ESTIMATION OF PATH DELAYS USING VHDL LOGIC SIMULATION

Miljana Sokolović, *Faculty of Electronic Engineering, University of Niš*
Dejan Maksimović, *Philips Semiconductors AG, Zurich*

Abstract – *This paper presents a VHDL based method for timing simulation within a VHDL logic simulator framework. This method is reminiscent to the one published in* [1, 2] *and enables the logic simulator to evaluate the longest and the shortest path delays of all signals in the circuit with only one run of the logic simulator. Timing simulation is performed at simulation time t=0 at the cost of a negligible increase of CPU time needed for the simulation. Results of the timing simulation of the ISCAS'85 benchmark circuits with a VHDL simulator are presented that prove that the proposed method is extremely efficient and appropriate for interactive use in the early phases of the design process where timing analysis needs to be repeated as the circuit design is optimized or refined.*

1. INTRODUCTION

Circuit operating frequency is one of the most important user requirements to digital integrated circuit designer. The maximum operating frequency is determined by the delay of the longest data path in the circuit, while the minimum duration of the clock pulse is determined by the delay of the shortest data path in the circuit. Circuit delays are usually being extracted with timing analysis programs [3-5]. When timing analysis shows that timing requirements are not being met, the designer must redesign the circuit and/or perform the delay minimization techniques on critical paths [6]. To avoid circuit redesign, the designer needs to carry out maximal delay estimation as early as possible in the design process.

It is shown in [1, 2] that a versatile logic simulator can produce an early estimation of circuit delays during the logic verification of the design in quite acceptable CPU time. In [2] a method is proposed that enables logic simulator to perform timing simulation. The method evaluates the longest path propagation delay for all signals in the circuit with only one run of the logic simulator.

In this paper an efficient implementation of the enhanced method in VHDL is presented. It is based on use of composite type signals that consist of signal logic value and needed timing data. To test the performance of the proposed method, longest and shortest path delays in ISCAS'85 benchmark circuits are evaluated using a VHDL simulator. The presented results prove an excellent performance of the method.

The paper is organized as follows. In section 2 the method proposed in [2] is described in short. Section 3 deals with enhanced method implementation in VHDL. In section 4 experimental results are presented.

2. TIMING SIMULATION WITH LOGIC SIMULATOR

When a digital circuit is simulated for one specific input vector, the instant time when the first activity occurs on a signal determines the shortest path delay to that particular signal for the given input vector. The time instant when the last activity occurs on a signal determines the longest path. In order to obtain the worst-case delays of both rising and falling signal transitions the circuit has to be simulated for all input vectors. Therefore, 2^n circuit simulations have to be carried out, where n is the number of primary inputs. It is obvious that this approach is not feasible when n becomes large.

In order to evaluate the longest and the shortest path delays to **all** the signals in the circuit with only **one** simulation of the circuit, simultaneous simulation of the circuit for all input vectors is necessary. To enable logic simulators to perform such a simulation, eight additional signal attributes need to be introduced to store the information of the arrival of the transitions at the signal and of the shortest and the longest path delays to the signal [2]. Having a signal named *S*, these four attributes are:

d1mn(S) - the shortest path delay of rising transition at *S*,
d0mn(S) - the shortest path delay of falling transition at *S*,
arr1mn(S) - rising transition of the shortest path arrival flag,
arr0mn(S) - falling transition of the shortest path arrival flag,
d1mx(S) - the longest path delay of rising transition at *S*,
d0mx(S) - the longest path delay of falling transition at *S*,
arr1mx(S) - rising transition of the longest path arrival flag,
arr0mx(S) - falling transition of the longest path arrival flag.

It is assumed that the circuit is described and simulated at structural level of abstraction and that the minimal and maximal delays of all building blocks for both rising and falling edge are known. At the start of the simulation the circuit is stimulated with both rising and falling transitions at all primary inputs. As the transitions propagate from primary inputs towards primary outputs, the gate delays are accumulated along the paths. Signal attributes are accessed from the processes in the gate models and their values are dynamically updated. Once the circuit activity is exhausted, the shortest and the longest path delays are available in signal attributes *d1mn*, *d1mx*, *d0mn* and *d0mx* of each signal in the circuit.

