Behavioral Modeling of Transmission Gates in VHDL*

Steven S. Leung

Department of Electrical Engineering
Michigan State University
East Lansing, MI 48824

Abstract

This paper presents a technique for describing the behavior of transmission gates (TGs) in VHDL. The concept of virtual signal is introduced into the TG's data structure to represent the nature of the connection. The model's semantics are coded in three parts: the state transition, the steady states, and the connecting protocol. Simulation results indicate that the model is correct and robust.

I. Introduction

VHDL allows description of hardware behavior down to the gate level, but modeling of switching elements at the more primitive transistor level is not supported [1–3]. In real world designs, especially in bus-oriented hardware designs, however, the transmission gate is an indispensable component. Thus, a TG model, even at a functional level, is much more desirable and will certainly enhance the utility of the language. In this paper, a technique for modeling the transmission gate in VHDL is presented. The approach of this technique is based on a critical observation that data flow is the primary focus of architecture design, and the direction of the data flow is a property that can be derived from the connecting circuit itself. The rationale of this approach is explained in the next section.

The VHDL TG model is presented in Section III. It is described in terms of data structure and semantics designs. The data structure reflects the two abstract properties of the transmission gate I/O ports, namely, the information about the data type and the nature of the connections. The semantics are coded in three parts: the state transition, the steady states, and the connecting protocol. The model has been tested in a bus-based ASIC processor design. The experiments on the model are described in Section IV.

II. The Behavior of TG: What do we Want to Model?

In a given circuit, transmission gates are used in two different contexts: realization of logic functions and node access controls. The former refers to the use of TG circuits to realize logic functions such as XOR, addition, multiplexing, etc. For this type of circuits, the lack of a transmission gate model in VHDL does not seem to be a major obstacle in architecture designs, since the logical equivalents are readily constructed through a higher level abstraction [4]. This is also made clear by the fact that the XOR function itself is a built-in logical operator provided by the language. Thus, the modeling effort of this work is focused on the use of transmission gates in the second context. The goal is to develop a VHDL model that can be instantiated in bus-oriented architecture designs. To achieve this goal, the TG's behavior is first investigated by considering a simple circuit in which an inverter output is connected to an inverter input through a transmission gate. The result is then extended to the situation of a general bus architecture.

Fig. 1 shows the flow direction of the transient current when the transmission gate is turned on. Note that while the current (or

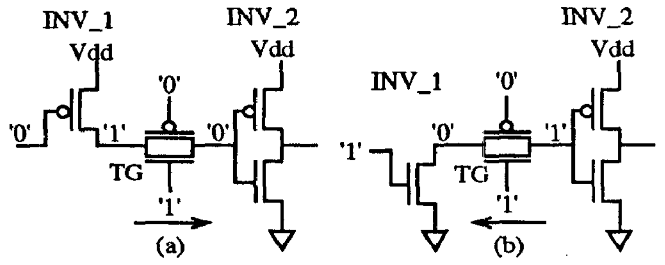


Fig. 1. The direction of current flow in an INV-TG-INV circuit.

electron) flow direction depends on the data value, the outcomes differ. In 1(a), both the output node of INV_1 and the input node of INV_2 will become '1'; but in 1(b) both nodes will become '0'. This suggests that the key to characterizing the behavior of the transmission gate lies not in the flow of current, but in the flow of data, even though the former is the mechanism that realizes the latter. Furthermore, the difference between the outcomes of 1(a) and 1(b) is due to the finite size of the load capacitor. In other words, the circuit to which the TG is connected, or the nature of the connection, determines the direction of the data flow. Obviously, this is a piece of information that should be combined with the data content and input to the transmission gate model.

In order to encode the nature of the connection into the model properly, the use of the transmission gates in a more general setting is considered. Fig. 2 illustrates a hierarchical bus system

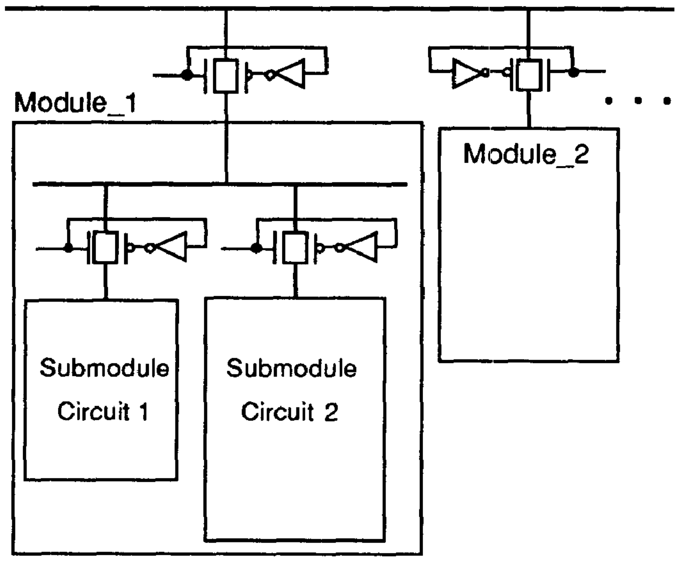


Fig. 2. A hierarchical bus architecture.

