A Mixed-Level Self-Verifying VHDL Simulation Environment with Selective Random Control of Data Transactions

Michael C Cogswell, Calvin C Paynton
IBM Corporation
3605 Highway 52 North
Rochester, Minnesota 55901

Abstract - The need for flexible, mixed-level simulation environments continues to persist due to increasingly high-density chip technologies coupled with ever decreasing product cycle times. The simulation development team at IBM-Rochester was challenged to create a portable mixed-level simulation environment to assure correct functionality of the I/O bus and memory subsystems for the next generation of PowerPC® systems. Since this new environment was to be used to verify a wide range of memory and I/O bus architectures, it had to be both robust and adaptable. The chip development teams also required that the tests case results be self-verifying, requiring little or no specific "expected value" details from the user. To further enhance the effectiveness of each simulation environment, both random test conditions and bad-machine-path functions were added to the user's test case command language. The resulting simulation environment is based on a network model and makes heavy use of mixed-level simulation techniques. The network manager as well as the high-level I/O bus and memory models are written in sequential VHDL.

I Introduction

The chip simulation team at the IBM Rochester Development Lab is responsible for the development of simulation models and environments for use in the verification of IBM products. This paper presents a description of the mixed-level simulation environment used in the verification of a wide variety of memory and I/O bus subsystem designs.

i Mixed Level Simulation Environment

The chip simulation team was challenged to create a flexible, high-level simulation environment to be used in the verification of the Register Transfer Level (RTL) descriptions for a series of new products. This environment is comprised of a number of models which must provide accurate interfaces and behaviors of the wide variety of Memory and I/O Bus devices which might be used in conjunction with the new designs. The models must also incorporate the ability for run-time checking of results, allow for random timing & data and provide functions for the injection of errors on the model<>RTL logic interface. It was therefore decided that a mixed-level simulation environment would provide the most flexible and timely solution. The RTL designs are essentially concurrent, structural descriptions while the models are behavioral descriptions. Each of these two descriptions require a different language or at the very least different language semantics [1].

II Implementation Language

The RTL designs were being written in concurrent VHDL so the next step was to select a behavioral language for the high-level models and several IBM-specific and industry standard languages fit the requirements needed for functional simulation [2]. Since most VHDL simulators such as MTI VISIM® and Cadence Leapfrog® support foreign language interfaces, languages such as C were considered. However, these models needed to be both portable and reusable and there is no industry standard or consensus on the form and content of the foreign language interface. Therefore we chose to create all models in IEEE 1076-1993 compliant VHDL [3] which fully defines the available VHDL constructs for sequential (behavioral) modeling. Using sequential VHDL also allows us to seamlessly change to different simulation frameworks if needed. Furthermore, since many of these models emulate the behavior of common industry devices such as JEDEC®-standard DRAMS and common industry I/O busses, writing these models in standard sequential VHDL allows them to be reused in other designs. As will be shown later in this paper, two levels of behavioral descriptions are used to promote maximum reuse of these models - the
Environment Behavior (EB) level and the Physical Interface Behavior (PIB) level. This environment is running on IBM RISC System/6000® workstations and could be ported to other machines if needed since nearly all of it is written in IEEE-1076 VHDL[4].

There are 4 types of objects in the simulation environment: concurrent RTL logic, SymNet, Environment Behaviors and Physical Interface Behaviors.

Concurrent RTL Logic
This is the logic being verified.

SymNet
The SymNet object manages packet lists for all of the EBs and PIBs. It also handles functions such as packet retrieval and packet compares. It is the heart of the simulation environment (see figure 2).

Environment Behaviors (EB)
The EB parses the testcase commands and builds PIB-specific packets which predict future data transaction results. These packets are sent to SymNet which adds the packets to the PIB-specific packet list. These lists are divided up into sessions as shown in figure 2. Packets sent to the Bus PIBs drive the RTL logic which in turn may drive other PIBs (e.g. Memory PIB). The EB has no direct connections to either the PIBs or the RTL logic. All communications are funneled through SymNet which acts as the networks message dispatcher.

Physical Interface Behaviors (PIB)
The PIB interface directly to the concurrent RTL logic and receives commands over this interface. Upon receiving a command from the logic, the PIB builds an internal version of a packet based on the actual address, data and control inputs present on the interface. This internal packet is sent to SYMNET and compared with the “expect” packet pre-constructed by the EB to see if they match. If the packets match (via the check_message function shown in figure 2) then the hardware is working correctly. If the packets miscompare, then an error has occurred in either the RTL logic or the bus interface.