The timing simulation mechanism is inserted into gate models with additional processes that monitors and updates the values of signal attributes. For a model of two-input NAND gate this process is represented by the pseudo code shown in Fig. 1.

```
process (a, b) {
  // update falling edge delay of signal f:
  if (arr0(a) .OR. arr0(b))
    d0(f) := MAX(d0(a),d0(b)) + tf;
    arr0(f) := 'true';
    f <= an_event;   }
  // update rising edge delay of signal f:
  if (arr1(a) .AND. arr1(b)) {
    d1(f) := MAX(d1(a),d1(b)) + tr;
    arr1(f) := 'true';
    f <= an_event;   }
}
```

Fig. 1: *Two-input AND gate model for longest path delay estimation with logic simulator*

Gate inputs are denoted *a* and *b*, gate output is denoted *f*, whereas gate propagation delays for rising and falling edge at the output *f* are denoted *tr* and *tf* respectively. Each falling transition at an input of the gate results in rising transition at

its output, but rising transition at an input is capable to produce falling transition at the output only if rising transition had previously arrived at the other gate input. Delay model of arbitrary complexity can be applied, taking into account input signal slopes, loading capacitances and other parameters that influence ranges of gate delays *tr* and *tf*. Each time a delay attribute of output signal *f* is changed, an event must be scheduled to signal *f* with zero delay in order to activate the processes performing the timing simulation in the gates driven by the NAND gate.

To implement the timing simulation algorithm in logic simulator, a generalized signal attribute modeling mechanism is proposed in [2]. It is built in AleC++ hardware description language – the input language of Alecsis simulator [7]. In AleC++ user-defined signal attributes can be accessed and updated in the processes during the simulation. The experimental results presented in [2] show that the proposed method is fast and appropriate for interactive use during logic verification.

3. IMPLEMENTATION IN VHDL

Although the user-defined signal attributes are available in VHDL [87], they cannot be accessed and updated from the processes during the simulation. Therefore, a different modeling mechanism is necessary for the implementation of the timing simulation in VHDL simulator. We propose the use of ***composite signals of the record type***. Such a composite signal consists of logic state and the timing attributes *arr1mn*, *arr1mx, arr0mn*, *arr0mx*, *d1mn, d1mx, d0mn* and *d0mx*. The record type DCSM_std_logic for the scalar signals can be declared in VHDL as:

```
type DCSM_std_logic is record
        statemn: std_ulogic;
        d0mn: time;
        d1mn: time;
        arr0mn: boolean;
        arr1mn: boolean;
        statemx: std_ulogic;
        d0mx: time;
        d1mx: time;
        arr0mx: boolean;
        arr1mx: boolean;
end record DCSM_std_ulogic;
```

For the bus signals type DCSM_std_logic_vector can be declared as:

```
type DCSM_std_logic_vector is array
  (natural range <>) of DCSM_std_logic;
```

Assuming the input and output signals of the DCSM_std_logic type, the complete model of a two-input NAND gate can be described in VHDL as shown in Fig. 2. Processes labeled *p1* and *p2* are intended for standard logic simulation. They model the logic function and delay function of the NAND gate. Processes labeled *p3* and *p4* are intended for the timing simulation. They monitor the timing attributes of gate inputs *inp1* and *inp2* and update the timing attributes of gate output *oupt*.

Note that processes *p3* and *p4* are sensitive to the timing attributes of the input signals *inp1* and *inp2*, but not to their logic states. Similarly, process *p1* and *p2* are sensitive to events on input signal states, but not to their timing attributes. Hence, the two types of processes are independent. Logic simulation and timing simulation can be performed separately or simultaneously.

It is also important that the changes of the timing attributes of the output signal *outp* are always scheduled with zero delay in processes *p3* and *p4*. If the timing simulation is initiated at some time instant of simulation time, it will be finished at the same time instant, since the activity related to timing simulation is performed with no delay. We suggest that timing simulation should be performed in initial time instant $t=0$ of simulation. If no timing attribute is changed during the simulation, timing simulation is suppressed and the simulator performs standard logic simulation. Timing simulation is invoked by setting the timing attributes *arr1mn, arr1mx, arr0mn* and *arr0mx* of primary inputs to value true. If the primary inputs are grouped into a bus signal *inp*, the timing analysis can be invoked by only one concurrent signal assignment command:

```
inp <= (others => ('0', 0 sec, 0 sec, true, true, '0', 0 sec, 0 sec, true, true));.
```

