

# **Self Adjusting Unidirectional Switch Models for Dynamic Load Calculation and Fast Switch Level Simulation**

**Zainalabedin Navabi and Zahra Razavi  
Electrical and Computer Engineering Department  
Faculty of Engineering; University of Tehran  
North Kargar Avenue; 14399 Tehran, Iran  
Fax: +(98-21) 688690**

## **ABSTRACT**

This paper discusses VHDL switch models that adjust their logic flow direction at the beginning of a simulation run. The timing of such a model depends on the capacitive load at its drain or source terminals, and this timing can change dynamically depending on the visibility of load connected to an output through transmission gates. The usage of these models is primarily in creating an interface between VLSI design tools and high level VHDL design environments.

## **1. Introduction**

A technique for modeling transistors at the switch level is presented here. The transistors are modeled as unidirectional on-off switches that adjust their logic flow direction at the start of a simulation run. This is achieved by sources identifying themselves at time zero. Unidirectionality in calculation of logic values provides a very fast switch level simulation. In addition to logic flow, capacitive load, which is used for delay calculations, is also handled by our proposed models. Node capacitance values flow between drain and source of transistors in a direction opposite to that of logic values. Load calculation in these models is dynamic. A transmission gate that has been turned on, conducts logic values from its input (Source) to its output (Drain), and conducts capacitance values from its output to its input. When this gate is off, it removes logic values from its output and removes capacitance values from its input. This effect causes load calculation to be dynamic and dependent on the on-off state of transistors in the circuit.

For these models to be useful, several utility and component packages for type declarations, input-output subprograms, and component declarations have been developed. A set of programs have been developed for converting CIF files to VHDL transistor level descriptions. Another program for extracting appropriate VHDL parameters from SPICE models has been developed.

These programs and the VHDL models provide a convenient interface between layout generation tools and VHDL design and simulation environments. For the VLSI designers our programs and models provide a fast VHDL based switch level simulation tool. For higher simulation speed, the models can be modified to use fixed delay values and perform only logic value calculations.

In the next section we will discuss a VLSI design environment and the usage of our models in this design process. Section 3 presents some of the previous work on using VHDL for switch level simulation. In Section 4 we will discuss the mechanism and the theory of operation of our self-adjusting models. The section that follows this discusses the implementation of the models and its supporting packages. Section 6 discusses various software programs that have been developed for the utilization of self-adjusting transistor models. The last section is Section 7 which presents conclusions and future direction of this work.

## **2. A VLSI Design Environment**

VHDL tools provide a very convenient environment for design and simulation of digital systems. In order to be able to take advantage of VHDL tools in a VLSI design environment, interfaces

at the transistor and geometrical level must be made for VHDL. Such interfaces will enable a hardware designer to integrate his layout level designs with higher level gate, dataflow and behavioral descriptions, and be able to test and verify an entire heterogeneous circuit.

In addition to providing a bridge between VLSI design and high level VHDL based design, self-adjusting models are useful for a stand alone switch level simulation. In this simulation the VLSI designer has control on the details of simulation by adjusting the transistor models.

### 3. Other Switch Level Models

For being able to use VLSI tools and VHDL tools in an integrated design environment, and in order to utilize the power of VHDL in switch level simulation of VLSI circuits, several methods have been proposed and implemented. One method is to model transistors as bidirectional switches and use these models on transistor level descriptions extracted from layout description files. One such implementation does not consider timing [1] and another [2], calculates load capacitance only at the beginning of simulation without the capability to modify capacitances when a transistor state changes. A major problem with both these implementations is their slow simulation speed. Because of the complexity of the models and the many processes and transactions that are involved in calculation of transistor node values, using these models for simulation of VLSI circuits with thousands of transistors is impractically slow.

An alternative scheme that can be used for integration of layout information with higher level descriptions is to extract logic level information from layout files and perform detailed timing simulation [3] using parameters extracted from transistor SPICE files. A problem with this method is that certain switch level hardware constructs cannot be mapped into gate level constructs. This is usually because of the use of switches in dynamic logic and memory structures. Otherwise, the method is reliable and achieves high simulation speed and accuracy.

