An Overview on Writing a VHDL Testbench

Dan Biederman
Member of Technical Staff
Hughes Missile Systems Company
Bldg 805, M/S F6
P. O. Box 11337
Tucson, AZ 85734

Abstract

This paper is an overview of VHDL testbenches and other related topics. It has been written for a digital design engineer with little VHDL or programming experience to get a better understanding of writing VHDL, and using a testbench. The future of VHDL is also discussed.

Introduction

Finding errors in the design of electrical circuits before the fabrication or production stage can reduce the product development time and cost. One way that engineers have to do this is by building a device around the prototype circuit called a hardware testbench. A hardware testbench would generate the inputs (Stimulus) and review the outputs (Monitor) of the device.

In recent years the design of digital circuitry, such as Application Specific Integrated Circuits (ASICs) and Programmable Logic Devices (PLDs), has been done using a software programming language known as a Hardware Description Language (HDL) such as VHDL. One of the reasons for this is that VHDL can ideally be synthesized into any current gate-level technology such as an ASIC or PLD. Thus an ASIC designed today can be redesigned several years later reusing the software code that was written originally.

Once a digital device has been designed, it needs to be tested. One method of testing these devices and their surrounding circuitry is to use a HDL testbench before the fabrication of the device. This paper discusses the methodologies of HDL testbenches and reviews their uses in the design process at the gate level, board level, and system levels to verify the proper operation of a digital design.

Writing a VHDL testbench

A testbench has three major parts, the stimulus, the monitor, and the device under test (DUT) as shown in Figure 1a. The stimulus is the part of the testbench that controls the input signal values. This portion of the software program should contain all of the information that the DUT needs to perform its job properly. This may include address and data buses, interrupt signals, enables, etc.

The monitor, on the other hand, checks or verifies the outputs of the DUT. This is usually done by waiting for the simulation event, and then checking that the event occurred with an "Assert" statement. By verifying the DUT using "Assert" statements, one can usually save simulation time, because all signal traces need not be stored, and also save engineering time, because all of the signals need not be verified individually.

In using VHDL, one can have the monitor and the stimulus in the same program (entity) as shown in Figure 1b. If a digital designer wishes to use only one program, and still keep the stimulus separate from the monitor, it can be achieved by using different processes for the monitor and different process for the stimulus.

0-8186-7873-9/97 $10.00

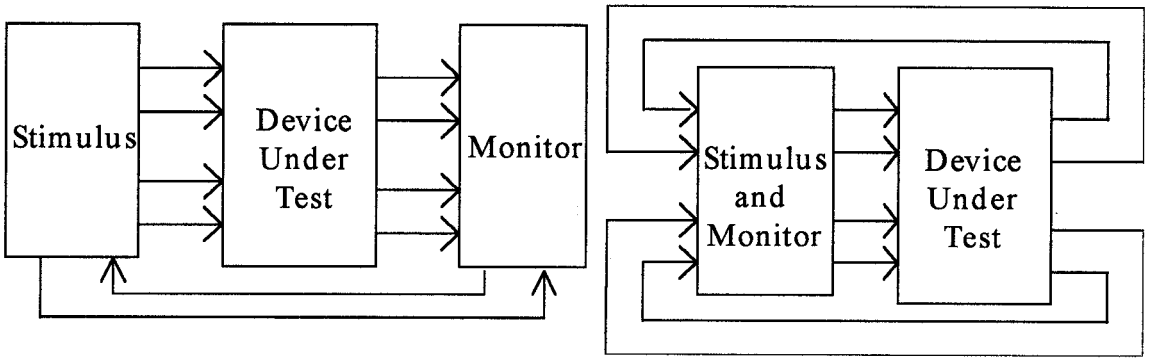

Figure 1: (a) Testbench configuration with separate Stimulus and Monitor sections, (b) Testbench configuration with a single Stimulus and Monitor section.

Using a descriptive coding style

Before one designs a VHDL module, one should consider a descriptive coding style that will be easy to reuse in future designs. The easiest way to describe your code is to use comment statements or remarks. These comments will describe the operations of the code for others to understand. This may be adequate for simple modules, but for complex modules, other methods may be more desirable.

Using prefixes, suffixes, underscores and capital letters

In writing code, your engineering team may want to decide on a coding style which everyone will follow. Prefixes and suffixes can be very helpful in telling the reader exactly what an identifier is. Underscores or capital letters can be used to separate words in a signal name. Look at the example below:

```
Address_Enable_Out_Sig_Low <=
      not (Address_Enable_Sig)

If (ResetInSigLow = '0' or
      COUNTER_VAR = 15) then
```

From the examples above, one can easily determine from the signal name that the signal, Address_Enable_Out_Sig_Low, is an active low output signal used as an address enable. Likewise the signal name, ResetInSigLow, is an active low input signal that occurs during a system reset. Even though VHDL is not case sensitive, it is often useful to write VHDL code as if it were case sensitive. For example, if all variables are capitalized it is immediately apparent to the reader where variables are used.