* This research was supported in part by the State of Michigan Research Excellence for Economic Development Fund.

in which multiple circuit modules communicate with each other through the bus. The TGs play the role of controlling the bus access. For generality, it is assumed that each module may have its own internal bus as illustrated by Module_1. Comparing Fig. 1 and Fig. 2, it is obvious that the input signal to the TGs that specifies the nature of the connection must be dynamic. Moreover, some of the TGs do not directly connect to either a load or a current source, but serve only as a messenger to pass along the data. At steady states, however, if a valid path exists, then it is characterized by one and exactly one driving source with one or more loads. Accordingly, Fig. 1 can be generalized to account for this situation by adding intermediate TGs in the path as shown in Fig. 3. Another generalization is to replace the

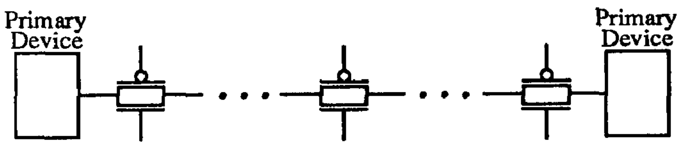


Fig. 3. A generalized valid data flow path at steady state.

inverter circuits at the two ends of the TG in Fig. 1 with two generic circuits called primary devices so that data can flow in either direction. The concept of primary device is associated with the concept of virtual_signal, and both are to be explained next. With these modifications, the situation of multiple loads can be viewed as the sum of multiple distinct paths each of which consists of the same driving source but a distinct load.

Based on the foregoing analysis, a type *virtual_signal* is defined to represent the nature of the connection, and thus the direction of the data flow. The allowed values and their meanings are specified in Table 1. A virtual_signal value of 1 designates a data source (driver) and a value of –1 designates a load (reader).

Table 1. Virtual_Signal Values and Connection Types.

Virtual_Signal Value	Connection Type (as seen by the receiver)
–1	reader
0	open
1	driver
2	request status

A value of 0 indicates physically an open connection, electrically a high-impedance state, and logically a don't care condition. The value 2 does not specify a connection type, rather, it is used by the TGs as an interrogation signal to configure the data flow dynamically. In addition, the circuit devices to which the TGs are connected can be classified either as primary device or secondary device. Primary device can originate a virtual_signal value that specifies a connection type while secondary device can only pass along the virtual_signal value. Five basic primary devices and their ranges of virtual_signal values are listed in Table 2.

Table 2. Primary Devices and their Ranges.

Primary Device	Virtual_Signal Value Range
gate input	{–1}
gate output	{1}
tristate output	{0, 1}
bi-directional buffer	{–1, 1}
bi-directional buffer with tristate output	{–1, 0, 1}

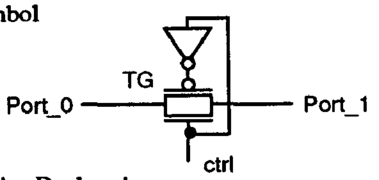
III. A VHDL Model of TG

With the question of what to be modeled settled, we now proceed to describe how the model is built. The VHDL model of the transmission gate is constructed in terms of data structure and semantics. The data structure represents the static abstract properties, i.e., information about the type of data and the nature of the connections, of the transmission gate, while the semantics describe operations on the data structure. The interpretation of the semantics constitutes the dynamic behavior of the transmission gate.

