

Behavioral VHDL Transistor Models

John Dube and Zainalabedin Navabi
VIEWlogic Systems, Inc.
293 Boston Post Road West
Marlboro, Massachusetts 01752
jdube@viewlogic.com

Electrical and Computer Engineering Department
Northeastern University
409 Dana Research Center
Boston, Massachusetts 02115
navabi@nuvlsi.northeastern.edu

Abstract

In order to accurately simulate the logic and timing properties of a custom designed circuit layout, behavioral CMOS transistor models were written using the VHSIC Hardware Description Language (VHDL). A simple netlist translator program maps the layout netlist to an equivalent transistor level VHDL architecture which can be directly simulated, either as a standalone component or as a system module. Simple VHDL constructs and existing delay models were used to develop the transistor models that provide the design engineer with accurate circuit timing and logic information for simulation with higher level models.

1.0 Introduction

In a top-down design environment, hardware description languages are often used to model custom VLSI circuits. VHDL provides a description and simulation environment for several levels of abstraction, including at the behavioral, dataflow, and structural levels. Designs already at the layout level, however, cannot be simulated using VHDL, and cannot be simulated simultaneously with higher level models. For system design where some components of the system have already been designed to the layout level, the integration of the simulation environment for high level models and the layout level components would give the accurate logic and timing behavior of layout level components to the high level system description. Developing behavioral transistor models in VHDL can accomplish this integration of the simulation environment. The transistor models, however, need to be efficient and accurate with respect to both logic and timing. This paper will show how transistor models were developed that can achieve these goals. The techniques illustrated in this paper take advantage of existing delay models and the rich hardware description language and simulation environment provided by IEEE Std. 1076 VHDL¹.

logic value. The logic value to be driven by the transistor onto the source terminal is found at the intersection of the gate logic value and the drain logic value in the drive table. Since the transistor is symmetrical, the same drive table can be used to find the logic value to be driven at the drain terminal, based on the logic values at the gate and source terminals. The *n_drive_table* is used for n-type MOS transistors, and the *p_drive_table* is used for p-type MOS transistors.

3.1 Resolution Functions

The resolution functions for the three fields of each node have been developed to accomplish the modeling strategy outlined in Section 2. Figure 2 shows the three fields of the record that fully describes the logic and timing values associated with a node.

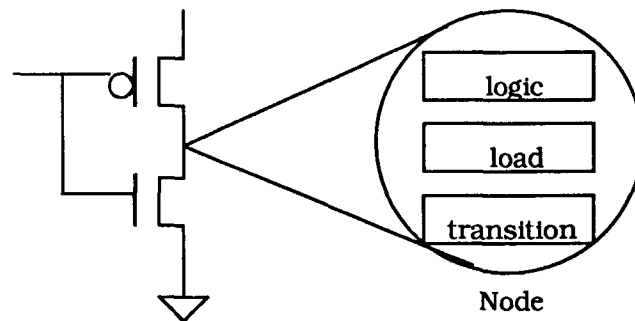


FIGURE 2. Construct of a node

3.1.1 Logic Resolution Function

The *logic* field, of type *seven_value*, stores the logic value of a node, which can be one of the seven values defined in the *seven_value* type definition. The *wiring* resolution function, shown in Figure 3, reads the logic value outputs of each transistor connected to that node, and determines a resulting logic value. The resolution function uses a look-up table that filters out weaker logic values and detects conflicts, and returns the logic value to assign to that node. This function accurately models the behavior of a node in an actual circuit.

```

FUNCTION wiring (drivers : seven_vector) RETURN seven_value IS
  VARIABLE wire_value : seven_value := 'U';
  CONSTANT table : seven_value_table :=
    (('U','Z','L','H','0','1','X'),
     ('Z','Z','L','H','0','1','X'),
     ('L','L','L','X','0','1','X'),
     ('H','H','X','H','0','1','X'),
     ('0','0','0','0','0','X','0'),
     ('1','1','1','1','X','1','1'),
     ('X','X','X','X','0','1','X'));
BEGIN
  FOR i IN drivers'RANGE LOOP
    wire_value := table (wire_value,drivers(i));
  END LOOP;
  RETURN wire_value;
END wiring;

```

FIGURE 3. *Wiring* Resolution Function

3.1.2 Load Resolution Function

The *load* field, of type *pr* (positive real), stores the capacitance value at the node. The loading resolution function, shown in Figure 4, calculates the capacitance at each node by summing up all of the capacitance values reported to it by transistors connected to that node. This is a very efficient scheme, since the loading function is called only once during the initialization phase.

```
FUNCTION loading (caps : pr_vector) RETURN pr IS
  VARIABLE load_value : pr;
BEGIN
  load_value := 0.0;
  FOR i IN caps'RANGE LOOP
    load_value := load_value + caps(i);
  END LOOP;
  RETURN load_value;
END loading;
```

