TheAutomatic Generation of Bus-Interface Models* 
Yew-Hong Leong William P. Birmingham 
Electrical Engineering and Computer Science Department The University of Michigan Ann Arbor, MI, USA 48109

Abstract

We describe a system, HIDE, that automatically creates VHDL bus-interface models from timing and bus-state diagrams. HIDE shrinks model generation time to days from months. An Intel iAPX80386 model was developed in less than one week. HIDE can also be used to develop executable specifications, as its inputs are a normal element of the IC design process.

1. Introduction

Simulation of digital systems is common design practice. For effective simulation, accurate models of devices must be available. Model development for highly complex VLSI components is time consuming, sometimes requiring several man-years of effort. This effort is spent in understanding how the device operates from its documentation, and writing the model. Often, models are produced by third parties, not by the component designers. For these groups, understanding the operation of a component is a major task; consider that complete microprocessor descriptions fill several hundred pages.

The modeling process can be hampered by hardware description languages (HDL). While HDLs are intended to facilitate model creation, they are often complex, and require considerable programming skill to be effectively used. Thus, a model writer must be an expert in both hardware and HDLs.

The hardware-modeling task can benefit from automation. In this paper, we describe HIDE (HDL Interface Model Designer), which automatically creates VHDL [7] bus-interface models (BIM) from timing and bus-state diagrams. The information required by HIDE is easily found in component documentation. Thus, HIDE reduces the time for both understanding a component's operation, and in writing the model.

This paper is organized as follows. BIMs are described in Section 2, followed by an overview of HIDE in Section 3. HIDE's algorithms are sketched in Section 4, with experimental results given in Section 5. Section 6 compares HIDE to related tools. Section 7 concludes the paper.

2. BIMs

Simulation models for complex components generally are behavioral or BIM, or some combination. Behavioral models implement the full functionality of a component. These models are complex to build, and simulate relatively slowly.

BIMs, also known as hardware-verification [11], bus-functional [4], and chip-level models [2], describe the interface behavior of a device. These models capture a component's signal activity on a bus-cycle2 by bus-cycle basis. BIMs fit the typical CAE environment, where a schematic is drawn and then simulated. A BIM allows the designer to ensure that a component is communicating properly with other devices. Since a BIM implements a sufficient subset of a component's behavior, it executes more quickly than behavioral models.

This work was funded by Digital Equipment Corp. and by the National Science Foundation grant MIPS-905781.

Fig. 1: Bus-state and timing diagram (adopted from [8]).

Bus activity consists of cycles, which are gross bus behavior. Example microprocessor cycles are read and write. Each cycle consists of one or more bus states. For example, the Intel 80386's read cycle is composed of two bus states, T1 and T2 [8]. For some devices, it is possible to map a cycle onto different bus states. Continuing the 80386 example, pipelined read and write allows address and control lines to appear one state earlier than normal read and write. For the pipelined case, the bus cycle has states T1P and T2P [8]. The movement from non-pipelined to pipelined is given by the level of control signals. For example, in Figure 1

---

2 Bus cycle is used in a general form; a device's interface does not have to be bus-oriented to be amenable to a BIM-style of modeling.
going from T1P to T2P occurs when the signals satisfy the following: \text{Hold} \cdot \text{NA#} \cdot \text{Request Pending}^3 (1).

Often, this state-transition control information is given as a bus-state diagram or as text. A portion of the 80386 bus-state diagram for pipelined bus cycles is shown in Figure 1. The complete state diagram is more complex in both the numbers of states and transitions. Note that the state diagram is not specific to a cycle type; the state diagram in Figure 1 applies to both read and write.

The bus state in models generated by HIDE is automatically determined by the signal level. Thus, a BIM will evaluate Equation (1) to determine if the next state should be pipelined. It is notable that HIDE can create models with this functionality.

Each signal's assertion level for each state in each cycle is given as (a) timing diagram(s), which also specifies the time between events. An event is a change in the level of a signal.

```vhdl
#Command File for read/write of a processor
variable i: integer;
variable start_address: vbit_vector(31 downto 0);
variable return_data: vbit_vector(31 downto 0);
start_address := "00";
for i in 1 to 10 loop
  write(2, start_address, "AA");
  read(2, start_address, return_data);
  start_address := increment_address(start_address);
end loop;
```

Fig. 2: Command file for microprocessor read and write.

BIMs are composed of procedures for each cycle. A microprocessor BIM has many procedures that are assembled in a command file by a model user to simulate interface behavior. A typical BIM procedure header for a read cycle is:

```vhdl
read(number_of_bytes, start_address, return_data)
```

which will return number of bytes of return_data when start_address is supplied. An example command file is given in Figure 2. The model user must interpret the datum returned.

A BIM is created by HIDE using only timing diagrams and bus state diagrams, as well as information about both the pins of a component and how certain bus activities (data control) are performed. Pin information lists each input and output signal, its corresponding pin number on a schematic symbol, and the pin type (input, output, time multiplexed, and so forth). Data-control information is explained later. Because all this information is readily available from component documentation, creating BIMs is straightforward, and does not require detailed knowledge of component operation.

