Approaches to Verifying Operational Test Release Vectors

Five techniques are employed to minimize the time to develop the test vectors used to test manufactured parts on an IC component tester.

by Joy Xiao Han

In today's competitive computer industry, the ability to accelerate the development process from design to manufactured part in a timely manner is an important factor. At HP's Chelmsford systems laboratory one of the things we do is design and develop chips for HP 9000 Series 800 workstations. Among the activities performed during this development is the generation of a series of test vectors called operational test release (OTR) vectors, which are used on component testers to verify the correctness of the manufactured parts.

It typically takes six months to generate and verify operational test release vectors. With the techniques described in this article we have been able to reduce the time spent creating these vectors to four months.

Test Vector Development

The final product of our test vector development process is a set of vectors (also called test patterns) that can be loaded onto a tester to verify manufactured parts. Physical defects that appear on manufactured parts can be so varied that it is often impractical to try to detect them directly. Instead, automatic test generation, waveform creation, and verification tools are employed to deal with a logical model of a physical defect, which is known as a fault. The most widely used fault model is the stuck-at fault in which the input or the output of a logic element is stuck at logic 0 or 1. For example, an open trace, assuming positive logic, might exhibit itself as stuck at logic 0.

Each vector that tests a typical block of application logic* has at least two parts. One part is made up of data and/or instructions which are applied to the input of the chip. The other part, called "expected results," is used for comparison with the actual output of the application logic to detect any faults.

The first steps of our test vector development process are shown in Fig. 1. We start by using a program called ATPG (automatic test pattern generator) from Crosscheck Corporation to produce a file containing the test input patterns, the fault-free simulation output patterns (expected results), and the scan-in and scan-out** patterns for the test logic. ATPG uses a circuit model of the chip to determine the content of the patterns produced.

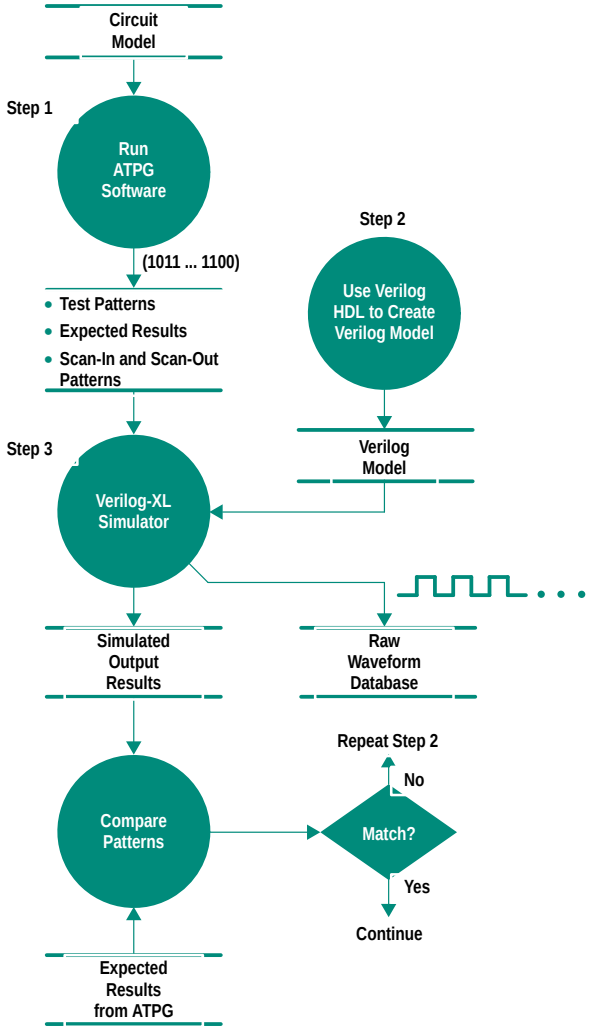

Fig. 1. The steps used to create a raw waveform data base.

