A VHDL-Based Bus Model for Multi-PCB System Design

Jari Toivanen**, Jari Honkola*, Jari Nurmi*, Jyrki Tuominen*

** Nokia Mobile Phones
Cellular Data
P.O. Box 68, FIN-33721 Tampere, Finland
Email: jt@nmptre.nmp.nokia.com

* Tampere University of Technology
Signal Processing Laboratory
P.O. Box 553, FIN-33101 Tampere, Finland
Email: {honkola, nurmi, tuominen}@cs.tut.fi

Abstract

In the development of bus-based systems and individual PCB boards interfacing to a bus, the simulation usually requires a specific test bench or creation of quite complex stimuli. This problem can be avoided with a VHDL model of the bus. In this paper, the bus model concept is discussed. The concept can be applied to serial and parallel, single and multiple master bus modeling on various hierarchical levels. The model includes timing and signaling analysis, master, slave and arbitration modules. ISA (PC/AT) bus is used as an example case of the modeling.

1. Introduction

Behavioral simulation plays an important role in the design verification of digital systems today. Design verification is usually partitioned to PCB boards or even smaller blocks. These PCB boards or blocks will be in many cases connected together and to external devices, e.g. sensors and actuators, with different kinds of buses. When we want to verify the bus interface of a bus unit, we are usually forced to build a test bench or to create quite a complex stimulus file. We have also problems if we want to verify a whole bus-based system by simulation because in many cases the bus functionality can not be achieved with plain signal lines between the bus units.

The bus model concept described in this paper is developed to solve the previously mentioned problems. It can be used as a test bench to generate bus events and to verify bus signaling for individual bus units as well as for a complete bus-based system. With the bus model, the bus cycles that we want to execute during the simulation are written as simple commands in a file. Bus traffic and bus protocol violations are displayed with textual messages.

2. The bus model concept

A bus model may include modules depicted in Figure 1. All buses do not necessarily need all modules and some may have also additional modules. On the left side of the dashed line the modules are described by the user and the modules on the right are part of the bus model.

The timing and signaling analyzer module observes the traffic on the bus and reports upon it. This module also recognizes the timing errors and reports protocol violations.

The master and slave modules facilitate testing the designs that are connected to the bus model. When developing any unit, its function can be tested by using these modules. The bus cycles wanted to be performed are given to the master module in a command file. Also the slave modules have a configuration file in which it is determined to which bus cycles the slave module responds to.

When multiprocessor buses are simulated, it may be necessary to use a separate arbitration module. Then it is possible to simulate the whole system without arbitration module implementation.

The design of a unit which is connected to the bus model can be done as a normal development process where the simulators are used in the verification of the design. The unit can be constructed by using components of commercial simulation model libraries [1] or application specific VHDL descriptions [2].

The verification of the unit under design can be started as soon as the first unit is ready for simulation, without building a unit specific test bench. So the bus model can be used from the very beginning to the verification of the complete system. The bus model consists of many modules which can be used in the simulation when necessary. When starting the debugging of the first unit, it is advisable to use the timing and signaling analyzer module and modules which generate or respond to events on the bus.

The timing and signaling analyzer module may need some parameters which are given, for example, in the schematics. Such parameters could be, e.g., the level of reporting and the speed of the traffic.

The modules that are used to generate and respond to the bus events are separated in this paper, but this is not possible for all buses. The master module is needed if a slave