FIGURE 4. *Loading* Resolution Function

3.1.3 Transition Resolution Function

The *transition* field, of type *real*, stores the rate of change of voltage, or *edge speed*, of a node that is changing logic values. When a transistor drives a node, it also reports the edge speed of the corresponding logic value change to the node. The *quickest* resolution function, shown in Figure 5, takes the fastest edge speed reported and assigns it to the transition field of the node. This field is then used by all transistors driven by that node to determine their *effective resistance* and *output edge speed*.

```
FUNCTION quickest (edgespeeds : real_vector) RETURN REAL IS
  VARIABLE trans_value : REAL := -1.0;
BEGIN
  FOR i IN edgespeeds'RANGE LOOP
    IF (edgespeeds(i) >= 0.00) THEN
      IF ((trans_value <= 0.0) OR (edgespeeds(i) < trans_value)) THEN
        trans_value := edgespeeds(i);
      END IF;
    END IF;
  END LOOP;
  RETURN trans_value;
END quickest;
```

FIGURE 5. *Quickest* Resolution Function

3.1.4 Other Utility Functions

Also included in the *mos_utilities* package is a procedure that interpolates values from a table. This procedure is used to find the effective resistance and output edge speed as a function of the input edge speed ratio. These values are stored in a table as a set of triplets, and the function performs linear interpolation to approximate the output values. In addition, there are functions in the *mos_utilities* package to map from the *seven_value* logic system to other logic value systems that may be used by higher level VHDL models. This facilitates the translation from a node of type *line* to a simpler node type used in higher level VHDL models.

4.0 VHDL Implementation of Transistor Model

The VHDL transistor models follow the modeling strategies outlined in Section 2, and take advantage of the utilities mentioned in Section 3. The transistor ports (gate, source, and drain) are all of type *line* and each of the fields are readable and writable (INOUT). This is necessary to read and write the load capacitance, logic, and edge speed fields of the connected nodes. Bidirectional transistor models have been developed that model the signal flow from the source to the drain, and from the drain to the source, when the transistor is conducting. A simplified, unidirectional model has also been developed for more efficient simulation when the bidirectional nature of the transistor is not necessary to model. Any transistor with its source or drain input connected to a supply source (Vdd or GND), or circuit input can be modeled as a unidirectional device with the signal flow away from that node. Any signal flow through a transistor towards the logic source or circuit input will be disregarded because of the strength of these nodes. Therefore, each transistor in the circuit layout can be represented by either the unidirectional or bidirectional switch models. Figure 6 shows the pseudo-code used to describe the behavior of a bidirectional transistor.

```
USE WORK.mos_utilities.ALL;
ENTITY switch IS
  GENERIC (length,width : pr);
  PORT (gate,source,drain : INOUT line);
END switch;

ARCHITECTURE bidirectional OF switch IS
  SIGNAL effective_gate : wired_seven;
  CONSTANT define capacitance constants;
BEGIN
  initialize node values (load, logic, and transition);

  gating: PROCESS(gate.logic)
  BEGIN
    if gate is high impedance, discharge effective_gate;
    else effective_gate <= gate.logic;
  END PROCESS gating;

  outing: PROCESS(source.logic,effective_gate,drain.logic)
  variable definitions;
  BEGIN
    Detect which input changed, find edge speed of changing input;
    Make sure that node did not change as a result of this transistor;
    Get delay data from data table using input edge speed ratio;
    Calculate output delay and edge speed;
    Assign output values to source and drain AFTER output delay;
  END PROCESS outing;
END bidirectional;
```

FIGURE 6. Transistor Model Structure

The transistor model uses two processes to describe the behavior of the transistor. The first process, *gating*, describes the behavior of the transistor when the gate node is left in a high impedance state. When a node is undriven, it will eventually discharge to an unknown logic state because of leakage current in the node capacitance. The gating process schedules an 'X' on the internal gate node after a specified discharge time. This will, in turn, cause the transistor to drive an 'X' to the source and drain terminals so that the designer will be alerted that a problem condition exists. The second process, *outing*,

describes the signal flow through the transistor. The process is activated by a change in the logic fields of the source and drain nodes, or by a change in the internal gate node, which would be caused by a change in the logic field of the gate terminal. Once the *outing* process is activated, the direction of signal flow is found by detecting which input changed and which is the dominant (stronger) node. The model also makes sure that the cause of the process activation was not a result of an event in this transistor. This eliminates the possibility of cyclical behavior. The output logic value is then determined by looking up the result value in the *drive_table*. The effective resistance and output edge speed values are then retrieved from the *data* package, based on the input edge speed ratio and the driving logic value. The output delay and edge speed are then found by scaling the effective resistance and output edge speed by the output load capacitance and transistor size. Finally, the output values are assigned to the source and drain nodes. The source of the signal flow is assigned a 'Z' value immediately to show that this transistor is driving in the opposite direction, and does not contribute to that node.