Designing flexibility into the software testbench

In writing a testbench, the flexibility of the code should be considered. This is due to the fact that requirements inevitably change. For example, an ASIC may originally be targeted to work at 15 MHz device, and as the system evolves, the ASIC may be required to work at 30 MHz to meet final system performance objectives. If flexibility is not included in the testbench, it may require much time to update the stimulus and monitor to the new speed.

Flexibility is an important issue when considering future reuse of the testbench. As technology advances, the 30 MHz device maybe too slow and might need several new features to be added to keep up with the competition. In order to be more time and cost efficient the old testbench may be reused, in whole or in part by changing the variable to the faster requirements,

and adding new modules to test the additional new features.

Avoiding Hard-Coded values in testbenches

In writing code, hard-coding of timing values should be avoided. These statements easily lose their meaning over time, often even to the digital designer who wrote the statements as memories fade. Also, force statements, a series of linear staements of the form "X occurs at time Y (or in VHDL, X <= Z after Y ns;) are not very flexible. Thus if the code consisted of 2000 force statements, and the digital designer decides to change the timing, every line might need to be to changed. Instead, stimulus can be structured in a more behavioral form and code can be parameterized.

Mimicking the hardware or system operation

The VHDL programmer should try to mimic the actual hardware or system behavior when possible. This will make the code easier to comprehend for someone who is unfamiliar with it, but understands the operation of the design being implemented. Also, the code will be more flexible to changes should they be required.

Parameterizing instead of Hard-Coding

Below is an example of a line of code with hard values and two examples of parameterized code:

```
Wait for 100 ns;
Wait for CLK_2_OUTPUT_DELAY;
Wait for 2 * CLOCK_200MHZ_PERIOD;
```

The first line is hard-coded. Thus in every instance the code will wait for a specific time. This is passable for a simple testbench, but for more complex testbenches, this wait statement is too rigid. Also, if there exist many related wait statements, all of them might have to be changed if timing changes.

The last two lines of code are parameterized statements. Parameterized code uses defined constants or variables in place of hard-coded values. Then, with a simple change in a small section of code (the constants or variables) the rest of the stimulus or monitor code automatically "readjusts" as new timing is calculated. These variables or constants, which may be defined in a package, are descriptive to the purpose of the constant or variable. Also, since these variables are defined in one place, changing the values of these variables only involves changing one line of code.

Creating reuseable libraries.

One of the great advantages of VHDL is that it can be synthesized directly into different technologies. Thus it is time efficient to create a Reuse Library for the VHDL code. A reuse library is simply a collection of pre-compiled design units that are shared. This is currently being done for synthesizable VHDL code, but one should also use this for behavioral testbench modules. This will allow future test programs to be generated from the reuse library components reducing development time of the testbench. Also, if a design is a legacy design, the testbench may be completely reused in the new design with few modifications.

Reusing code at different levels of the design

One advantage of VHDL code is that the code can be used at different levels in the design hierarchy. Thus a test module for an ASIC on a board might be modified to test the board itself. Likewise if a board level testbench exists, a subset of the testbench could be modified to test the ASIC level design. The same is true for the system level.

Also, the digital designer may wish for the testbench to only test the functionality of the design at the board or system level and not the timing. If this is the case, the designer may use the pre-synthesis VHDL code instead of a synthesized gate-level netlist. This will allow for faster simulations to verify the operation of new design at the board or system levels, without using the computing resources to validate timing of each individual gate of the design. This allows for reduced simulation times in initial functional verification.

Using different designers to develop the hardware and the testbench

One important concept to consider is having separate designers for the device under test and the testbench. The designers should work independently, each basing his functionality on the original specification, not the other's interpretation. When the testbench and the DUT are connected together, and potential errors are uncovered, the designers can examine their interpretations of the specification and correct and eliminate the errors. This technique reduces the chances of human error in the final product.

Taking advantage of behavioral (non-synthesizable) code

Since a testbench does not need to be synthesized, the designer may take advantage of the non-synthesizable language constructs that exists in VHDL. Thus unsynthesizable "wait" statements and "loop" statements, among others can be used to implement the stimulus according to what the specification requires and can make the task of monitoring results much easier.