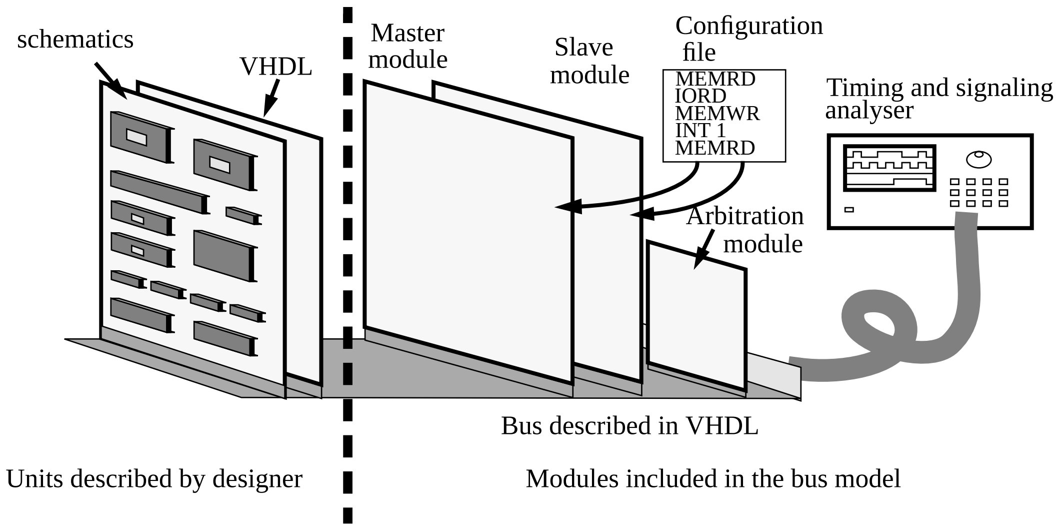

Fig. 1. The structure of the bus model.

module is wanted to be tested, because in most buses the traffic is initiated by the master module. The master module uses at least one parameter which determines the name of the command file. This file includes the commands that are interpreted to bus cycles during the simulation. The commands in the file are bus type dependent. These would be, for example, write and read cycles if the bus type is parallel, or sending messages on a serial bus. In the bus types which support interrupts, the names of the command files executed when interrupts are requested can also be given in a command file. This command file includes similar commands as the command file of the master module. The slave modules have their configuration files which are connected to the module in the same way as the command file is connected to the master module. The information in the configuration file depends on the bus type used, but there must be at least the address of the module and the initial value of the registers and the memory.

The reports of the modules can be directed either to the command window of the simulator or to a file, where they can be examined. Of course, the results of the simulation can be examined also in the waveform display of the simulator. The most important part of the bus model is the module which recognizes the errors on the bus. From the report of this module it is easy to verify whether the design works correctly or not.

When developing a complete bus-based system, the modules of the bus model can be left out when the corresponding physical units are completed. When the whole system is completed, all the bus model modules can be left out from the simulation, making the simulation faster.

3. Implementation of the bus model

With bus models we mainly do behavioral simulations and verification of the system specification. We can also do bus performance analysis and verify individual bus units.

In verifying designed bus units, it is important that they can be connected to the bus model without any or with minor changes. This way we can ensure that we are verifying what we have really designed. Unfortunately this is not always possible because of a number of limitations. We have to do changes, e.g., if the bus data communication has been implemented by modulation or the signal levels are different from digital circuit signal levels. Figure 2. depicts a bus model, which can vary from a simple model which connects units by a set of signals to a complex behavioral description.

One of the biggest problems with bus modeling is the lack of bus control circuit models. There are two solutions for this problem. The first one is a self-made control circuit simulation model. This choice can be very time consuming and very difficult to implement. On the other hand, by doing an accurate, reusable (e.g. synthesizable VHDL) model, we can use this description effectively in future designs, for example in integrating bus control and bus module functions on the same chip.

Another possibility to avoid missing simulation models is to raise the abstraction level of the bus model and connect it to that interface, where the bus controller is normally connected to. Other reasons for the higher abstraction level can be the requirements of shorter simulation times or system simulation on the message level.

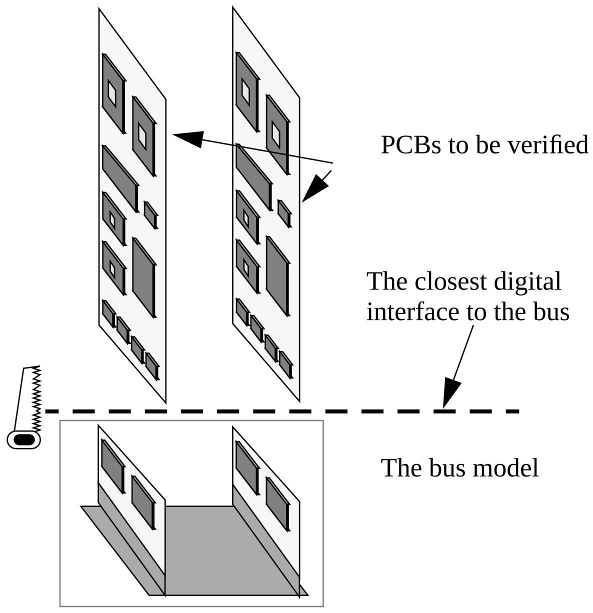

Fig. 2. The interface of the bus model to bus units.