### 4. Self-Adjusting Switch Models

Instead of the methods discussed above, we are proposing the use of unidirectional transistor models that adjust their own logic flow direction at the beginning of a simulation run. Once the direction of a transistor is known, it can calculate its logic output value by the use of simple processes. Also since the output of the gate is known, it need not respond to events occurring on its output, and therefore simulation time will not be spent for forward and backward logic propagation. For accurate timing calculation, a switch model passes capacitance seen at its drain to its source only when the switch in on. The capacitance passed as such, will be added to all other capacitance values that have a conducting path to the node. The models are capable of dynamically adjusting their timing during the simulation run.

	'T', 'G', 'U', '0', '1', 'Z', 'X',
'T' :	('T', 'G', 'U', '0', '1', 'Z', 'X')
'G' :	('G', 'G', 'G', '0', '1', 'Z', 'X')
'U' :	('U', 'G', 'U', 'U', 'U', 'U', 'X')
'0' :	('0', '0', 'U', '0', 'X', '0', 'X')
'1' :	('1', '1', 'U', 'X', '1', '1', 'X')
'Z' :	('Z', 'Z', 'U', '0', '1', 'Z', 'X')
'X' :	('X', 'X', 'X', 'X', 'X', 'X', 'X')

Figure 1. Logic value system and resulting wired values

There are three key points in the implementation of self-adjusting transistor models. These are seven value logic, source triggering, and load flow direction. The first point is a seven value logic system that distinguishes between nodes at which logic flow direction is undecided, those whose direction are known, and the gates of the transistors. Transitional logic values, that are given to transistor nodes, will turn into actual values after all source inputs have identified themselves. Figure 1 shows this logic value system. The table also shows values that result when a node is driven by two or more signals of this seven-value logic type. Logic value 'T' is transitional value for all Source and Drain transistor terminal. This value indicates that the direction of logic flow to the terminal is yet undecided. Value 'G'

is the initial value of all transistor Gate terminal. Value 'U' is "unknown" and is used for those nodes that are connected to transistors whose direction is determined, but their terminal values are unknown due to open gate or unknown input.

The second key point in the implementation of the models is that all source inputs identify themselves at the beginning of a simulation run. This identification causes actual logic values ('U', '0', '1') to propagate through all transistors, and thereby, replace the transitional values ('T'). Source inputs consists of supply, ground, and all primary inputs.

```

USE WORK.t_utilities.ALL;
ENTITY ground IS PORT (gnd : OUT t_bit := ('T', 0 ffr)); END ground;
--
ARCHITECTURE triggering OF ground IS
BEGIN
  gnd <= '0';
END triggering;

```

Figure 2. Triggering ground terminal

Figure 2 shows a *ground* component that connects to the ground node of the transistors. At the start of a simulation run a '0' on the Source (or Drain) of a transistor connected to the ground causes the transistor to set its logic flow direction away from the Source (or Drain) terminal and towards the other non Gate terminal, i.e., Drain (or Source). A transistor whose direction is determined in this way, informs other transistors of its direction by placing a 'U' on its Drain. This 'U' value causes the next transistor to set its direction in the same fashion as the first one. This effect dominos until a transistor whose direction is determined sees a Gate (identified by value 'G') connected to its Drain or Source terminal. At this point, the domino effect that started from a ground, supply, or primary input ceases, while other domino effects, due to other sources continue until they too reach a Gate input. This entire process takes place at time 0, and depending on the number of series transistors, after a few delta times all transistors determine their logic flow direction.

```

ARCHITECTURE self_adjusting OF nmos IS
...
  IF direction = und THEN
    IF source'EVENT AND source /= 'G' THEN
      direction := s2d;
      drain <= 'U';
    ELSIF drain'EVENT AND drain /= 'G' THEN
      direction := d2s;
      source <= 'U';
    END IF;
  ELSIF direction = s2d THEN
    ...
  END self_adjusting;

```

Figure 3. Finding direction in a switch model

Figure 3 shows portion of a process statement in the self-adjusting architecture of an NMOS transistor that is responsible for setting the direction of the transistor. This part of code executes only once at the start of simulation. When direction is undecided (*und*), an event of a terminal, e.g, Source terminal, causes the direction to be set away from that terminal and towards the other transistor terminal. An event on the Source terminal causes the direction to be set at Source-to-Drain (*s2d*). The placement of 'U' on the output terminal (Drain), causes transistors connected in series to this transistor to execute their part of direction determination code.