3.1 Data Structure

Without loss of generality, the TG circuit is considered as a three-port device with two I/O ports controlled by a single signal ctrl. Fig. 4 shows the TG model's VHDL entity declaration and the corresponding graphic representation. The

(a) Circuit Symbol



(b) VHDL Entity Declaration

```
entity Transmission_Gate is
  port (Data_0, Data_1 : buffer INTEGER := 0;
        D0_in, D1_in : in INTEGER;
        state0_out, state1_out : out VIRTUAL_SIGNAL := 0;
        state0_in, state1_in : in VIRTUAL_SIGNAL;
        ctrl : in BIT);
end Transmission_Gate;
```

(c) Graphic Representation of the Data Structure

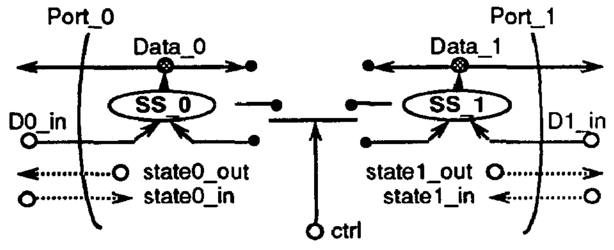


Fig. 4. The TG's entity declaration and data structure.

control signal ctrl is of type BIT and each I/O port is translated into four signals accessible from the outside world. Among these four signals, Data_i of an appropriate type in interest represents the "official" value of Port_i. The buffer mode is used for this signal so that it can be read from both inside and outside but can only be driven within the TG. An auxiliary input port and a resolution process are created to resolve the "official" value. Since the language has no restriction on reading a port value from outside an entity, accomplishment of the "official" status of Data_i can only rely on programming discipline. In other words, it is the programmer's responsibility to refrain from reading Di_in anywhere in the program.

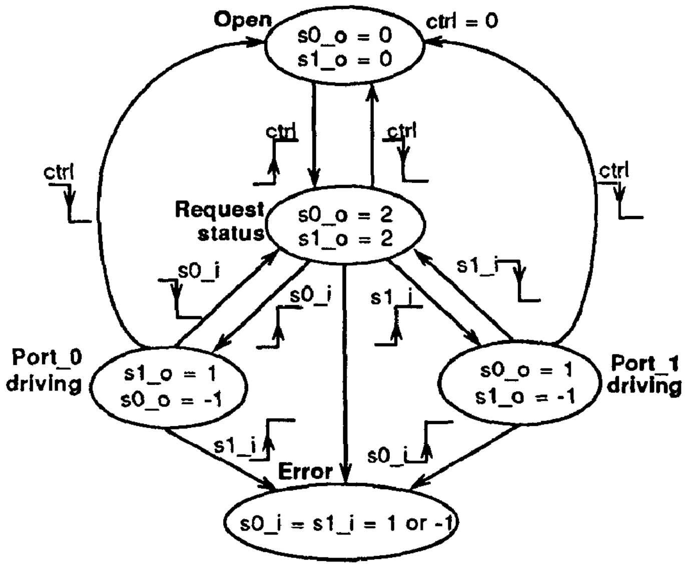
In addition, each I/O port also has two virtual_signals as indicated by the dotted lines in the figure. They do not correspond to physical wires but are used for encoding the connection types so that information of the data flow direction can be specified in the semantics of the TG body and derived dynamically.

3.2 Semantics

The semantics of the TG's behavior are partitioned into three parts of the state transition, the steady states, and the connecting protocol. The first two are realized by the process statements within the TG architecture body. The third part stipulates how the outside world interacts dynamically with the TG and is implemented by both the TG and the connecting processes.

3.2.1 The State Transition

The state of the TG is defined by the signal ctrl and the four virtual_signals. Since changes in data signals are ultimately originated from the external world, these data signals are not considered as constituents of the TG's internal state. The state transition (Fig. 5) is implemented by a VHDL process statement in the TG's architecture body (Fig. 6).



* s0_i(o) = state0_in(out)
s1_i(o) = state1_in(out)

Fig. 5. The state transition of the TG.

Initially, the TG is in a state of open connection characterized by a ctrl signal value of '0' and all four virtual_ signals equal to 0. This is also the state to which the TG will return from other non-error states whenever the signal ctrl experiences a '1' to '0' transition. When the signal ctrl rises from '0' to '1', the TG will set the output virtual_signals of both I/O ports to 2 and then monitor the responses on the two input virtual_signals. A single driving signal (virtual_signal value 1) from either port will cause the TG to enter one of the two connecting (driving) states. At any time, the two input virtual_signals having the same value of 1 (both driving) or –1 (both reading) will render the TG to enter the error state and the simulation will terminate. Essentially, this process performs constraint violation checking and executes the protocol on the part of the TG to configure the data flow direction.