3.1. Signaling and bus arbitration

In most design and simulation cases we use commercial simulation model libraries. When we are building a bus model, it must be compatible with these libraries. In practice all desired technologies and signal values can be implemented with 9- or 12-level signals.

When more than one bus unit can asynchronously initiate bus operations, we have to consider bus arbitration and its implementation in the bus model. In traditional multiprocessor buses the arbitration logic can be either distributed or centralized. In the case of serial buses the arbitration methods are based on contention arbitration and token passing (e.g. bitwise arbitration).

In centralized arbitration logic cases the bus model includes a behavioral description of the arbitration logic which can be used in simulations. In distributed logic cases every bus unit has its own bus arbitration request logic [3]. Generally, the implementation of the bus arbitration is quite straightforward but bus type dependent.

3.2. Bus modules

The bus model must support the simulation and verification process from a single bus unit design to a bus-based system design. For this reason the bus model has to include modules, which can generate bus events for units under verification and reply to bus events activated by the units. In parallel buses and single master serial buses, the modules can be generally divided into two classes: master modules which are able to start the transfer events (read/write, interrupt services etc.) on the bus, and slave modules which reply to requests of the master and possibly request interrupt type of services (e.g. DMA).

Schematics has been used to connect the bus modules to the bus model, because module connection and removing is faster graphically. This model can be used as the system test bench by configuring the modules. There are two ways to do the configurations. The configuration files can be VHDL descriptions connected to the simulation environment like any behavioral VHDL description. This is not flexible and convenient, because the configuration files are modified frequently and VHDL description recompilation and simulator restart are needed. A much better way is to write the configuration data in a text file, which is read every time the simulation starts and, if needed, during the simulation. This makes it possible to modify the configuration during the verification process (interactive simulation). The command format in the text files is bus type specific.

The configuration text file of the master module contains an introduction part for simulation setup process and a command part where events which will be executed have been written sequentially. A new command (event) is read and interpreted from the configuration file when the previous one has been executed. In the introduction part there can be information of master priorities, serial data communication rate, response time etc. Typical operations in the command part are, e.g., read, write, and block transfer. In the buses where interrupts are possible, there must be separate command parts for interrupt services (analogous to interrupt service programs). They can be separate files or included in a shared configuration file (Figure 3.).

The configuration text file of the slave module is read in

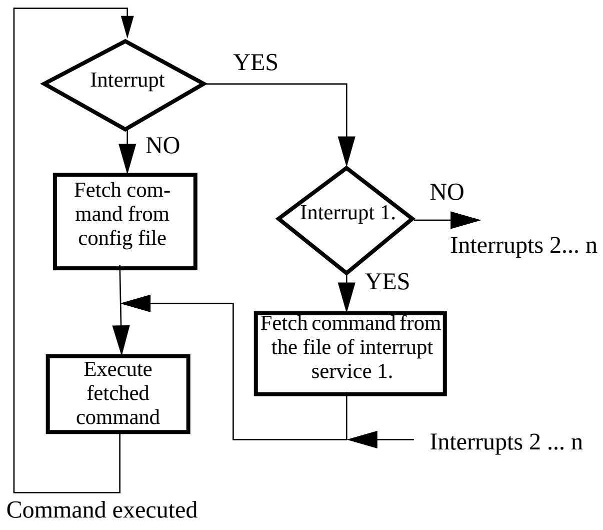

Fig. 3. Execution of commands in the master module.

the setup process of the simulation. It contains parameter settings and memory or register content initialization for the module. The size of the memory or register space is good to be defined as a parameter, making the use of the module much more flexible. The base address of the slave module is the most obvious parameter setting. Other parameters for the slave module can be the word length, operation response time, interrupt invocation data (interrupt number, invocation moment, R/W acknowledgment address) etc. In the setup process there must be commands for setting the defined data into the memory of the module. Undefined memory location contents are set to a specified value, typically 00H or FFH.