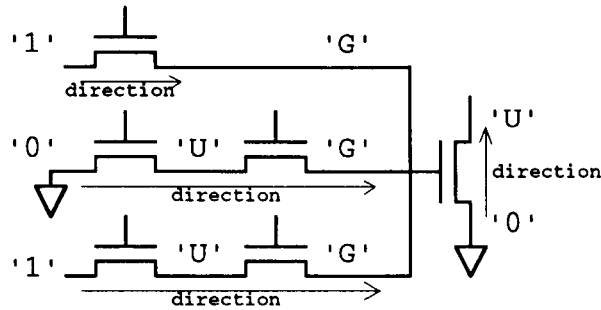


Figure 4. Propagating values and setting switch directions

In order for a transistor terminal to be capable of being input or output, INOUT mode is used for all port signals. Since each port carries both logical and load values, a record type is used for port declarations. The fields of a port are *logic* and *cap*[4]. The *logic* field contains a wiring resolution of all values placed on a node, while the *cap* field contains the sum of all capacitances visible at a node. For each transistor model to be aware of connections made to its terminals, *logic* fields of all signals are declared as resolved signals using the resolution shown in Figure 1. A 'G' value connected to a terminal results in 'G' regardless of other values driving the terminal. This way, direction determination of several series transistor paths connected to a gate input all stop when they reach their common node. Figure 4 shows direction and node logic values of series transistors connected to a gate input.

```

...
ELSIF direction = s2d THEN
  IF effective_gate'EVENT OR source.logic'EVENT THEN
    CASE effective_gate IS
      WHEN '0' =>
        drain.logic <= 'Z'; source.cap <= 0 ffr;
        WAIT ON drain.logic'TRANSACTION;
      WHEN '1' =>
        drain.logic <= source.logic AFTER (!*unit_n_res/w)*drain.cap;
        source.cap <= drain.cap AFTER (!*unit_n_res/w)*drain.cap;
        WAIT ON drain.logic'TRANSACTION;
      WHEN OTHERS =>
        drain.logic <= effective_gate;
        source.cap <= 0 ffr;
        WAIT ON drain.logic'TRANSACTION;
    END CASE;
  END IF;
...

```

Figure 5. Flow of logic and load values

Another key point in the implementation of self adjusting models is the opposite flow of logic and load in each transistor. Figure 5 shows portion of an NMOS transistor process that executes when the direction is determined to be from the Source to the Drain terminal. In this case, when an event occurs on one of the inputs (Gate or Source) when the Gate is on, the logic value on the Source terminal is assigned to the Drain terminal (direction is from Source to Drain) using a delay that depends on the size of the transistor (*length* and *width*), and is proportional to the channel unit resistance and the capacitance seen at the Drain terminal. At the same time, the capacitance seen at the Drain terminal is assigned to the Source *cap* field (direction is from Drain to Source). All capacitances assigned to a node are added by a resolution function and the sum is used in calculating delay of an output terminal that drives such a capacitive node.

```

PACKAGE t_utilities IS
TYPE t_value IS ('T','G','U','0','1','Z','X');
TYPE capacitance IS RANGE 0 TO 1E16
UNITS
  ffr; -- Femto Farads (base unit)
  pfr = 1000 ffr;  nfr = 1000 pfr;
  ufr = 1000 nfr;  mfr = 1000 ufr;
  far = 1000 mfr;  kfr = 1000 far;
END UNITS;
TYPE t_values IS ARRAY (NATURAL RANGE <>) OF t_value;
TYPE capacitances IS ARRAY (NATURAL RANGE <>) OF capacitance;
FUNCTION t_resolve (drivers : t_values) RETURN t_value;
FUNCTION t_load (contributors : capacitances) RETURN capacitance;
SUBTYPE t_logic IS t_resolve t_value;
SUBTYPE t_cap IS t_load capacitance;
TYPE t_bit IS RECORD
  logic : t_logic;  cap : t_cap;
END RECORD;
TYPE t_bit_vector IS ARRAY (NATURAL RANGE <>) OF t_bit;
SUBTYPE lambda IS INTEGER RANGE 1 TO 100;
TYPE resistance IS RANGE 0 TO 1E16
UNITS
  l_o; -- Milli-Ohms (base unit)
  ohms = 1000 l_o;
  k_o = 1000 ohms;
  m_o = 1000 k_o;
  g_o = 1000 m_o;
END UNITS;
FUNCTION "*** (a : resistance; b : capacitance) RETURN TIME;
PROCEDURE putbit (SIGNAL s: OUT t_bit; bit_file: IN STRING);
END t_utilities;
--
USE WORK.t_utilities.ALL;
PACKAGE t_parameters IS
  CONSTANT unit_n_res : resistance := 6 k_o;
  CONSTANT unit_p_res : resistance := 9 k_o;
  CONSTANT unit_gate_cap : capacitance := 3 ffr;
  CONSTANT open_gate_res : resistance := 800 m_o;
END t_parameters;