The next step in our process is to use the Verilog hardware description language (Verilog HDL) to create a behavioral and structural model of the target hardware. The instructions in the program are structured to test the application logic via special test pins collectively called a test access port (TAP). The TAP provides access to test logic circuits that are built into a component to test the component itself and the interconnections between components. The TAP also provides access to circuits that allow control and observation of the

* We use the term application logic in this paper to refer to the logic on the chip that performs the intended functions of the component. The other logic on the chip is called test logic, which is included on the chip for testability.

** Scan-out patterns are also treated as expected results.

Overview of the Test Access Port

Since the emergence of surface mounted devices a great deal of concern and discussion has gone into determining how to test boards crammed with these high-density devices. In 1990 these concerns resulted in ANSI/IEEE Standard 1149.1-1990, Standard Access Port and Boundary-Scan Architecture. This standard defines test logic that can be included on an integrated circuit to provide standardized approaches to testing the component itself or the interconnections between components on a printed circuit board. The standard also allows for observing or controlling the behavior of a component during its normal operation. The test logic allows test instructions and test data to be fed to a component, and upon execution of an instruction, allows the results to be read out and observed. All instructions, test data, and results are communicated in serial format.

The test logic defined by the standard consists of a chain of boundary-scan cells and test support logic, which are accessed through the TAP inputs (see Fig. 1). A boundary-scan cell is a shift-register stage that is connected between each input or output pin on an IC and the application logic to which each pin is normally connected (see Fig. 2). The scan cell has two states of operation. One state allows a sequence of bits representing data and instructions to be shifted (scanned-in) into a chain of scan cells, resulting in latching each cell to the desired value. The scan-in and scan-out lines shown in Fig. 2 carry the bits from one cell to another. The logic specified in the standard is designed so that the serial movement of instruction data is not apparent to the circuits whose operation is controlled by the instruction.

The other state of operation for the scan cells involves testing the application logic. The test operation involves either receiving test data from the application logic via the signal-in line and then latching the output, or shifting test data into the application logic via the signal-out line. The test logic is specified such that the movement of test data has no effect on the instruction present in the test circuitry.

After the test state is done the scan mode can be invoked again to shift out the latched test results for comparison with the expected results.

The clock, shift, and mode lines shown in Fig. 2 are controlled by the TAP signals (described below). The TAP lines are responsible for sending the proper signal sequences to control the scanning or testing states. In addition, the mode line is controlled according to the type of pin it is connected to (e.g., input, output, bidirectional, tristate, etc.).

The IEEE standard defines a minimum of three input connections and one output connection (see Fig 1). An optional fourth input (TRST*) provides for asynchronous initialization of the test logic circuitry defined in the standard.

