an understanding of the concept of universal time [4]. At a given level of abstraction, although each entity, by virtue of its independent nature, may have its own notion of time, for any meaningful interaction between entities A and B, both A and B must understand at the level of a common denominator of time. This is termed 'universal time,' assuming that the system under consideration is the universe. Otherwise, A and B will fail to interact with each other. The universal time reflects the finest resolution of time among all of the interacting entities. However, the asynchronicity manifests as follows. Where entities A and B interact, between their successive interactions, each of A and B proceed independently and asynchronously. That is, for A, the rate of progress is irregular and uncoordinated and reflects lack of precise knowledge of that of B and vice-versa. At the points of synchronization, however, the time values of A and B must be identical. Where entities X and Y never interact, their progress with time is absolutely independent and uncoordinated with one another and the concept of universal time is irrelevant.

Thus, at any given level of abstraction in hardware, the entities must understand events in terms of the universal time and this time unit sets the resolution of time in the host computer. The host computer, in turn, executes the hardware description, expressed in a HDL, of a digital system. Consider a hardware module A with a unique clock that generates pulses every second connected to another hardware module B whose unique clock rate is a millisecond. Figure 3 shows a timing diagram corresponding to the interval of length 1 s between 1 s and 2 s. Figure 3 superimposes the 1000 intervals each of length 1 ms corresponding to the clock of B. Clearly, A and B are asynchronous. Module A is slow and can read any signal placed on the link every second. If B asserts a signal value  $v_1$  at 100 ms and then another value  $v_2$  at 105 ms, both within the interval of duration 1 second, A can read either  $v_1$  or  $v_2$ , but not both. The resolution of A, namely 1 second, does not permit it to view  $v_1$  and  $v_2$  distinctly. Thus, the interaction between A and B is inconsistent. If A and B were designed to be synchronous, i.e. they share the same basic clock, A would be capable of reading every millisecond and there would be no difficulty. In reality, microprocessors that require substantial time to generate an output of a software program are often found interfaced asynchronously with hardware modules that generate results quicker. In such situations, the modules and the microprocessor understand the universal time, i.e. they are driven by clocks with identical resolutions although the phases of the clocks may differ thereby causing asynchrony.

Thus, the host computer which is responsible for executing the hardware descriptions corresponding to the entities, must use the common denominator of time for its resolution of time. When the host computer is realized by a

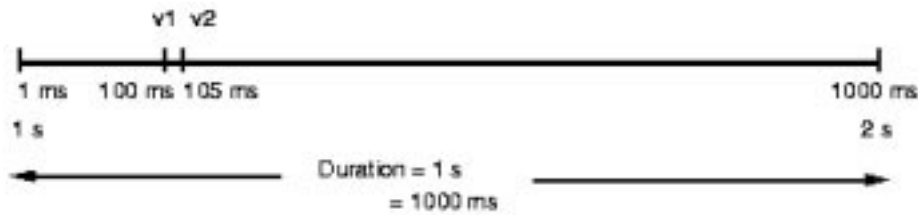


Fig. 3. The concept of universal time.

uniprocessor, the underlying scheduler implements this unit of time. When the host computer is realized by multiple independent processors, each local scheduler, associated with every one of the processors, will understand and implement this unit of time.

## CONCLUSIONS

This paper has traced the VHDL architects' journey into the world of delta delay including the original need for zero delay usage that evolved from a misconception that zero delays enhance simulation throughput without any penalty, the subsequent difficulties with the VHDL implementation of zero delay, the adapting of Conlan's BCL model of time into VHDL as delta delay without a clear understanding of the consequences, and

the problems that confront VHDL today. This paper has traced the origin of BCL to the goal of relating the results of discrete event simulation of discrete sub-systems with those of DEVS simulation of continuous sub-systems, for a given system, through an underlying timing framework. This paper has presented a simple solution to the problem that involves the elimination of zero delay usage and the specification of actual component delay values in terms of universal time. Presently, the author is pursuing the integration of GDEVS with VHDL that promises to yield a practically realizable approach to the accurate simulation of both discrete and continuous elements of a digital system, within a single uniform environment.

*Acknowledgment*—The author sincerely thanks Prof. Norbert Giambiasi of the University of Marseilles, France, and Prof. Bernie Zeigler of the University of Arizona, Tucson, for their valuable insights.

