

# On Simulating Parallel Algorithms with VHDL

Stavros Souravlas, Efthimios Kotsialos, Athanasios Margaritis, and Manos Roumeliotis  
 University of Macedonia, Applied Informatics Department,  
 156 Egnatia St. 54006, Thessaloniki, GREECE  
 email:{sourstav,ekots,amarg,manos}@uom.gr

## I. ABSTRACT

Hardware Description Languages (HDL) like VHDL are widely used to design and simulate with programmable logic devices. Simulation of very large scale integrated digital systems (VLSI) is of great importance as it assures system correctness and maximizes system performance ([1], [2], [4]). The utilization of parallel simulation introduces the problem of how to partition the logic gates and system behaviors of the circuit among the available processors in order to obtain maximum speedup. This paper presents how a series of sequential statements like VHDL *processes* and *signals* can be used to simulate parallel message broadcasts. Processes and signals are the most important VHDL parts for simulation.

A VHDL **process** is a construct for embodying algorithms. A process is used to describe the behavior of a part of the design using a series of sequential statements. In a parallel processor system, communication is performed via message passing; at any instance of time, a number of processors exchange messages, that is, many processes are running. In our modeling approach, a set of parallel running processes will be mapped to a set of VHDL processes. A VHDL process has two states: active or suspended. The processes to be executed are put into a process queue. Once a process is selected for execution, it is put to active state and all the sequential statements inside are executed one by one. As soon as the last statement is executed, the process reaches the "suspended" state. A process becomes active again only when there is a change of value in any of its *driving signals*. A **driving signal** is a special VHDL element that can cause a process to execute. A process associated with a signal is executed if the signal changes value. The list of signals that may cause a process to run is called **sensitivity list**. It is obvious that more than one signal may cause a process to execute and there may be many processes running, triggered by the same signal. It is important to note that VHDL signals do not change value like variables do in a high-level programming language. A new value is *scheduled* to be assigned to a signal one

**delta delay** after the current simulation time. This means that if the simulation time is  $T_c$  then a signal scheduled to change value will not be updated until  $T_{c+\delta}$ . Delta delays can cause the simulator to repeat the series of statements inside a process more than once.

We now describe how we can use the VHDL tools to simulate a parallel message passing algorithm. The VHDL code description will be presented in the paper. Assume that a communication scheduling algorithm that allows pipeline based processing of messages has been developed. Also, assume that the pipeline strategy has three stages, *stg0*, *stg1*, and *stg2*. To make a VHDL description for the message passing problem we first need to activate transfer processes by assigned transfer signals and then to identify a set of variables that act as stage registers.

When a transfer process executes, a message from a node is passed to the staging register of the pipeline. Suppose that there are 6 parallel processes that can execute at a time:  $P_0, P_1, P_2, \dots, P_5$ . Each of the transfer processes is sensitive to a change of value that occurs in a signal of an originally defined array of bit or logic\_vector type. We name this array *transfer\_processor\_list*. In other words, this processor list is a bit vector with its values being 1 if the corresponding processor is to send a message during a simulation cycle, or 0 if the corresponding processor remains idle. The decision on the processes that will execute is taken by the scheduling algorithm that successively updates the bit vector. A new simulation cycle begins when the *transfer\_processor\_list* gets a new value or when a previously executed process suspends. Each simulation cycle consists of a number of *delta cycles*. Each delta cycle updates the value of the stage registers.

Figure 1 illustrates the simulation cycles created and the changes occurring in each cycle, for the six processes. Initially, all signals that may activate processes are set to zero. Assume that the algorithm decides that processes  $P_0$  and  $P_1$  must send a message at a time. This means that the 6-bit signal *transfer\_processor\_list* transitions to "110000" (the first two bits change value) during the first simulation cycle. This causes two transfer processes