Once the simulation time advances to $t>0$ sec, longest and shortest path delays of all signals in the circuit are available and can be logged out. One additional process in the top-level entity can be added that waits for some small time period (for example, 1 pico second), prints the shortest and the longest path delays to the screen or file, and then suspends to the end of the simulation. An example of such a process is as follows:

```
log: process
begin
  wait for 1 ps;
  --- print the timing analysis results
  wait;
end process log;
```

```
entity nandg is
    generic (
          tpd_hlmn : time := 0.95 ns;
          tpd_lhmn : time := 1 ns;
          tpd_hlmx : time := 0.9 ns;
          tpd_lhmx : time := 1.05 ns);
          port (out1 :    out DCSM_std_logic := ('0', 0.0 sec, 0.0 sec, false,
false, '0', 0.0 sec, 0.0 sec, false, false);
                in1, in2:  in DCSM_std_logic := ('0', 0.0 sec, 0.0 sec, false,
false, '0', 0.0 sec, 0.0 sec, false, false)
    );
end nandg;
architecture only of nandg is
begin

---- Logic function:
    p1: process(in1.statemn, in2.statemn)
    variable val,ex_value : std_logic := '0';
    begin
        val := in1.statemn and in2.statemn;
        val := not (val);
        if val /= ex_value then
          ex_value := val;
          case val is
            when '0' =>
                out1.statemn <= val after tpd_hlmn;
            when '1' =>
                out1.statemn <= val after tpd_lhmn;
            when others =>
          end case;
        end if;
    end process p1;

    p2: process(in1.statemx, in2.statemx)
    variable val,ex_value : std_logic := '0';
    begin
        val := in1.statemx and in2.statemx;
        val := not (val);
        if val /= ex_value then
          ex_value := val;
          case val is
            when '0' =>
                out1.statemx <= val after tpd_hlmx;
            when '1' =>
                out1.statemx <= val after tpd_lhmx;
            when others =>
          end case;
```

```
      end if;
   end process p2;

---- Timing simulation:
   p3: process (in1.d0mn, in1.d1mn, in1.arr0mn, in1.arr1mn,
                in2.d0mn, in2.d1mn, in2.arr0mn, in2.arr1mn)
       begin
         if (in1.arr0mn or in2.arr0mn) then
         out1.d1mn  <= min(in1.d0mn, in2.d0mn) + tpd_lhmn;
         out1.arr1mn <= true;
         end if;
         if (in1.arr1mn and in2.arr1mn) then
         out1.d0mn  <= min(in1.d1mn, in2.d1mn) + tpd_hlmn;
         out1.arr0mn <= true;
         end if;
  end process p3;

   p4: process (in1.d0mx, in1.d1mx, in1.arr0mx, in1.arr1mx,
                in2.d0mx, in2.d1mx, in2.arr0mx, in2.arr1mx)
       begin
         if (in1.arr0mx or in2.arr0mx) then
         out1.d1mx  <= max(in1.d0mx, in2.d0mx) + tpd_lhmx;
         out1.arr1mx <= true;
         end if;
         if (in1.arr1mx and in2.arr1mx) then
         out1.d0mx <= max(in1.d1mx,in2.d1mx) + tpd_hlmx;
         out1.arr0mx <= true;
         end if;
       end process p4;
end only;
```

Fig. 2: *VHDL implementation of two-input AND gate model*

The sequential elements are modeled in a similar manner. For timing simulation, data inputs of sequential elements are the ending points of the signal propagation paths, whereas the outputs of the sequential elements are the starting points of the signal propagation paths. That means the sequential element must assign the value true to the timing attributes *arr1mn, arr1mx, arr0mn* and *arr0mx* of output signals at time t=0 and log out the timing attributes *d1mn, d1mx, d0mn* and *d0mx* of data inputs at some time t>0.