Since timing and bus-state diagrams are normally produced as part of the component design process, BIMs can be used as executable specifications.

3. System Organization

HIDE is shown in Figure 3. For capturing timing diagrams, the XWave tool is provided. XWave's output is a tabular representation of the timing diagram, which is converted to an event graph (EG) [3] by HIDE. XWave also captures bus-state diagrams\(^4\). Pin (signal) information is supplied through a file.

Many bus activities involve some form of data transfer. Information regarding how the data is arranged on the data bus is not represented in either the timing or state diagrams. This information is found in the text of a data book. For example, during the 80386 memory-read cycle, the location of the valid data on the bus is given by the BS16# and BE signals. The designer provides this information through the data-control file.

4. Generating VHDL for a BIM

HIDE generates VHDL procedures for each bus cycle of a component. Consider Figure 1, the bus cycle begins with state T1P, and depending on some conditions, the bus cycle transits to either T2I (idle state) or T2P. Even though the timing diagram shows all three states in the bus cycle, it does not fully describe the transition between states. Thus, both state and timing diagrams are required.

\[^3\text{Req. Pending is not a signal, but an indicator in the 80386 that some bus activity is required.}\]

\[^4\text{This version is being constructed; tables are used now.}\]
the general structure of this procedure in VHDL. The procedure initializes the Next State variable to the initial state of the bus cycle. Then, the procedure enters a while loop with case statements for each state of the bus cycle. Within each state, code is generated from the EG, as described in Section 4.3. Code to determine the next state of the bus cycle is easily translated from the state diagram. If necessary, code for proper placement of data on a bus is generated.

procedure Bus_Cycle_X
(Input_Parameter1, ... Output_Parameter) is
Next_State := Initial State;
while (NOT (Next_State = EXIT)) loop
  case Next_State is
    when State1 => code for State1 converted from EG of State1
    # code for transfer of data if State1
    # involves data transfer
    Next State :=
      Get_Next_State(Next_State, Variable1, ... );
    when State2 =>...
    end case;
  end loop;
  end Bus_Cycle_X;
end procedure;

Fig. 4. General structure of a VHDL procedure.

Specific input and output parameters for each procedure in a BIM are required. A model builder must specify these parameters to HIDE, and describe how to assign or access them. This information, also input via the data-control file, is automatically inserted in the proper place in the VHDL procedure. Figure 5 shows typical parameter access and assignment specification.

Input Parameters: Parameter1, Parameter2;
Output Parameters: Parameter3;
State: StateX;
If (Parameter1 = Value1 and Parameter2 = Value2) then
  Parameter3 := Value_1
else
  Parameter3 := Some_Other_Value;
end if;

Fig. 5. Typical parameter access and assignment specification.

4.2 Description of the EG

XWave produces a set of timing dependencies from the timing diagrams:

\[ T_d = ( t_d / d_d = ( e_from - e_to / t_min, t_max ) \text{ where } e_from \text{ occurs before } e_to \text{ in time}) \]

where \( t_{\text{min}}(\text{max}) \) is the minimum(maximum)- timing constraint between the two events, and \( e_{\text{from}} \) and \( e_{\text{to}} \) are events \( E \), which is the set of all events. \( E \) has a number of attributes: signal name, direction, value, and edge number (an identifier).

Often, the timing dependency between two adjacent clock edges is omitted from the timing diagram. While not shown explicitly, timing dependencies between clock edges do exist. Thus, HIDE appends timing dependencies between every pair of adjacent clock edges, and propagates timing values[1]. For a timing diagram with a clock edges, we have the set

\[ T_{\text{clk}} = \{ t_{\text{clk}} / t_{\text{clk}} = ( e_{\text{from}} - e_{\text{to}}, t_{\text{clk_min}}, t_{\text{clk_max}} ) \text{ where } e_{\text{from}} \text{ occurs before } e_{\text{to}} \text{ in time} \}

where \( t_{\text{clk_min}}(\text{max}) \) is the minimum(maximum) high or low period of the clock, and

\[ e_{\text{from}} = (\text{clock, value}, i), e_{\text{to}} = (\text{clock, value}, j) \text{ for } i = 1, \ldots, n-1, j = i + 1. \]

The EG of the timing diagram is \( G = (V, E) \), where \( V = T_d U T_{\text{clk}} \). Figure 6 shows a portion of the timing diagram, T21, and its EG representation.

![Fig. 6. EG for T21.](image)

4.3 EG Transversal and Code Generation

HIDE builds the EG while reading the input file. The VHDL code for each state is then generated by traversing the EG with the code generation (CG) algorithm sketched in Figure 7.

For each clock event, a wait statement is generated. Then, for each input event that has a EG edge leading directly or indirectly (a path) to a clock event, a timing constraint is propagated (denoted by procedure Propagate [1] in CG) from that event to the clock event, and a statement is generated to check the setup time implied by the propagated timing constraint. Similarly, for each event that has an EG edge leading from the clock event to it, the timing constraint is propagated. For an input signal, a statement is generated to check hold times. For an output signal, a VHDL signal assignment statement is generated with the proper delayed timing.