3.2.2 The Steady States

Among the four allowed states in the state transition diagram, only three correspond to physical configurations. They are the open state and the two driving states — one from Port_1 to Port_0, and the other from Port_0 to Port_1. The two driving states can be visualized as effects on the data structure as illustrated in Fig. 7. Thus, the implementation of the steady states is manifested as the problem of assigning the right value at the right time to the I/O port's "official" value holder. This is done by a process statement for each port. The process statement for Port_0 (SS_0) is listed in Fig. 8. Essentially, this process responds to the driving condition of the other port (Port_1) by assigning the value of Data_1 to Data_0. When Port_1 is not driving, it then responds to the change of D0_in and assigns it to Data_0.

```
ST: process
      variable s0, s1 : VIRTUAL_SIGNAL;
    begin
      state0_out <= 0;                  -- TG in open state
      state1_out <= 0;
      wait until ctrl = '1';
      while ctrl = '1' loop
        state0_out <= 2;                -- Request status
        state1_out <= 2;
        wait on state0_in, state1_in, ctrl;
        if ctrl = '1' then
          s0 := state0_in;
          s1 := state1_in;
          assert not ((s0 = 1) and (s1 = 1))
            report "Both ends of the TG are driving"
            severity ERROR;
          assert not ((s0 = -1) and (s1 = -1))
            report "Both ends of the TG are reading"
            severity ERROR;
          if s1 = 1 then                -- Port 1 driving
            state0_out <= 1;
            state1_out <= -1;
            while ctrl = '1' loop
              wait on state0_in, state1_in, ctrl;
              assert state0_in /= 1
                report "Both ends of the TG are driving"
                severity ERROR;
              if state1_in /= 1 then exit; end if;
            end loop;
          elsif s0 = 1 then             -- Port 0 driving
            state0_out <= -1;
            state1_out <= 1;
            while ctrl = '1' loop
              wait on state0_in, state1_in, ctrl;
              assert state1_in /= 1
                report "Both ends of the TG are driving"
                severity ERROR;
              if state0_in /= 1 then exit; end if;
            end loop;
          end if;
        end if;
      end loop;
    end process ST;
```

Fig. 6. The process statement implementing the state transition.

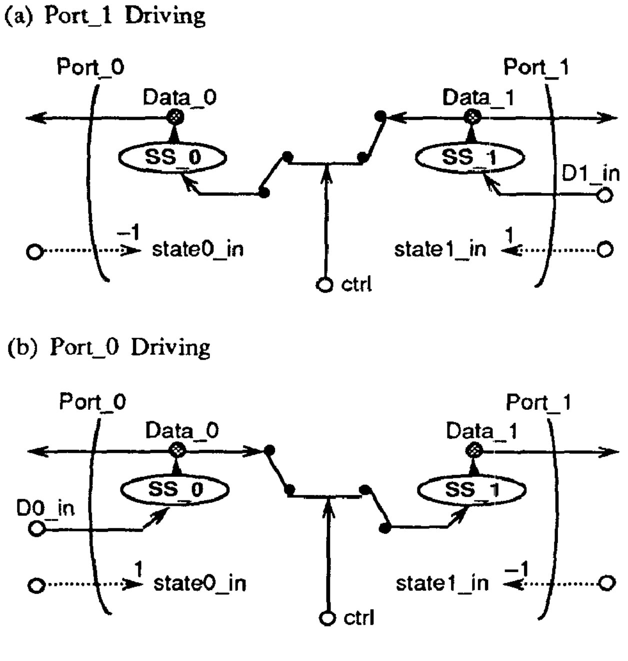


Fig. 7. The data path of the two driving steady states.

```
SS_0: process
      begin
        if ctrl = '0' then              -- TG OFF
          wait on ctrl, D0_in;
          if not D0_in'STABLE then
            Data_0 <= D0_in;
          end if;
        else                              -- TG ON
          if state1_in = 1 then       -- Port 1 driving
            Data_0 <= Data_1;
            wait on Data_1, state1_in, ctrl;
          else
            wait on D0_in, state1_in, ctrl;
            if not D0_in'STABLE then
              Data_0 <= D0_in;
            end if;
          end if;
        end if;
      end process SS_0;
```

Fig. 8. The process statement imlementing the steady states.