For example, the model of a D-type flip-flop can be described with the VHDL code shown in Fig. 3. The delay parameters of the flip-flop are directly added to the appropriate path delays. The propagation delays tr_ck_q and tf_ck_q are initially assigned to timing attributes *d1* and *d0* of the output *q*, for both shortest and longest path. The set-up times tsu_d_ckmn and tsu_d_ckmx are added to the delay of the path ending at data input *d*. The log file logfile is declared in a package to avoid the overleaping of writes from different instances of flip-flop dff.

```
entity dff is
generic(tr_ck_qmn:  time := 0.85 ns;
        tf_ck_qmn:  time := 0.9 ns;
        tsu_d_ckmn: time := 1.05 ns;
        tr_ck_qmx:  time := 0.9 ns;
        tf_ck_qmx:  time := 0.95 ns;
        tsu_d_ckmx: time := 1 ns);
 port(d,ck : DCSM_std_logic
         := ('0',0sec,0sec,false,false, '0',0sec,0sec,false,false);
      q: out TS_std_logic
       := ('0',0sec,0sec,false,false, '0',0sec,0sec,false,false));
end dff;

architecture dff_arch of dff is
begin
 --- Logic simulation:
p1: process(ck.statemn)
 variable ex_value,val: std_logic := '0';
 begin
  if rising_edge(ck.statemn) then
    val := d.statemn;
    if (ex_value /= val) then
      ex_value := val;
      case val is
        when '0' =>
          q.statemn <= '0' after tf_ck_qmn;
        when '1' =>
          q.statemn <= '1' after tr_ck_qmn;
        when others =>
          q.statemn <= val;
      end case;
    end if;
  end if;
 end process p1;

 p2: process(ck.statemx)
  variable ex_value,val: std_logic := '0';
 begin
   if rising_edge(ck.statemx) then
     val := d.statemx;
     if (ex_value /= val) then
       ex_value := val;
       case val is
         when '0' =>
           q.statemx <= '0' after tf_ck_qmx;
         when '1' =>
           q.statemx <= '1' after tr_ck_qmx;
         when others =>
           q.statemx <= val;
       end case;
     end if;
   end if;
 end process p2;

 ---- Timing simulation:
 q.arr1mn <= true;
 q.arr0mn <= true;
 q.d0mn <= tf_ck_qmn;
 q.d1mn <= tr_ck_qmn;
 q.arr1mx <= true;
 q.arr0mx <= true;
 q.d0mx <= tf_ck_qmx;
 q.d1mx <= tr_ck_qmx;
end only;

log_timing: process
 use std.textio.all;
 file log: text open write_mode is "DFF.delays";
 variable line_1: line;
 begin
         wait for 1 ps;
         write (line_1, outp.d0mn+tsu_d_ckmn, left, 10);
         write (line_1, outp.d0mx+tsu_d_ckmx, left, 10);
         write (line_1, outp.d1mn+tsu_d_ckmn, left, 10);
         write (line_1, outp.d1mx+tsu_d_ckmx, left, 10);
         writeline (log, line_1);
         wait;
 end process log_timing;
end architecture dff_arch;
```

Fig. 3: *VHDL implementation of D-type flip-flop model*

4. EXPERIMENTAL RESULTS

The efficiency of the proposed method is tested on ISCAS'85 benchmark circuits [9]. They are simulated with VHDL simulator ActiveHDL [10]. Timing simulation is performed at t=0 seconds, the results are logged at t=1 picosecond and simulation is finished at t=2 picoseconds. To be able to easily validate the results, for this experiment all the gates in the ISCAS'85 circuits were first chosen to have delays *trmn=trmx=tfmn=tfmx*=1ns. In such a case, the longest path delays of rising and falling edge of each signal are the same and match the topological level of that particular signal expressed in nanoseconds, as shown in column D, in Table 1. Since the gate delays are 1 nanosecond and the simulation is stopped after only 2 picoseconds, only timing simulation and initialisation phase of the logic simulation is done. The results of the shortets and the longest path delays analysis are also shown in Table 1, for the simulation of the same circuits which gates have following delays: *tfmn*=0.95ns, *trmn*=1ns, *tfmx*=0.9ns, *trmx*=1.05ns. For each

ISCAS'85 circuit, the simulation result was a list of *d1* and *d0,* minimal and maximal parameters of all primary output signals. The maximal of these values is given in column $D_{mx,}$ while the minimal values are given in column D_{mn} of Table 1 representing the delay of the longest and the shortest paths in the circuit.

Table 1: Results of timing and logic simulation of ISCAS'85 benchmark circuits with a VHDL simulator

Circuit name	Number of signals	*Number of inputs*	*Number of outputs*	*Number of gates*	*D* [ns]	D_{fmn} [ns]	D_{fmx} [ns]	D_{rmn} [ns]	D_{rmx} [ns]
c17	11	5	2	6	3	1.95	2.85	1.95	3
c432	196	36	7	160	17	1.95	16.5	1.95	16.5
c499	243	41	32	202	11	0.95	1125	1	11.4
c880	443	60	26	383	24	5.8	23.4	5.9	23.7
c1355	587	41	32	546	24	2.9	23.55	2.95	23.25
c2670	1426	233	140	1193	32	4.9	31.65	4.85	31.2
c3540	1719	50	22	1669	47	13.7	46.2	13.5	46.95
c5315	2485	178	123	2307	49	3.95	48.3	3.85	47.25
c6288	2448	32	32	2416	124	4.85	120.9	4.9	120.9
c7552	3719	207	108	3512	43	5.9	41.85	5.8	42

To estimate the additional CPU time consumed by the timing simulation during the simulator run, the ISCAS'85 circuits are also simulated using standard logic package ieee.std_logic_1164. Signals of type std_logic and std_logic_vector were used in this experiment. Since timing simulation in the first experiment is finished at t=2 picoseconds, the logic simulation in the second experiment is also finished at t=2 picoseconds. Timing simulation does not increase significantly the total CPU time.