Let \( d \) be the average distance of a node from a clock node (e.g., \( \text{ADS} \) in Fig. 6) has \( d = 1 \), and \( n \) be the total number of nodes. The average case complexity of CG is \( O(n d) \); best case is \( O(n) \); and, worst case is \( O(n^2) \) [10].

5. Experimental Results

HIDE generated a BIM for the 80386. The model was used to simulate a design containing the 80386 and
memory chips in a CAD system. The simulation corresponded exactly to the timing specifications found in the data book. The input to HIDE took less than one man-week to generate. HIDE ran in about two minutes for each bus cycle on an Apollo DN3500 workstation. We estimate a similar model could have taken three to four man-months to write and debug by hand.

\[
c = \text{number of clock events;}
\]
\[
\text{for } y = 1 \text{ to } c \text{ do}
\]
\[
e_{\text{clk}} \in E \{ e_{\text{clk}} \mid \text{signal} = \text{clock} \} \quad \text{and} \quad (e_{\text{clk}} \cdot \text{edgevalue} = y)
\]
\[
\text{generate wait statement for } e_{\text{clk}}
\]
\[
\text{forall } v_j \in V \mid 3 \{ v_j = (e_{\text{clk}}, v_{\text{ijkl}}, \text{min}_i, \text{max}_i) \} \quad \text{and} \quad \text{and} \quad v_k = (e_{\text{clk}}, v_{\text{ijkl}}, \text{min}_k, \text{max}_k) \quad \text{and} \quad \text{and} \quad v_j = (\text{eq}, v_{\text{ijkl}}, \text{min}_q, \text{max}_q)
\]
\[
\text{Propagate}(e_{\text{clk}}, e_{\text{eq}})
\]
\[
\text{if } e_{\text{dir}} = \text{IN}
\]
\[
\text{generate min, typical or max setup time violation check for } e_{\text{ij}} \quad \text{signal}
\]
\[
\text{else}
\]
\[
\text{report illegal timing constraint specification}
\]
\[
\text{end for}
\]
\[
\text{forall } v_p \in V \mid 3 \{ v_p = (e_{\text{clk}}, e_{\text{ijkl}}, \text{min}_j, \text{max}_j) \} \quad \text{and} \quad \text{and} \quad v_k = (e_{\text{clk}}, v_{\text{ijkl}}, \text{min}_k, \text{max}_k) \quad \text{and} \quad \text{and} \quad v_p = (e_{\text{eq}}, v_{\text{ijkl}}, \text{min}_p, \text{max}_p)
\]
\[
\text{Propagate}(e_{\text{clk}}, e_{\text{eq}})
\]
\[
\text{if } e_{\text{dir}} = \text{IN}
\]
\[
\text{generate min, typical or max hold time violation check for } e_{\text{ij}} \quad \text{signal}
\]
\[
\text{else}
\]
\[
\text{generate output statement for signal e_{ij} with min, typical or max timing delay}
\]
\[
\text{end for}
\]

6. Discussion

Historically, simulation models have been written by hand. Due to the complexity of both components and HDLs, this is often not time effective. A number of approaches have been taken recently to improve the model writer's productivity. In particular, a number of tools produce VHDL code from a variety of input formats. One of the most extensive systems is BIF [6], which is typical of the table-based approach. The behavior of a component, both interface and internal, is entered via sets of state tables, which are converted to efficient VHDL code. The HUM system [12] supports a similar paradigm to BIF, except that the input format is a spreadsheet.

There are a number of important differences between HIDE and table-based approaches. HIDE is dedicated to producing BIMs, not general behavior models, as are the table-based systems. Thus, HIDE's user interface, XWave, is highly efficient when compared to general-purpose-input techniques. XWave does not require specialized training. Thus, the model generation process is much simpler than with general purpose modeling tools. Furthermore, since HIDE only produces BIMs, it takes advantage of language constructs to improve model-simulation performance.

The advantages of using a BIM-style model are discussed by Armstrong [2] and Jennings [9]. Jennings does not, however, provide tools to build these models.

7. Conclusion

The HIDE system provides for the automatic creation of synchronous VHDL bus-interface models without requiring detailed knowledge of a component's behavior. BIMs are an efficient modeling paradigm for hardware verification, and fit naturally into CAE environments. HIDE generates all set-up and hold checks between relative signals, and checks other intra-signal timing. HIDE is efficient (O(n²) worst case) and has generated a VHDL model for Intel's IAPX8686.

HIDE is currently being extended to generate asynchronous devices, and for devices where internal registers affect bus signals. While HIDE currently generates VHDL, it can be easily modified to produce models in a variety of HDLs.

Acknowledgments

We would like to thank Dave Kieras for help with XWave, Ajay Daga, and an anonymous reviewer.

References