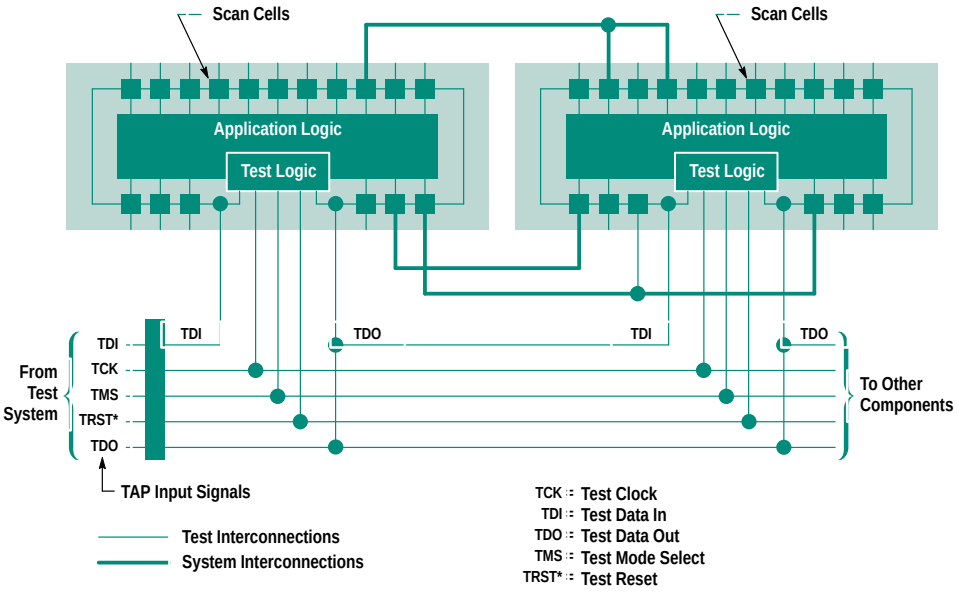

Fig. 1. A simplified block diagram of the test logic defined in ANSI/IEEE Standard 1149.1-1990 surrounding application logic.

application circuits during their normal operation. The specification for the TAP logic is given in IEEE Standard 1149.1-1990, and a brief overview is provided above. Also included in the Verilog HDL model are calls to a utility called $tds_monitor()* which associates timing information with the bit patterns sent to the device under test. These calls will create a raw waveform database containing timing and pattern information. It is called a raw waveform database because during simulation runs every change on the component's pins is included in the database. One of the techniques described below explains how this data is manipulated to produce a more refined waveform database.

The third step shown in Fig. 1 involves running the Verilog-XL logic simulator using the Verilog HDL model created in step 2 and the patterns created in step 1 as inputs. A waveform database and an output pattern file are created from this simulation. The output patterns from the simulation are compared to the expected output patterns generated by the ATPG software. If the patterns don't match, steps 2 and 3 are repeated until they do. If the patterns do match, we move on to prepare the waveform database to become our operational test release vectors.

A great deal of time can be spent going back and forth between steps 2 and 3. The rest of this paper describes some techniques that I have found to be helpful in getting through steps 2 and 3 quickly. These techniques verify that the TAP circuits are functioning properly.

* The $tds_monitor() runs as part of the Verilog-XL simulator.