```

Figure 6. Utilities and parameter packages

### 5. Implementation of Self-Adjusting Models

The implementation of self-adjusting switch models uses three packages. The first package is *t\_utilities* the declaration of which is shown in Figure 6. This package includes various utility type declarations, resolution functions (*t\_resolve* and *t\_load*), type declaration for port signals (*t\_bit*), RC calculation function ("*\*\**"), and a procedure for inputting logic values from external files (*putbit*).

```

USE WORK.t_utilities.ALL;
USE WORK.t_parameters.ALL;
PACKAGE t_components IS
  COMPONENT ground  PORT (gnd : OUT t_bit := ('T', 0 ffr));
  END COMPONENT;
  COMPONENT supply  PORT (vdd : OUT t_bit := ('T', 0 ffr));

```

```

END COMPONENT;
COMPONENT cap
  GENERIC (cap_val : capacitance := 0 ffr);
  PORT (line : OUT t_bit := ('T', 0 ffr));
END COMPONENT;
COMPONENT nmos
  GENERIC (l : lambda := 2; w : lambda := 6);
  PORT (source : INOUT t_bit := ('T', 0 ffr);
        gate : INOUT t_bit := ('G', l*w*unit_gate_cap);
        drain : INOUT t_bit := ('T', 0 ffr) );
END COMPONENT;
COMPONENT pmos
  GENERIC (l : lambda := 2; w : lambda := 6);
  PORT (source : INOUT t_bit := ('T', 0 ffr);
        gate : INOUT t_bit := ('G', l*w*unit_gate_cap);
        drain : INOUT t_bit := ('T', 0 ffr) );
END COMPONENT;
END t_components;

```

Figure 7. Package containing component declarations

Another package (*t\_parameters*) that is also shown in Figure 6, contains parameters that are used in transistor RC delay calculations. The resistance values shown in this figure are for a square transistor channel, and the capacitance value is for the Gate of a minimum (2 Lambda by 2 Lambda) size transistor. These parameters are extracted from transistor SPICE models.

A third package used for the implementation of our models contains declarations for *ground*, *supply*, *cap*, *nmos*, and *pmos*. The *ground* and *supply* components trigger the transistor terminals that are connected to them. The *cap* component places a capacitance on the *cap* filed of a transistor terminal. This component models line capacitances. The *nmos* and *pmos* components model the transistors that are used in a CMOS circuit. The size of a transistor is passed to its appropriate model by use of the generic parameters of the *nmos* and *pmos* components.

```

ARCHITECTURE self_adjusting OF nmos IS
  SIGNAL effective_gate : t_logic;
BEGIN
  effective_gate <= '0'
    AFTER open_gate_res * gate.cap WHEN gate.logic = 'Z' ELSE gate.logic;
PROCESS
  TYPE logic_flow IS (und, d2s, s2d);  VARIABLE direction : logic_flow;
BEGIN
  IF direction = und THEN
    IF source.logic'EVENT AND (source.logic /= 'G') THEN
      direction := s2d;  drain <= ('U',0 ffr);
    ELSIF drain.logic'EVENT AND (drain.logic /= 'G') THEN
      direction := d2s;  source <= ('U',0 ffr);
    END IF;
  ELSIF direction = s2d THEN
    IF effective_gate'EVENT OR source.logic'EVENT THEN
      CASE effective_gate IS
        WHEN '0' => drain.logic <= 'Z';  source.cap <= 0 ffr;
          WAIT ON drain.logic'TRANSACTION;
        WHEN '1' => drain.logic <= source.logic AFTER (l*unit_n_res/w)*drain.cap;
          source.cap <= drain.cap AFTER (l*unit_n_res/w)*drain.cap;
          WAIT ON drain.logic'TRANSACTION;
      END CASE;
    END IF;
  END PROCESS;

