Testing and Debugging

This chapter covers some ways that PLD and FPGA designs can be tested and debugged.
You should be able to: explain the pros and cons of simulation versus hardware testing of FPGA designs; describe the use of test vectors and test benches; explain the difference between functional (RTL) and gate-level (timing) simulations; give examples of non-synthesizable VHDL features that can be used in testbenches; describe different ways to generate test vectors; select between external and internal data generators and logic analyzers.

Design Verification

Verification means testing a design to verify that it meets requirements. The effort required to verify a design often exceeds that required for the design itself.

To "re-spin" a IC design that has an error requires much time (months) and money ($100k's). Such errors are often "fatal" because the product misses a market window or for financial reasons. Thus ASIC designers check their designs carefully before "taping out" to manufacturing. ASIC verification is done through extensive simulations to verify both the functionality of the design and that it will operate at the required clock frequency.

FPGA designers have it relatively easy. Recompiling an FPGA design to fix an error may only take a few hours for a moderately complex design. In many products the FPGA is configured by software running on an attached processor and a new FPGA configuration file can be included in a firmware update. This allows the FPGA design to be updated after the product is in the field.

Simulation vs Hardware Verification

FPGA designers also have the option of testing their designs on the FPGA itself in addition to simulating their designs. This allows the design to be tested in the final circuit configuration (with the same interfaces, power supply, etc.).

Simulations have several advantages. Simulations can be easily automated as described below.

It's also relatively easy to supply test data ("test vectors") to an FPGA being simulated and collect and process the results.

On the other hand, simulations can run many orders of magnitude slower than real time. This makes it time-consuming (or impractical) to run extensive tests.

Conversely, testing a design on the actual FPGA hardware often requires building custom test jigs and using specialized equipment for testing (described below).

However, tests on an FPGA design can run in real time. This is ideal for situations where the testing needs to process a lot of of data.

The differences between testing by simulation and using the FPGA hardware can be summarized as follows:

criteria	simulation	hardware
simulation speed	slow	real-time
test vector interface	easy	hard
test automation	easy	hard
test environment	simplified	realistic

Functional (RTL) versus Timing (Gate-Level) Simulation

Functional testing verifies the design assuming zero propagation delay through the combinational logic. This checks that the logical or functional design is correct. This can be done before the design is mapped into gates and placed on specific portions of the chip.

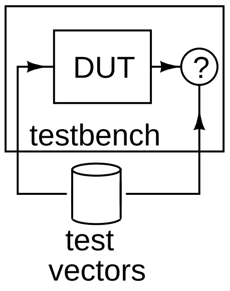
Timing simulation verifies that the design will behave correctly with the actual signal delays in the final design.

FPGA software uses timing constraints (to be covered later) and a model of delay through logic blocks and interconnects to try to ensure that the timing requirements will be met. This can only be done after the design has been synthesized and fit to specific functional block on the FPGA so that the actual delays are known.

It is typically assumed that the FPGA synthesizer will correctly estimate the delays and ensure that all timing constraints are met. Thus the main purpose of timing simulations for FGPA designs is to ensure no timing constraints have been missed and resulted in timing that is out of spec.

Test Benches and Test Vectors

A "test bench" (or testbench) is software used to test a device. Test benches typically use "test vectors" which are sequences of input and output bit vectors. The test bench applies the input part of the test vector to the device under test (DUT) and checks if the DUT's output matches the output part of the test vector. Any differences are recorded as errors that need to be fixed.



In some cases the test bench can compute the expected output itself based on a known-good model of the device behaviour instead of reading the test vectors from a file.

Test benches are often written in the same HDL[1] as the design being tested to make it easier to interface the testbench and DUT.

Since the testbench code does not need to be synthesized, it can use additional features of these HDLs. Examples of language features that are used in test benches but are not synthesizable to hardware include while-loops, file i/o and print statements. These language features makes writing a model to predict the expected behaviour much easier than it would be if we were limited to using synthesizable code.

Generating Test Vectors

In most cases it's possible to predict the desired output of a design. We can build a list of test inputs and the expected outputs for each case.

[1]Hardware Description Language of which VHDL and Verilog are the two most common.