Sim cycle	$\delta$	$P_0$	$P_1$	$P_2$	$P_3$	$P_4$	$P_5$	stg0	stg1	stg2
1	+0	1	1	0	0	0	0	NULL	NULL	NULL
	+1	1	1	0	0	0	0	$P_0, P_1$	NULL	NULL
2	+0	0	0	1	1	0	0	$P_0, P_1$	NULL	NULL
	+1	0	0	1	1	0	0	$P_0, P_1$	stg0	NULL
	+2	0	0	1	1	0	0	$P_2, P_3$	$P_0, P_1$	NULL
3	+0	0	0	0	0	1	1	$P_3, P_3$	$P_0, P_1$	NULL
	+1	0	0	0	0	1	1	$P_3, P_3$	$P_0, P_1$	stg1
	+2	0	0	0	0	1	1	$P_3, P_3$	stg0	$P_0, P_1$
	+3	0	0	0	0	1	1	$P_4, P_5$	$P_2, P_3$	$P_0, P_1$

Fig. 1. Simulation Cycles & Signal updates

$P_0$  and  $P_1$  to execute. After a  $\delta$  cycle, the first stage of the pipeline (which is modeled by stage register stg0) is updated with its new value, that is, it handles the proper data for the messages from  $P_0$  and  $P_1$ . In Figure 1, we show that by assigning the values  $P_0$  and  $P_1$  to stg0.

The scheduling algorithm keeps on generating interprocessor communications. Assume that  $P_2$  and  $P_3$  are scheduled to execute. The value of the signal *transfer\_processor\_list* changes from "110000" to "111100" (see Figure 1) during simulation cycle 2. Stage register stg1 will now be responsible for the execution of  $P_0, P_1$  (previously handled by stg0) while stage register stg0 is responsible for the newly triggered processes. This is shown in Figure 1, by assigning the value stg0 to stg1 and the values  $P_2, P_3$  to stg0. These value changes occur within two more  $\delta$  cycles.

Finally,  $P_2$  and  $P_3$  are scheduled to execute. The *transfer\_processor\_list* transitions form "111100" to "111111". The last stage register stg2 now becomes responsible for the communications previously handled by stg1, while the stage register stg1 is responsible for the communications previously handled by stg0. These changes require three  $\delta$  delays +0,+1,+2. During the last  $\delta$  delay, the stage registers have been assigned their proper value, that is,  $stg0 \leftarrow P_4, P_5$ ,  $stg1 \leftarrow P_2, P_3$ , and  $stg2 \leftarrow P_0, P_1$

At this point, it is important to note that the value of  $\delta$  delay is normally assigned by the programmer and its value may vary between several executions of the simulation or between value assignments to the stage registers. Factors that affect the value of  $\delta$  are: (1) the duration of transfer tasks, and (2) link availability. To assign a proper  $\delta$  delay, we use the VHDL key-word **after**. In our previous example, if we wish to trigger  $P_0$  and  $P_1$  after 20ns (recall that in this case *transfer\_processor\_list* transitions to "110000"), we should write:

tps="11000" **after** 20ns;

Such  $\delta$  values are very important for VHDL parallel simulation. When describing parallel behaviors with VHDL sequential statements, the sequence of these statements is of no importance, unless we force them to be executed in specified times.

This paper is the beginning of a research on how to model parallel algorithms and applications with a hardware description language like VHDL. We initially focus on producing the VHDL source that describes the necessary operations required by a parallel algorithm. Our key idea is to map parallel executing processes to VHDL processes and use signals drivers for process activation. The signal values are updated by the message passing algorithm.

## REFERENCES

- [1] Dragos Lungeanu, C.J. Richard Shi, "Parallel and Distributed VHDL Simulation", *IEEE Design, Automation and Test in Europe (DATE '00)*, pp. 658, March 27 - 30, 2000 Paris, France.
- [2] Dragos Lungeanu, C.J. Richard Shi, "Distributed Event-Driven Simulation of VHDL-SPICE Mixed-Signal Circuits", *IEEE International Conference on Computer Design: VLSI in Computers & Processors (ICCD'01)*, pp. 0302, September 2001.
- [3] Wilco van Hoogstraeten, Henk Corporaal, "ADVISE- Performance Evaluation of Parallel VHDL Simulation", *IEEE 30th Annual Simulation Symposium*, p. 146-156, 1997 IEEE.
- [4] Tun Li, Yang Guo, Si-Kun Li, "Design and Implementation of a Parallel Verilog Simulator: PVSIM", *IEEE 17th International Conference on VLSI Design*, p. 329, January 05 - 09, 2004 Mumbai, India.