```

```

    WHEN OTHERS => drain.logic <= effective_gate;
        source.cap <= 0 ffr;
        WAIT ON drain.logic'TRANSACTION;
    END CASE;
END IF;
ELSE -- direction = d2s
IF effective_gate'EVENT OR drain.logic'EVENT THEN
CASE effective_gate IS
    WHEN '0' => source.logic <= 'Z'; drain.cap <= 0 ffr;
        WAIT ON source.logic'TRANSACTION;
    WHEN '1' => source.logic <= drain.logic AFTER (!*unit_n_res/w)*source.cap;
        drain.cap <= source.cap AFTER (!*unit_n_res/w)*source.cap;
        WAIT ON source.logic'TRANSACTION;
    WHEN OTHERS => source.logic <= effective_gate;
        drain.cap <= 0 ffr;
        WAIT ON source.logic'TRANSACTION;
    END CASE;
END IF;
END IF;
WAIT ON effective_gate, source.logic, drain.logic;
END PROCESS;
END self_adjusting;

```

Figure 8. NMOS transistor model

Figure 8 shows a complete model for an NMOS transistor. The first process sets the effective value of the Gate terminal to 'Z' after a delay that depends on the gate resistance and capacitance. The second process in this model sets the switch direction and assigns logic and load values to the output and input of the switch, respectively.

### 6. Extracting VHDL From Layout

Most VLSI layout editors provide the standard CIF (Caltech Intermediate Format) format. This file contains all geometrical information, and with the availability of fabrication technology data, detailed information such as line capacitances can be extracted from it. Many layout editors provide an extraction program that extracts an SPICE file from layout information. SPICE files with technology dependent transistor models can be used for detailed device level simulation. For utilizing our self-adjusting switch models, we have developed two software packages. The first package translates a CIF file into a VHDL file, extracting transistors and commutative line capacitances. A CIF file is first translated into an SPICE file with equivalent line capacitances, and then into an appropriate VHDL file. The second package uses SPICE transistor models for extracting necessary parameters for the VHDL self-adjusting switch models. This translation process is shown in Figure 9.

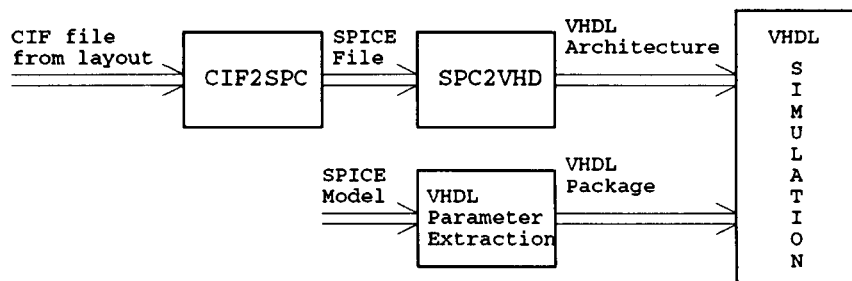


Figure 9. Using CIF files in a VHDL simulation environment

```

M1 45 37 41 45 PMOS L=2U W=6U
M2 45 44 37 45 PMOS L=2U W=6U
M3 41 42 44 45 PMOS L=2U W=6U
M4 45 42 43 45 PMOS L=2U W=6U
M5 31 43 44 45 PMOS L=2U W=6U
C45 45 0 10.208FF
* Node 45 = reg\vdd:
C44 44 0 13.986FF
C43 43 0 6.696FF
C42 42 0 4.64FF
* Node 42 = reg\c1_i:
C41 41 0 8.207FF
* Node 41 = reg\o1_o:
C40 40 0 10.208FF
* Node 40 = reg\gnd:
C37 37 0 3.596FF
C31 31 0 5.104FF
* Node 31 = reg\i1_i:
C19 19 0 0.1FF
*.MODEL NMOS
*.MODEL PMOS
M6 41 37 40 19 NMOS L=2U W=6U
M7 37 44 40 19 NMOS L=2U W=6U
M8 41 43 44 19 NMOS L=2U W=6U
M9 40 42 43 19 NMOS L=2U W=6U
M10 31 42 44 19 NMOS L=2U W=6U
* Total Nodes: 9 ; Total Elements: 10 ;
.tran 1.000n 35.000u 0 0 ;*ipsp*
.END