3.3. Reporting bus events

When the designer is simulating a system with the bus model he must know, e.g., bus events, operation addresses and data contents within a certain time interval. The bus model includes a reporting module, which is a separate component in the model and reports of the different bus events into a comment window of the simulator or into a file. The reporting module also prints the start time of a certain command or event.

In parallel bus models, the interpretation of the bus events can be divided into three phases. In the first phase the unit initiating the event, the address of the unit and the type of the bus cycle are clarified. In the second phase the unit can affect the length of a bus cycle by delaying or by choosing one alternative of different reply modes. In the final phase the bus event terminates and, e.g., read data can be printed out. The reporting module must be divided into several processes like one process for each bus cycle type, interrupt and DMA request processes.

3.4. Timing and signaling errors

The role of the timing and signaling analyzer module is remarkable because timing and signaling errors are the most important verified items in the designed units. Analysis of the errors must be done on the basis of the signal values and timing. The printed error messages should be categorized into several classes by the error severity for easier interpretation.

In Figure 4. there is one example of signal timings on a parallel bus. For each time shown in the figure, minimum and/or maximum time values have been specified. The timing analysis must be divided into several processes, because in parallel buses there are several timings which do not have dependencies with each other and the order of the signal events cannot always be predicted. In most cases it is necessary to include the interpretation of the bus events as a part of the timing analysis, because the timings are dependent on the types of the bus cycles.

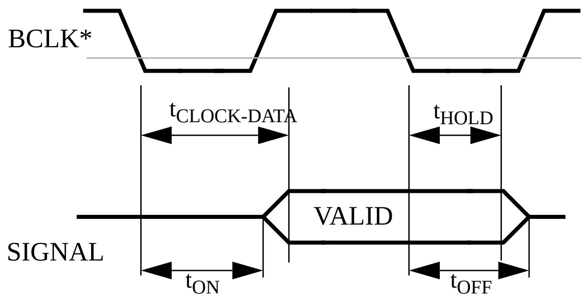

Fig. 4. An example of signal timings.

4. Modeling example (ISA)

A model of the ISA bus, known also as the PC/AT bus, was implemented as an example [4]. ISA is a typical example of an asynchronous parallel bus. The ISA bus model was implemented according to the ISA Bus Specification and Application Notes released by Intel Corporation [5].

Table 1: Driver types and signal states.

Driver type	logical 1	logical 0	unknown	tristate
TTL	1	0	X	not specified
open collector	H	0	X	not specified
Tristate	1	0	X	Z

4.1. Bus signal implementation

The ISA bus model was implemented on low hierarchical level to support ISA bus add-on card design, especially when the bus interface logic is integrated into an ASIC. In this implementation the behavioral model of the bus was not actually needed. All what is needed is behavioral models of bus modules and a bus interaction analyzer that can be connected together with signal lines in a structural VHDL description or with wires in a schematic. Std_logic_1164 multi-value logic system can be used to implement TTL, tristate, and open collector drivers and receivers (Table 1.)[6]. Signal type std_logic is utilized because resolution function is built in that signal type. Standard component simulation models from the most widely used commercial simulation model libraries can be directly connected to the bus model signal lines.

4.2. Bus modules

To make the bus model support add-on card design verification as well as bus-based system simulation both master and slave modules have to be implemented. The names