Figure 1 shows an example simulation environment for a memory subsystem as it might be used in an N-way PowerPC system. The Physical Interface Behaviors (PIB)
such as the PowerPC Bus PIB (PPC BUS PIB) are designed to be reused in other subsystems.

II SymNet - The Sim Network Manager

As can be seen in Figure 1, the PIBs are connected directly to the RTL logic being tested. Commands and data are passed between the PIBS in the form of network packets. Two other parts of the simulation environment operate behind the scenes: the EBs and SymNet. The Environment Behaviors (EB) create these packets based on testcase commands. SymNet, as depicted in the block diagram of Figure 2, is common to each simulation environment and has the following characteristics:

1. Contains a suite of API functions used by the PIBs & EBs in creating the network packets and passing them to each other. This powerful set of functions allow for the standardization of the PIBs & EBs, aiding in their reusability & ease of maintenance.

2. Acts as the global command-and-control for all message packet transactions. The packets can be triggered off each other to assist in the verification of ordering rules, be delayed by random timings to emulate the operation of a real system, or simply become active immediately.

3. Supplies a set of testcase commands allowing the hardware designer maximum flexibility in determining how the testcase is to execute. The designer can do such things as: a) Wait until the system quieses before continuing, b) Wait a random or fixed amount of time before continuing, c) Wait for a session, described later, to quies, or c) Wait upon some I/O (anywhere in the design) to change.

4. Incorporates an interface to the simulator being used allowing the designer to issue simulator commands directly from the testcase. This is extremely useful in Bad Machine Path (BMP) testing as the designer has a very easy & efficient way of injecting errors in relation to other events in the testcase.

5. Provides programmable clock signals to be used by both the models and the RTL hardware design.

SymNet uses a message passing network model to manage packet transactions. To simplify network control, the PIBs are grouped into sessions. Functions such as put_message (which sends a packet to another PIB) use the session name to communicate with other PIBs. SymNet acts as the go-between since the PIBs themselves have no knowledge of the existence of other PIBs. This architecture makes it very easy to alter the
type and number of PIBs in the environment. For the example in Figure 1, since the Memory PIBs do not directly communicate to the PowerPC Bus PIBs, more Memory PIBs (or more PPC BUS PIBs) could be added without changing the PIBs themselves.

SymNet also uses a “plexus ring” to link all models together in the message passing environment, allowing for message transfers to occur in VHDL simulation delta time. This is extremely important for an environment which supports self verification of results.

III High Level Models

There are two levels of models provided: Environment Behaviors and Physical Interface Behaviors (PIB).

i Environment Behaviors

Environment Behaviors (EB) are responsible for the overall flow of commands and data used in the simulation. EBs have these responsibilities:

1. Testcase Parsing and Error Checking. Each RTL subsystem has its own testcase command definition to acutely tailor the testcase structure and content to the subsystem. This was much preferred (particularly from our customer’s perspective) over a single testcase definition language for all of the different designs simulated in environment.

2. Identify all of the PIBs present in the particular concurrent RTL environment. The PIBs themselves are unaware of what other PIBs are in the environment.

3. Construct all required message packets that are needed for the simulation. These packets are either created directly from testcase commands or are inferred from specific testcase events. These packets include generated data as well as expected results. All packets are placed on sorted packet lists maintained by SymNet.

4. Factor in the degree of randomness used in the simulation. The user can leverage how much to control the simulation directly in the testcase with SymNet/EB generated random events. In practice, the amount of randomness allowed increases as the RTL design matures.

5. Generate/monitor specific VHDL signals from the RTL design which can dynamically alter the testcase flow.

ii Physical Interface Behaviors

The Physical Interface Behaviors (PIBs) model the functional behavior of specific devices or interfaces. They are written in such a way as to be independent of both the controlling EB and the concurrent RTL design and therefore can be “plugged-in” to other RTL design configurations since the interface between the PIBs and RTL logic is kept as general as possible. Some of the PIBs also support a simple testcase definition language providing more direct control of PIB functions including the injection of interface errors and the control of random parameters used to stress the RTL design.