```

Figure 10. SPICE file for a CMOS latch with refreshing path

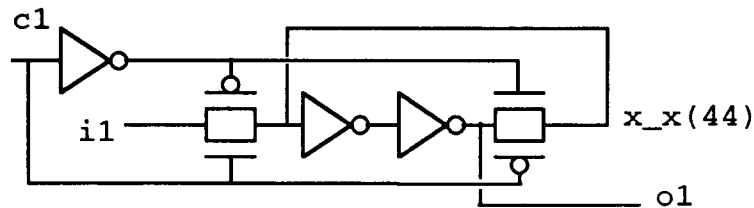


Figure 11. Latch used for simulation example

The model extraction program runs SPICE simulation on an inverter model in order to extract pull-up and pull-down resistances as well as the gate capacitance. For this purpose a large capacitance is placed at the output of the inverter and pull resistances are calculated by reading rise and fall times from the SPICE output file. For the gate capacitance, output of an inverter is connected to the input of another through a large resistance. The delay time read from the SPICE output file yields the value for the gate capacitance ( $delay\_time = 2.2 R * C$ ). For a typical 2 Micron process values of 4.54 K-Ohm and 7.27 K-Ohm were extracted for the pull-down and pull-down resistances of a 2 Lambda by 2 Lambda transistor, and 2 Femto-Farad was extracted for the Gate capacitance of the same size transistor.

```

USE WORK.t_utilities.ALL; USE WORK.t_components.ALL;
--
ENTITY d_latch IS PORT (c1, i1 : INOUT t_bit; o1 : INOUT t_bit);
END d_latch;
ARCHITECTURE tgate_based OF d_latch IS
  ALIAS x42 : t_bit IS c1;
  ALIAS x41 : t_bit IS o1;
  ALIAS x31 : t_bit IS i1;
  SIGNAL x_x : t_bit_vector(31 TO 45);
BEGIN
  m1: pmos GENERIC MAP (2,6) PORT MAP (x_x(45), x_x(37), x41);
  m2: pmos GENERIC MAP (2,6) PORT MAP (x_x(45), x_x(44), x_x(37));
  m3: pmos GENERIC MAP (2,6) PORT MAP (x41, x42, x_x(44));
  m4: pmos GENERIC MAP (2,6) PORT MAP (x_x(45), x42, x_x(43));
  m5: pmos GENERIC MAP (2,6) PORT MAP (x31, x_x(43), x_x(44));
  c45: cap GENERIC MAP (10 ffr) PORT MAP (x_x(45));
  s1: supply PORT MAP (x_x(45));
  c44: cap GENERIC MAP (13 ffr) PORT MAP (x_x(44));
  c43: cap GENERIC MAP (6 ffr) PORT MAP (x_x(43));
  c42: cap GENERIC MAP (4 ffr) PORT MAP (x42);
  c41: cap GENERIC MAP (8 ffr) PORT MAP (x41);
  c40: cap GENERIC MAP (10 ffr) PORT MAP (x_x(40));
  g1: ground PORT MAP (x_x(40));
  c37: cap GENERIC MAP (3 ffr) PORT MAP (x_x(37));
  c31: cap GENERIC MAP (5 ffr) PORT MAP (x31);
  m6: nmos GENERIC MAP (2,6) PORT MAP (x41, x_x(37), x_x(40));
  m7: nmos GENERIC MAP (2,6) PORT MAP (x_x(37), x_x(44), x_x(40));
  m8: nmos GENERIC MAP (2,6) PORT MAP (x41, x_x(43), x_x(44));
  m9: nmos GENERIC MAP (2,6) PORT MAP (x_x(40), x42, x_x(43));
  m10: nmos GENERIC MAP (2,6) PORT MAP (x31, x42, x_x(44));
  putbit (c1 ,"c1_vals.bit");
  putbit (i1 ,"i1_vals.bit");
END tgate_based;

```

Figure 12. Equivalent VHDL file for the CMOS latch of Figure 10

In the 2 Micron technology mentioned above, an example SPICE file for a CMOS latch with refreshing path is shown in Figure 10. The corresponding circuit is shown in Figure 11, and Figure 12 shows the VHDL file extracted from this SPICE file. Figure 13 shows a simulation run of this VHDL description. The input, *i1*, the clock, *c1*, the output, *o1*, and the point where input and output are multiplexed, *x\_x(44)*, are shown in this figure. The value pair of these signals within parenthesis show the logical value and the capacitance on each signal. As seen, the capacitance on signals changes during the simulation. For example, the output capacitance changes from 8 (femo farads) to 130 when the clock becomes 0. This is because the feedback path to the input of the latch inverter conducts, and the output sees two gate capacitances of the input of the inverter in addition to its own wire capacitance. The delay values at the output are .227 and .188 nano-seconds for the *tphl* and *tplh* respectively. These values correspond to .3 and .2 nano-seconds resulted from the SPICE simulation using the same transistor models.