5.0 Extraction and Simulation

This section will discuss how the timing data used in the transistor models is obtained, and how a VHDL architecture is extracted from the layout netlist. According to the delay calculation strategy outlined in Section 2, the delay information of a transistor, namely the effective resistance and output edge speed, must be found over a wide range of input edge speeds. This information is obtained by a program called *mkvp* ("make VHDL parameters"). This procedure takes SPICE MOSFET models as inputs, calculates the necessary transistor characteristics, and outputs them into a VHDL-usable format, called the *data* package. This package is then included by the transistor models described above to make timing calculations. Once the transistor models are complete with timing and logic information, they can be instantiated in a VHDL architecture. The VHDL equivalent to the layout netlist that is to be simulated will be created by a program called *sim2vhdl*. This program reads the SIM format layout netlist and writes the corresponding VHDL switch level architecture. Figure 7 summarizes how the programs *mkvp* and *sim2vhdl* interact with layout netlists and SPICE MOSFET models to provide a VHDL architecture that can be directly simulated.

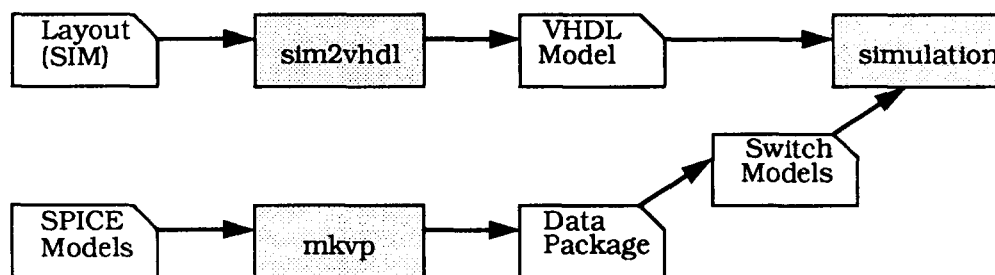


FIGURE 7. Extraction Program Summary

6.0 Conclusions

This paper shows how behavioral transistor models can be developed that can be useful when a designer needs a high level model of a circuit whose layout level description is available. Accurate logic and timing information can be discerned from the layout, once it is mapped to an equivalent VHDL architecture. This mapping is simple once complete

transistor models and necessary utilities have been developed. The methods presented in this paper provide an alternative method to accomplish the timing and logic verification of a circuit, and at the same time, building a switch-level architecture that can be simulated along with other system modules in a multi-level simulation. The resulting simulation is much more efficient than using a circuit-level simulator, such as SPICE, with timing results typically within 10% of such circuit-level simulators.

REFERENCES

- [1] "IEEE Standard VHDL Language Reference Manual", IEEE Std 1076- 1987, The Institute of Electrical and Electronic Engineers, Inc., 1988.
- [2] J.K. Ousterhout, "Switch-Level Delay Models for Digital MOS VLSI", Proceedings IEEE 21st Design Automation Conference, 1984, pp 542-548.
- [3] J.K. Ousterhout, "A Switch-Level Timing Verifier for Digital MOS VLSI", IEEE Transactions on Computer Aided Design of Integrated Circuits, July 1985, pp 336-349.
- [4] Coelho, D. "VHDL Handbook", Kluwer Academic Publishing, Boston, 1988.
- [5] Armstrong, J.R., "Chip-Level Modeling with VHDL", Prentice-Hall Inc., 1988.
- [6] Lipsett, L., Carl Schaefer, and Cary Ussery, "VHDL: Hardware Description and Design," Kluwer Academic Publishing, Boston, 1988.
- [7] Masakazu Shoji, "CMOS Digital Circuit Technology," Prentice Hall, Englewood Cliffs, New Jersey, 1988.
- [8] Z. Navabi, J. Dube, and A. Huang, "Modeling Strategy for Post Layout Verification," Proceedings of the 1990 IEEE ASIC Seminar, September 17-21 1990.
- [9] Z. Navabi, "Behavioral Level Modeling of Gate Level Loading Effects," 1991 International Conference on Computer Hardware Description Languages, April 1991.
- [10] J. Dube and Z. Navabi, "Behavioral VHDL Transistor Slope Models," Proceedings of the 1991 IEEE ASIC Conference and Exhibit, September 1991.