The advantage of this two-level approach is three-fold:

1. The EB models can be adapted to specific concurrent RTL design requirements without effecting any PIBs.

2. The PIB models need only be defined down to the level of detail required by the interface of the concurrent RTL design. For example, the memory PIB cited in Figure 1 could be a model of a generic JEDEC-compliant DRAM device. It would therefore need to decode JEDEC commands, detect timing errors on input signals and generate the appropriate response packets for the EB. It need not actually model the geometry of any particular DRAM device since it doesn’t model the physical DRAM storage medium. All data is encapsulated within the packets.

3. The underlying architecture of both the PIBs and the RTL design can change without modifying the EB models. The EBs handle the testcases and packets and do not directly interface with the concurrent RTL model or the PIBs. This is one of the great advantages of the network model used throughout the simulation environment.

IV Testcase Definition Language

i General Command Syntax

Each concurrent RTL design is supported by a testcase definition language tailored to its intended use. The testcase definition languages generally provide support for these functions:

1. Designation of data transactions pertinent to the RTL design. For example, indicating that certain bus transactions with the given commands and data are to be carried out on the RTL design.
2. Designation of design-related operating modes are now no longer in effect.

3. Control of randomly generated test conditions. These criteria are highly user-selectable and allow random generation of data streams, bank selection, transaction modes, bus retries, etc.

4. Error injection criteria. Most of the EB models support a “Bad Machine Path” whereby command and/or data errors are intentionally injected into the specified data stream providing an easy and implementation-independent way to test error-checking logic in the concurrent RTL design.

ii Model-Specific Testcase Definitions

Figure 3 shows the syntax diagram of the testcase definition language for the concurrent RTL design of a bus device. There are parameters, keywords and immediate data (e.g. the BUID option requires a 9-bit Bus Unit ID binary value as indicated by the italicized “B9”). Omitting certain keywords will automatically trigger random generation of the required values. These random values are associated with a generated random seed so that the same testcase conditions can be repeated by using the same testcase and random seed.

The above figure also contains “Bad-Machine-Path” functions through the use of the APAR (Address Parity) and DPAR (Data Parity) keywords. The D1 field (1 decimal digit) indicates which word of the address (or data) is to be preconditioned to produce bad parity.

iii Self-Verification Of Results

Since each EB knows which PIBs are affected by each testcase command, they automatically build message packets that “follow” the data streams throughout the data flow of the system. For example, a memory READ command causes the generation of at least two packets:

1. A Memory PIB “read” packet indicating that a memory read command is to be expected over the Memory command bus and the data received should match the data built into the packet. When the actual read command is received over the command bus of the RTL logic (which is directly interfaced to the Memory PIB), the Memory PIB queries SymNet if there is a “read” packet matching the specifications of the current PIB command. If so, the data packet is sent over to the requesting PIB which then presents the data values onto the PowerPC Bus. If there is no such packet, the simulation is interrupted and an error message is issued.

2. There is also a PowerPC bus packet generated which predicts the upcoming Bus transactions. So in the case of a read command, the BUS PIB would expect to see the specific address/data pair as part of a read transaction. If the Memory PIB or RTL logic corrupted this data, the BUS PIB would issue a bus error. Likewise on a Memory Write command, if the RTL logic corrupted the data sent to the memory PIB, there would be no corresponding “write” packet on SymNet’s packet list and an error would be indicated. All of these errors are flagged as “exceptions” and the user is given the ability to ignore such events and continue, expect the event and continue,
or simply allow the testcase to end upon detection of the error. The EB automatically creates both of these packets from a single testcase command.

V Conclusion

The simulation environment described in this paper evolved over a 3-year span. We have continued to update and refine our methodology to keep at least a half-step ahead of our chip design customers. Throughout this work we have developed an even deeper appreciation for the breadth and flexibility of the VHDL standard and foresee increased use of VHDL in future environments.

VI Acknowledgments

We would like to thank the IBM Rochester Chip simulation team under the direction of Mike Andraschko. We also want to acknowledge the contributions of the RTL design team under the direction of Bill Cochran whose challenging requirements kept us honest and true.

The authors can be reached at:
mcogswell@vnet.ibm.com
calvimp@vnet.ibm.com

VII References


* PowerPC is a trademark of International Business Machines Corporation.

* JEDEC (Joint Electron Device Engineering Council) is a subdivision of the Electronics Industry Council (EIA).

* IBM and RISC System / 6000 are registered trademarks of International Business Machines Corporation.

* Leapfrog is a registered trademark of Cadence Design Systems, Inc.

* MTI is a trademark of Model Technology, Inc.