Using an input file can add tremendous flexibility as well. This allows the same program to read in variables that may change the overall operation of the testbench. For example, the stimulus of a testbench of an Intel 8255A, programmable peripheral interface, could contain a data file for the input, and a data file that would contain the mode word that would be written to the device. The monitor could use a data file for the expected outputs.

Testing the fabricated design using the software testbench.

One advantage of a software testbench is that it can be used at various levels of testing. For example, an ASIC design can be tested before synthesis, to verify the functionality of the ASIC can be tested; after synthesis, to verify functionality and timing; prior to release to fabrication, to verify the total operation of the system; and after fabrication, to verify the ASIC design. After the ASIC has been fabricated, this testing can occur straight from the HDL by connecting a hardware modeler to the workstation network, or by a separate tester that can use the HDL or translate the HDL to a special language required by the tester.

Using vendor technology dependent libraries

Much can be said in favor and against vendor libraries. Thus this paper will review same advantages and disadvantages.

Advantages

Efficient / Specialized Components

The vendor libraries allow the designer to take advantage of highly specialized or efficient components in their designs. Some vendors allow complex multipliers, analog-to-digital converters and even microprocessors to be added to the design. These components allow enhancement of the design to greater levels.

Architecture-Specific Constructs as With Programmable Logic Devices (PLDs)

Every PLD company has a different architecture for its devices. Thus if one's code can be designed for architecture of the PLD being used, one can get higher density of gates and greater throughput on the PLD. For example, Xilinx 4000 series FPGAs use Configurable Logic Blocks that make it easy to implement certain DSP functions such as digital filters [1].

Disadvantages

Unavailable Simulation Models and Non-Digital Simulations

One of the main problems with using these components in the design is that they are difficult to test with a testbench. For example, if a digital designer uses the vendor's microprocessor, how would the microprocessor in the system be tested. Or if an analog-to-digital converter is used, how does one test the mixed signal chip. Most HDLs do not allow for analog inputs. A second problem with the libraries is they are vendor dependent. Thus if problems occur and the vendor quits supporting the libraries or goes out of business, it will be difficult to get the design fabricated. A third problem is that most of the libraries are technology dependent. Thus if your design needs to use the latest and greatest libraries,

these devices may not exist in the newest technology.

Non-Portability across vendors and even across same vendor's technology

Many of the ASIC and PLD companies have proprietary library designs that are not compatible with any other companies' libraries. Also, old library designs becoming obsolete over time. This makes designs with the old libraries obsolete as well. One of the main goals of HDLs is to reduce design time through reuse and flexibility, and these vendor technology dependent libraries hender the flexibility of the HDL code, and can reduce the ability of a company to reuse old HDL code.

The future of hardware description languages

There are several ways that HDLs are going. Two of the most popular are system level HDLs and Analog (AHDLs) or Mixed signal HDLs. The system level HDL allows a designer to write a high-level code similar to a component specification. The synthesizer will then convert the code into a gate-level design used in ASICs or PLDs. This may eliminate the need for the lower level HDLs used today.

Analog HDLs will allow analog designer the opportunity to design using reusable code similar to VHDL[2]. Currently a large push has been made to enter analog field programmable arrays into the market. AHDLs will be necessary to the success of these new devices. Likewise with the development of AHDLs, a new set of HDLs will be developed that will combine the analog and digital HDLs into a single language.

Conclusion

In conclusion, this has been an overview of VHDL and VHDL testbenches. It has discussed topics about writing VHDL testbenches. Using a descriptive coding style, mimicking the hardware, and parameterizing the hardware have been discussed as ways of making VHDL code more flexible and easier to use by others. Also how using reuse libraries to reduce design time for the designers of future projects has been discussed. The advantages and disadvantages of using vendor dependent libraries have been reviewed. And a final look into the future of Hardware Description Languages was discussed.

Acknowledgments

I would like to acknowledge the help of my wife Shobana for helping me by reviewing this paper. Also, I would like to thank Mr. David Clark, a Senior Engineer and VHDL expert at Hughes Missile Systems Company, for his comments and suggestions.

Bibliography

Dan Biederman graduated from Tennessee Technological University with his B.S.E.E. in 1993. In 1995, he completed his M.S.E.E.. His thesis was titled *A Neural Network-Based Digital Multiplier.* He has worked at Hughes Missile Systems Company since 1996 as part of an ASIC-level and board-level digital design team. He has written several VHDL testbenches for both the ASIC-level and board-level designs.

References

1. Newgard, Bruce. **Signal Processing with Xilinx FPGAs.** Xilinx, San Jose. July 1996.
2. Rhodes, David L. *A Design Language for Analog Circuits.* IEEE Spectrum. pages 43 - 48. October 1996.