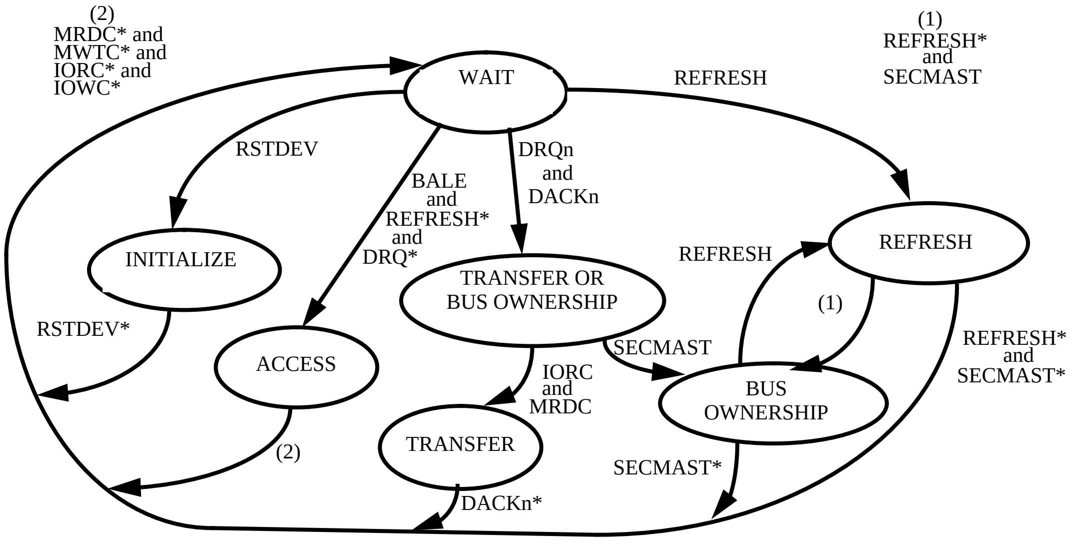

Fig. 5. Main state machine.

master and *slave* are not exact but the risk of misunderstanding is minimal. In an usual ISA bus configuration, platform resources act as a master and add-on cards as slaves. Text files are used to configure the master and the slave modules.

The master module is able to execute memory and I/O access cycles. Access cycles are generated from interpreted configuration file commands in a sequential order during simulation. If interrupts are not requested, access cycles are executed from the main configuration file. In the case of interrupt request, all the access cycles from the interrupt service file are executed once. The following line is an example of the master module command format.

```
MRD(“ADDR”, "SBHE","LADDR");
```

With this command a memory read access cycle can be executed. The following example is from an actual master command file.

```
-- This is comment
MRD("00000","0","00");--Memory read
MWR("00000","0","00","AA","BB");--Memory write
IOR("08000","1"); -- I/O read
IOW("08000","1","BB"); -- I/O write
-- The end
```

The master module also generates the 8 MHz system clock for the bus, enables MEMR and MEMW when needed, and enables Reset Device signal of the bus in the beginning of the simulation.

The configurable slave module that can act as an add-on card is also included in the bus model. Multiple slave modules can be included in the simulation. Every module must have its own configuration file. The slave module is able to respond to the memory and I/O access cycles generated by the master module. One interrupt activation is also possible for one slave. The current slave module has 128 16 bit readable and writable locations that can be preset. The slave module configurations are read in the beginning of the simulation. Here is an example of a slave module configuration file.

```
-- This is comment
BASE(000826","00");--Set base addr of add-oncard
MODE("0");--Add-on card type selection mem or I/O
BITN("0");--Add-on card bit count selection 8/16
ACTY("1");--Access type selection.
SETB("00", "FF");--Byte value presetting
SETW("10", "01", "01");--Word value presetting
INTR("3 ","6000 ns ","02")--Interrupt generation
-- The end
```

4.3. Bus cycle reporting

There are four distinct bus cycles on ISA bus: access, transfer, refresh, and bus ownership. It is reasonable to include the bus initialization to the same state machine as the previously mentioned cycles. In Figure 5. there is the main state machine that is used to select the cycle type on a bus transaction. A transition in the diagram occurs when a named signal is enabled or a signal with * character is disabled. The same main state machine is used in bus cycle reporting and in timing and signal analyzing. Most states in the main state machine must be divided in substates during reporting or analyzing process.