## REFERENCES

1. M. R. Barbacci, A comparison of register transfer languages for describing computers and digital systems, *IEEE Transactions on Computers*, C-24(2) Feb. 1975, pp. 137–150.
2. G. M. Baudet, M. Cutler, M. Davio, A. M. Peskin and F. J. Rammig, the relationship between hdl and programming languages, in *VLSI and Software Engineering Workshop*, Port Chester, NY (1982) pp. 64–69.
3. *IEEE Standard VHDL Language Reference Manual, ANSI/IEEE Std 1076–1993*, IEEE, Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA (1994).
4. Sumit Ghosh, *Hardware Description Languages: Concepts and Principles*, IEEE Press, New Jersey (2000).
5. M. Becker, Faster Verilog simulations using a cycle based programming methodology, *Proc. 1996 International Verilog HDL Conference*, Santa Clara, California, February 26–28 1996, pp. 24–31.
6. R. Piloty and D. Borrione, The Conlan Project: concepts, implementations, and applications, *IEEE Computer*, C-24(2) 1985, pp. 81–92.
7. M. Shahdad, R. Lipsett, E. Marschner, K. Sheehan and H. Cohen, VHSIC Hardware Description Language, *IEEE Computer*, CAD-6(4) 1985, pp. 94–103.
8. Sumit Ghosh, Fundamental principles of modeling timing in hardware description languages for digital systems, *Proc. Int. HDL Conference (HDLCON'99)*, Santa Clara, CA, April 6–9, 1999, pp. 30–37.
9. Paul Menchini, Comments during open discussion in the session on timing in hardware description languages, *Int. HDL Conference (HDLCON'99)*, April 6–9, 1999.
10. Bernie Zeigler, *Theory of Modeling and Simulation*, John Wiley & Sons, New York (1976).
11. Saburo Muroga, *Logic Design and Switching Theory*, John Wiley & Sons, New York (1979).
12. Norbert Giambiasi, Bruno Escude and Sumit Ghosh, GDEVS: a generalized discrete event specification for accurate modeling of dynamic systems, in *AIS'2000*, Tucson, AZ, March 2000.
13. Bruno Escude, Norbert Giambiasi and Sumit Ghosh, GDEVS: a generalized discrete event specification for accurate modeling of dynamic systems, *Transactions of the Society for Computer Simulation (SCS)*, 17(3) September 2000, pp. 120–134.
14. Norbert Giambiasi, Bruno Escude, and Sumit Ghosh, Generalized discrete event specifications: coupled models, *Proc. 4th World Multi-Conference on Systemics, Cybernetics And Informatics (SCI 2000)* and *5th Int. Conf. Information Systems, Analysis And Synthesis (ISAS 2000)*, July 23–26 2000.
15. Bruno Escude, Norbert Giambiasi, and Sumit Ghosh, Coupled modeling in generalized discrete event specifications (GDEVS), *Transactions of the Society for Computer Simulation (SCS)*, *Special Issue on Recent Advances in DEVS Methodology*, December 2000.

16. Sumit Ghosh and Tony Lee, *Modeling and Asynchronous Distributed Simulation: Analyzing Complex Systems*, IEEE Press, New Jersey (2000).

**Sumit Ghosh** is the Hattrick Chaired Professor of Information Systems Engineering in the Department of Electrical and Computer Engineering at Stevens Institute of Technology in Hoboken, New Jersey. Prior to Stevens, he had served as the associate chair for research and graduate programs in the Computer Science and Engineering Department at Arizona State University. Before ASU, Sumit had been on the faculty of Computer Engineering at Brown University, Rhode Island, and even before that he had been a member of technical staff (principal investigator) of VLSI Systems Research Department at Bell Laboratories Research (Area 11) in Holmdel, New Jersey. He received his B. Tech. degree from the Indian Institute of Technology at Kanpur, India, and his M.S. and Ph.D. degrees from Stanford University, California. He is the primary author of five reference books: *Hardware Description Languages: Concepts and Principles* (IEEE Press, 2000); *Modeling and Asynchronous Distributed Simulation of Complex Systems* (IEEE Press, 2000); *Intelligent Transportation Systems: New Principles and Architectures* (CRC Press, 2000; First reprint 2002); and *Principles of Secure Network Systems Design* (Springer-Verlag, 2002; Translated into Chinese, 2003–2004); and *Algorithm Design for Networked Information Technology Systems: Principles and Applications* (Springer-Verlag, due out October 2003). As the first vice president of education in the Society for Computer Modeling and Simulation International (SCS), he is tasked to develop comprehensive graduate and undergraduate degree programs in modeling and simulation and an accreditation process. Sumit organized a NSF-sponsored workshop titled, *Secure Ultra Large Networks: Capturing User Requirements with Advanced Modeling and Simulation Tools (ULN'03)* (with Prof. Bernard Zeigler of Univ. Arizona and Prof. Hessam Sarjoughian of ASU) at Stevens Institute of Technology, May 29–30, 2003. Details on his research pursuits may be found at <http://attila.stevens-tech.edu/~sghosh2>.