5. CONCLUSION

During the design process of digital integrated circuits it is extremely important to estimate the longest and the shortest path delays as early as possible. Design verification is one of the early phases in the design process. It is usually performed using a logic simulator to verify that the circuit meets the required logic function and a timing simulator to verify that the circuit meets the required timing specifications. The method proposed in this paper combines these two tools into one by enabling the VHDL simulator to perform the timing simulation. In this way the design process is simplified and accelerated.

The problem of introducing of the timing simulation into the VHDL simulator is solved by the use of signals of record type in VHDL. Additional timing data is associated with each signal and used only when the timing simulation is requested. Eight timing attributes are added to each signal.

Extra memory for storing the timing attributes is proportional to the total number of signals in the circuit. For simulated ISCAS'85 circuits this number is less than 4000. Even if very complex circuits with millions of signals should be simulated, the required extra memory would be of the order of only tens of Mbytes. Storing memory is not a critical issue and the simulator performance should not be affected by the memory requirements.

The experimental results also indicate that extra simulation CPU time due to timing simulation is almost negligible for ISCAS'85 circuits. Timing simulation affects only the initialization phase of the simulation, whereas the rest of the simulation flow usually takes the most of the CPU time. The proposed method is simple for implementation and libraries of gate models used in semi-custom design methodologies.

REFERENCES

[1] D. M. Maksimović, V. B. Litovski, "Logic Simulation Methods For Longest Path Delay Estimation", IEE Proc.-Comput. Digit. Tech., Vol. 149, No. 2, March 2002.

[2] D. M. Maksimović, V. B. Litovski, "Tuning logic simulators for timing analysis", Electronics Letters, Vol. 35, No. 10, pp. 800-802, Stevenage, UK, May 1999.

[3] J. K. Ousterhout, "CRYSTAL: A Timing Analyzer for NMOS VLSI Circuits", Proc. 3rd Caltech Conf. on VLSI, pp. 57-69, March 1983.

[4] N. Jouppi, "TV: An NMOS Timing Analyzer", Proc. 3rd Caltech Conf. on VLSI, pp. 71-85, March 1983.

[5] C. Oh, M.R. Mercer, "Efficient Logic-Level Timing Analysis Using Constraint-Guided Critical Path Search", IEEE Trans. on VLSI Systems, Vol. 4, No. 3, September 1996, pp. 346-355.

[6] B. Hoppe, G. Neuendorf, D. Schmitt-Landsiedel, W. Specks, "Optimization of High-Speed CMOS Logic Circuits with Analytical Models for Signal Delay, Chip Area, and Dynamic Power Dissipation", *IEEE Trans. CAD*, Vol. 9, No. 3, pp. 236-247, March 1990.

[7] Ž. Mrčarica, D. Glozić, V. Litovski, D. Maksimović, T. Ilić, D. Gavrilović, *Alecsis 2.3 - the simulator for circuits and systems*, User's manual, Laboratory for Electronic Design Automation (LEDA), University of Niš, Faculty of Electronic Engineering, 1/1998, Niš, Yugoslavia (http://leda.elfak.ni.ac.yu)

[8] IEEE-Std. 1076, 1993 Language Reference Manual

[9] E. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits And a Target Translator in FORTRAN", *Int. Symp. on Circuits and Systems*, pp. 1929-1934, Kyoto, Japan, June 1985.

[10] Active-HDL, ver. 5.1, ALDEC Inc., 2003.

Sadržaj – U ovom radu predložen je metod analize kašnjenja korišćenjem VHDL simulatora. Ovaj metod je nadgradnja onih objavljenih u [1, 2], i omogućava sumulatoru da odredi minimalna i maksimalna kašnjenja svih puteva u kolu, i to izvršavanjem samo jedne simulacije u kolu. Vremenska analiza se izvršava u trenutku t=0, i samo neznatno produžava procesorsko vreme potrebno za simulaciju. Da bi se verifikovala efikasnost predloženog metoda, prikazani su i rezultati simulacije ISCAS'85 benchmark kola, korišćenjem VHDL simulatora. Ovaj metod izuzetno je pogodan u ranim fazama procesa projektovanja gde je potrebno ponavljati vremensku analizu uvek kada se kolo optimizuje ili redizajnira.

PROCENA KAŠNJENJA LOGIČKOG PUTA KORIŠĆENJEM VHDL LOGIČKE SIMULACIJE

Miljana Sokolović and Dejan Maksimović