ps	delta	i1	c1	x_x(44)	o1
0	+0	(T, 0)	(G, 96)	(G, 48)	(T, 0)
0	+1	(U, 5)	(G, 100)	(G, 61)	(T, 8)
0	+2	(U, 5)	(G, 100)	(G, 61)	(U, 8)
200000	+0	(1, 5)	(1, 100)	(G, 61)	(U, 8)
200000	+2	(1, 5)	(1, 100)	(Z, 61)	(U, 8)
200228	+0	(1, 66)	(1, 100)	(1, 61)	(U, 8)
200324	+0	(1, 66)	(1, 100)	(1, 61)	(1, 8)
800000	+0	(1, 66)	(0, 100)	(1, 61)	(1, 8)
800130	+2	(1, 5)	(0, 100)	(Z, 61)	(1, 8)
800147	+0	(1, 5)	(0, 100)	(1, 61)	(1, 69)
800222	+0	(1, 5)	(0, 100)	(1, 61)	(1, 130)
1200000	+0	(0, 5)	(0, 100)	(1, 61)	(1, 130)
1400000	+0	(0, 5)	(1, 100)	(1, 61)	(1, 130)
1400000	+2	(0, 5)	(1, 100)	(1, 61)	(1, 69)
1400081	+2	(0, 5)	(1, 100)	(Z, 61)	(1, 8)
1400092	+0	(0, 66)	(1, 100)	(0, 61)	(1, 8)
1400215	+2	(0, 66)	(1, 100)	(0, 61)	(Z, 8)
1400227	+0	(0, 66)	(1, 100)	(0, 61)	(0, 8)
1400228	+0	(0, 127)	(1, 100)	(0, 61)	(0, 8)
2000000	+0	(0, 127)	(0, 100)	(0, 61)	(0, 8)
2000000	+2	(0, 66)	(0, 100)	(0, 61)	(0, 8)
2000130	+2	(0, 5)	(0, 100)	(Z, 61)	(0, 8)
2000147	+0	(0, 5)	(0, 100)	(0, 61)	(0, 69)
2000222	+0	(0, 5)	(0, 100)	(0, 61)	(0, 130)
2400000	+0	(1, 5)	(0, 100)	(0, 61)	(0, 130)
2600000	+0	(1, 5)	(1, 100)	(0, 61)	(0, 130)
2600000	+2	(1, 5)	(1, 100)	(0, 61)	(0, 69)
2600081	+2	(1, 5)	(1, 100)	(Z, 61)	(0, 8)
2600092	+0	(1, 66)	(1, 100)	(1, 61)	(0, 8)
2600169	+2	(1, 66)	(1, 100)	(1, 61)	(Z, 8)
2600188	+0	(1, 66)	(1, 100)	(1, 61)	(1, 8)
2600228	+0	(1, 127)	(1, 100)	(1, 61)	(1, 8)

Figure 13. Simulation output of VHDL description of Figure 12

## 7. Conclusions

Modeling transistors for fast switch level simulation was shown in this paper. The method enables the use of VHDL for simulation of large VLSI circuits. The tools discussed here enable the use of powerful VHDL based tools in VLSI design environments. Not all VLSI constructs can be modeled by unidirectional switches described here. However, most common structures, are easily mapped to our proposed models.

## References

1. Navabi, Z., "VHDL: Analysis and Modeling of Digital Systems", McGraw Hill Publishing, New York, N.Y., 1993.
2. J. Dube and Z. Navabi, "Behavioral VHDL transistor slope models", Proceedings of the 1991 IEEE ASIC Conference and Exhibit, Sept. 1991.
3. Navabi, Z., A. Hashemi, M. Eghtesad, and M. Vai, "Modeling Timing Behavior of Logic Circuits Using Piecewise Linear Models", Proceedings of the 1993 CHDL Conference, Ottawa, Canada, April 1993.