3.2.3 The Connecting Protocol

The simulated behavior of the TG inherently depends on the nature of the connection, and the connecting protocol defines how this information is passed to the TG during simulation. The protocol involves both the TG and the connecting device, with the connecting device playing a passive role. That part of the protocol for the connecting device is also divided into the two processes of state transition and steady states. The implementation can be described algorithmically as follows:

```
• State_Transition Process:
   { if state_out (from TG) changes to 2 then
        report connection_type via state_in;
     elsif value of state_out drops from 1 then
        send the current state_out value to readers;
     end if; }
• Steady_States Process:
   { if state_out (from TG) = 1 then
        assign Data value of TG to readers;
     elsif a driver exists then
        assign the current driver's value to D_in;
     end if; }.
```

IV. Simulation Experiments

VHDL programs have been developed to test the TG model in three different contexts. The VHDL programs were run on the Intermetrics VHDL simulator installed on a MicroVax II. In the first experiment, the TG is tested as a stand-alone device with integer as the data type for the I/O ports which are connected to separate controlled driving sources. The TG's control port is connected to a clock signal which turns on for 30 ns every 100 ns starting at 50-ns. Data are then driven to the two ports with different timing characteristics to test various dataflow cases, and the simulation results have confirmed that the model exhibits the intended behavior. To further test the robustness of the TG model, two TGs are instantiated and connected in series. The TG-to-TG connections are directly made from one TG's out-mode port to the other's in-mode port as shown in Fig. 9. The control ports of both TGs are tied to the same clock signal and the same input pattern of the one-TG experiment is applied to the open ends of the two-TG circuit. As expected, the simulation result is the same as before except that some of the signals settle down at a greater delta cycle.

In the second experiment, a TG entity is instantiated in a circuit module to control the connectivity between a register's I/O port and a bus so that a second source (directly from another circuit module) can write to the register [5]. The three signals — the register's I/O port, the second source, and one end of the TG

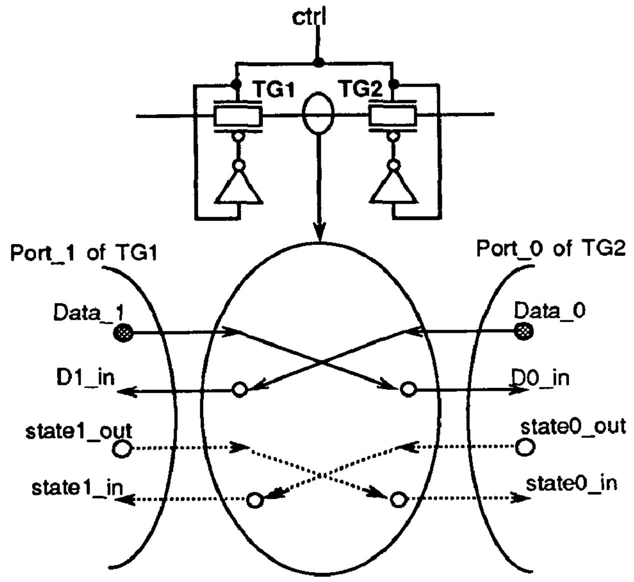


Fig. 9. The TG-to-TG connection.

forms an internal bus. To resolve the value of the internal bus, a resolution mechanism consisting of a transition process and two steady state processes is developed to implement the connecting protocol described previously. The module is then tested for its functional specifications which involve both read from and write to the register through the TG.

The third experiment involves instantiation of the TG models in different circuit modules (one of which is the register module just described). These modules, as part of the datapath in a special ASIC processor design, communicate with each other via a bus. Simulation results have again confirmed that the TG model correctly produces the intended behavior.

V. Summary

A VHDL TG model for architecture design has been presented. The basic idea of this work is to characterize the types of a TG's connections with the concept of virtual_signal so that the TG can configure the data flow dynamically. The major focus is on providing constructs and mechanisms for modeling TGs in the context of controlling node access. The model has been tested in a special ASIC processor design [5] and simulation results indicate that the TG model is both correct and robust.

As the model is developed based on a general bus architecture and the connecting process is algorithmically defined, the model can be generated automatically from circuit schematics and can be incorporated in existing CAD tools/systems transparent to the user. Extending the results of this work to the cases where TGs are used to realize logic functions appears promising. Also, timing delays in this work are implicitly assumed to be lumped at external connecting signals. A more elaborate model incorporating accurate timing information will certainly enhance the utility of the model.

References

[1] VHDL Language Reference Manual, IEEE, 1988.
[2] VHDL User's Manual, vol. II, User's Reference Guide, Intermetrics, IR-MD-065-1, Aug. 1985.
[3] M. Shahdad, "An overview of VHDL: language and technology," 23rd DAC, 1986, pp. 320–326.
[4] J. Armstrong, Chip Level Modeling with VHDL. NJ: Prentice Hall, 1989.
[5] S. S. Leung and M. A. Shanblatt, ASIC System Design with VHDL: A Paradigm. MA: Kluwer Academic, 1989.