The reporting module of the bus model displays information about the current bus activities during simulation. TEXTIO standard package is used to display information. In some simulators information can be displayed on the screen but in all simulators information can be saved in a file. An example of reporting module displayed information is below. BMI stands for Bus Model Information.

```
BMI:Executing 16-bit memory read cycle at time
BMI:1764.000000ns
BMI:ADDR: 00000 SBHE: 0 LADDR: 00
BMI:Cycle type 0-wait
BMI:High data byte: AB Low data byte: CD
BMI:Cycle completed
```

4.4. Timing and signal analyzer

This is the most important module of the bus model from the user‘s point of view. This module can be called the analyzer module. It displays information about timing and protocol errors during the simulation. This module appeared to be the most time consuming in bus model implementation. The main reason for that is the nearly unlimited number of different error combinations and a great number of signal timing variations. More than half of the VHDL description lines of the whole bus model are in this module. Some compromises had to be made. We gave up the intention to built an analyzer that can give a precise information message of all signal errors and concentrated to timing errors because they are more difficult to notice in visual check of the simulation results.

On ISA bus there are two typical signal timing specifications. First, the interactive action where a signal value change triggers a response signal on another bus module. Another typical situation is where two signals from the same unit have a specified time difference. In both cases maximum, minimum or both time constraints can be defined. The signal propagation delay between two units can be 0-11 ns on ISA bus. The signal propagation delay is not implemented in our bus model. However, it is not a problem because worst case can be used in all timing analysis. Following there is an example of the analyzer information messages. BMW stands for Bus Model Warning.

```
BMW:Unlatched addr stable state is not long enough
BMW: before read or write command falling edge.
BMW:Unlatched addr have to stay stable at least
for 109ns
BMW: before read or write command falling edge.
```

4.5. Portability and simulations

This simulation model was tested in Intergraph [7] and Mentor VHDL simulators. Portability of our model was not complete, and some changes were needed in signal line implementation and in TEXTIO package function usage. The problem with signal lines was that simulator dependent signal types had been used. Another portability problem was the use of a TEXTIO package function to test end of line type variable. With careful model design, portability between these simulators can be achieved in this kind of cases. Simulation times are highly dependent on the user described units that are connected to the bus model. Simulations that we have done with the bus model master, two slaves and reporter and analyzer are able to give some rough estimates of simulation times. In that configuration, one bus cycle took about one and a half second of elapsed time in simulation in Intergraph VHDL simulator. The computer hardware we used was a Sun SparcStation IPC with 24 Mbyte main memory.

5. Conclusions and future work

In this paper a VHDL-based bus model concept has been described. The modeling of the example case took three man months of which the signal and timing analyzer implementation took about two thirds of the effort. VHDL as a description language was suitable for this kind of modeling.

As it is shown in the modeling example, a bus model helps designers to verify effectively their bus units in the system under development. By configuring master and slave modules the designer can flexibly generate the desired stimuli to the bus unit under design. The reporting module and the signal analyzer of the model give valuable information of the bus communication and errors. If the selected bus is planned to be used for a long time, a lot of work in the creation of complex stimulus files can be avoided by designing a bus model for future product development.

The bus modeling work has been continued with two buses: the CAN bus and a telecom serial bus. The CAN bus model implementation will support the system design verification on the message level. The telecom serial bus model will be designed to work as a configurable ASIC test bench for large telecom system development.

6. Acknowledgments

This work has been a part of Finnish technology program ESV (Electronics design and manufacturing technologies) and is supported by the Technology Development Center (TEKES), ABB Corporate Research Finland, Kone Elevators, Mariachi Oy, Nokia Research Center and Nokia Cellular Systems.

7. References

[1] SmartModel Library User´s Guide, Logic Modeling Corp., 1992. 122 pages.
[2] Douglas L. Perry. VHDL. McGraw-Hill Inc., 1991. 458 pages.
[3] Joseph Di Giacomo. Digital Bus Handbook. New York: Mc Graw-Hill Publishing Company, 1990. 560 pages.
[4] Jari Toivanen. VHDL-Based Modeling of Buses (in Finnish). MSc Thesis. Tampere University of Tech., 1993. 77 pages.
[5] ISA Bus Specification and Application Notes. Intel Corporation OCPD Technical Marketing, September 12, 1989. 73 pages.
[6] Jean-Michel Berge, Alain Fonkoua, Serge Maginot, Jacques Rouillard, VHDL Designer´s Reference, Kluwer Academic Publishers, 1992. 455 pages.
[7] AdvanSIM-1076: Digital Lab User Guide Version 7.0. Intergraph Corporation, July 1992.