For simple designs the test vectors can be generated by hand (e.g. using a vector waveform editor). However, in many cases the operations carried out by the DUT are too complex to compute by hand or there are too many test cases to be generated. In this case the test vectors must be computed by other means.

input to DUT	expected output
0001	0000
0010	0011
...	
0100	1010

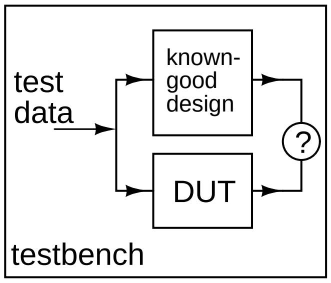
Exercise 1: Write a few test vectors for an adder with two-bit inputs and a three-bit output. Would it be practical to cover all possible input conditions for a 64-bit adder?

Typically this is done by using a reference design that is known to be correct. For example, test vectors for a signal processor could be generated by applying the input part of the test vector to a program written in, for example, Matlab. The software implementation would be supplied with test inputs representing the expected output for various inputs.

The results of the simulation might be required to be "bit exact" in which case the DUT output must match the test vectors xactly or some roundoff error may be allowed.

Selection of test inputs depends on an understanding of typical types of errors and normally will include both expected operating conditions and unusual "corner cases."

In some cases the reference design is a version of the product that already exists. In this case the test vectors can be generated by capturing the inputs and outputs of the reference unit.



In some cases it's impractical to generate test vectors because they would be too large or because the tests can be more easily written by having the testbench react to the output of the design being tested.

In this case the testbench would include code to generate the expected output from the input.

Test Strategies

It's often more effective to test components of a design individually rather than the complete design. This "unit testing" makes it easier to isolate the source of a problem.

On larger projects the design is tested periodically during development (often every night) to find newly-introduced errors called "regressions."

It's often useful to start testing before a design is complete. As each part of a design is completed, tests vectors and scripts are prepared and added to the test suite for regression testing.

Running these tests manually would take too long and be too error-prone. Scripts are used to automate testing by compiling the code, running simulations and summarizing the results.

Many EDA (Electronic Design Automation) tools, including the FPGA design and test software from Altera and Xilinx can be controlled by scripts written in a language called tcl (Tool Command Language, pronounced "tickle")[2].

Test Equipment

When testing the design on the actual hardware we need to replace the software testbench with test equipment.

Test inputs can be supplied by test equipment that outputs programmed data to the DUT. This "digital pattern generator" is a signal generator for digital signals. In many cases it's more practical to configure another FPGA to supply the test inputs.

A logic analyzer is test equipment that captures digital signals. It is similar to an oscilloscope but for digital signals. They can capture the states of many bits simultaneously. The data can be sampled at a fixed clock rate or on the edges of a signal that is defined to be a clock. Logic analyzers have flexible trigger conditions, including configurable state machines, so that they can capture very specific events. Logic analyzers differ in the number of inputs, the speeds that they can capture, the amount of memory, logic families supported and how they connect to the DUT. Probing high-speed signals requires that test connectors have been designed onto the PC board so that these signals can be probed by the logic analyzer without distorting them.

In many cases FPGA designers can replace logic analyzers with additional logic that can be configured into the FPGA to capture signals in the same was as an external logic analyzer. The on-chip logic analyzer feature is called SignalTap by Altera and ChipScope by Xilinx.

Some disadvantages of using the built-in logic analyzer are that the design must be recompiled each time there is a change to the signals that are being probed and that the logic analyzer functionality consumes FPGA resources.

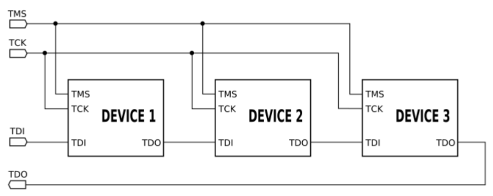
These features are controlled by a PC over the JTAG (programming and boundary scan) port on the chip.

JTAG

"JTAG" (Joint Test Action Group) is a standard for configuring and testing ICs. It is a clocked serial interface that was originally designed to test connectivity between the pins different packages on one PCB. It does this "boundary scan" by passing data through a long chain of flip-flops (i.e. a shift register) that allows setting and reading the state of each pin.

JTAG interfaces can also be used to program FPGAs, PLDs and various type of memory. They are also used to interface with on-chip debugging features such as logic analyzers.

Each JTAG port has serial data in and out pins, a clock signal and a test-mode-select (TMS) signal that is used to switch between the interfaces's control- and data-transfer modes. The data pins pass data serially between all the devices on a board while the clock and mode select signal are applied in parallel to all devices[3]:



[2]For an slightly unusual application, see the lab 4 GUI.

[3]From Wikipedia.