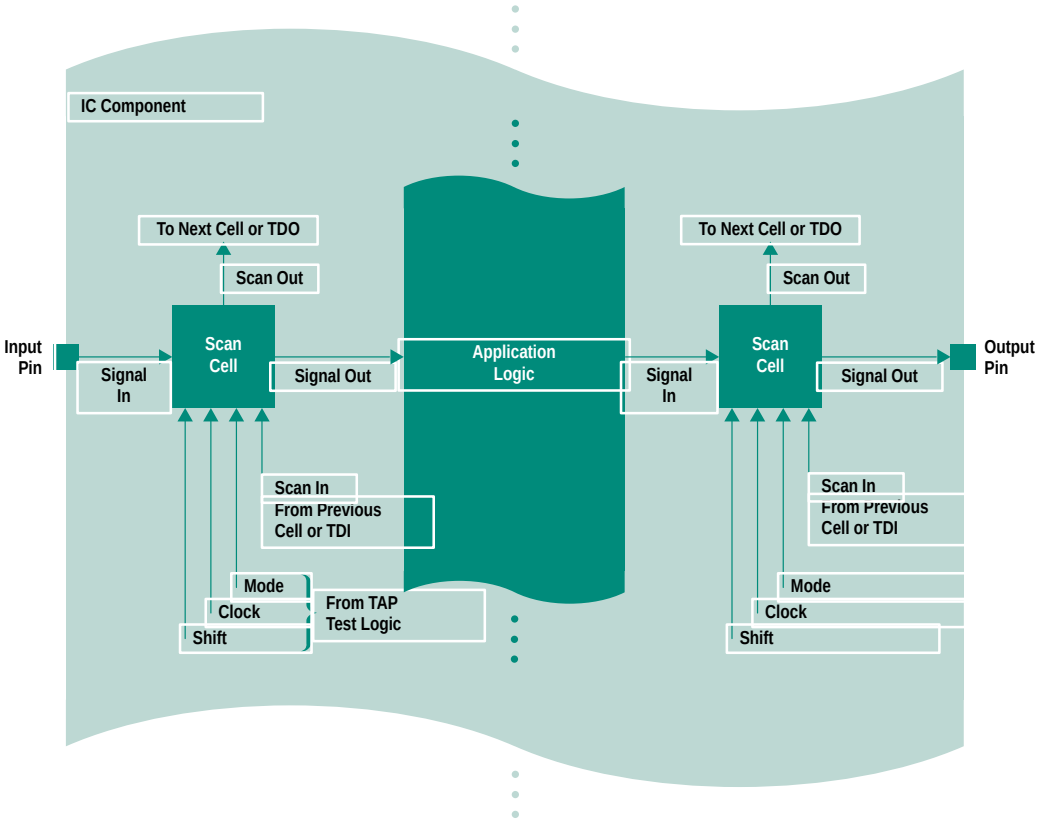

Fig. 2. The location of scan cells in relation chip I/O pins and the application logic. The mode, clock and shift signals are derived from the TAP input signals TMS, TCK, and TRST* respectively. The first scan-in signal and the last scan-out signal correspond to TDI and TDO respectively.

Test Clock. TCK is the test clock input that provides the clock for the test logic. This clock is provided so that scan cells surrounding the application logic can be controlled independently of system clocks. TCK allows shifting of test data concurrently with system operation (allowing online monitoring). It also ensures that test data can be moved to or from a chip without changing the state of the application logic.

Test Mode Select. TMS is the signal used by the TAP controller to control test operations. One use of TMS is to select whether the test circuitry is in the test state or the scan state. To guard against race conditions, the TMS signal like the TDI signal described below must be sampled on the rising edge of TCK.

Test Reset. The optional TRST* signal is included to allow for asynchronous reset of the TAP controller. The reset signal only affects the test logic and has no impact on the application logic.

Test I/O Lines. TDI and TDO are the test data input and output lines respectively. They provide for the serial movement of test data through the circuit. Data presented at TDI is clocked into the selected register on the rising edge of TCK, while output data appearing at TDO is clocked out on the falling edge of TCK. To simplify the operation of components that are compatible with the standard, data must be propagated from TDI to TDO without inversion.

Bibliography

1. *IEEE Standard Test Access Port and Boundary-Scan Architecture,* IEEE Std. 1149.1-1990, IEEE Standards Board, May 1990.

2. R. G. Bennets, *Design of Testable Logic Circuits,* Addison-Wesley, 1984.

3. V. D. Agrawal and S. C. Seth, *Test Generation for VLSI Chips,* IEEE Computer Society Press, 1988.

Technique 1

Check the scan chain* without a system clock. This step is mainly used to make sure that the scan chain on the chip is not broken. This test works by ensuring that whatever value is scanned in (SI in Fig. 2) should be exactly identical to the value scanned out (SQ). The scan chain is advanced by clock pulses. For example, a pulse from CLKA followed by a pulse from CLKB causes the data on SI to be propagated to SQ. SQ turns into the SI for the next scannable flip flop on the scan chain. If the chip passes this test, we know that the scan logic is set properly and that there is continuity in the scan chain.

Technique 2

Check the scan chain with a system clock. This check verifies that the combinational logic in the application logic portion of the chip works. Usually the master clock (MCLK) is opposite in state to the system clock (CLK) (i.e., when CLK is on, MCLK is off and vice versa). The only exception to this behavior occurs when we execute a double-strobe test to check the time margin on the chip.

The opposite clock states are verified by ensuring that the data on the D input in Fig. 2, which is the result of all previous combinational logic, is inverted at MQ when CLK is low. On the other hand, when CLK is turned on, the inverted value of MQ (the exact value of D) appears at Q. This is the same value we can monitor at SQ by advancing the scan chain with pulses from CLKA and CLKB in the correct sequence.

Technique 3

Check the TAP state sequence. Since the TAP logic is basically a state machine its current state is recorded in a mode register. I have found it necessary to pay attention to the

* A scan chain is a shift-register path through a circuit which is typically placed there to improve testability. See "Overview of the Test Access Port" on page 56.

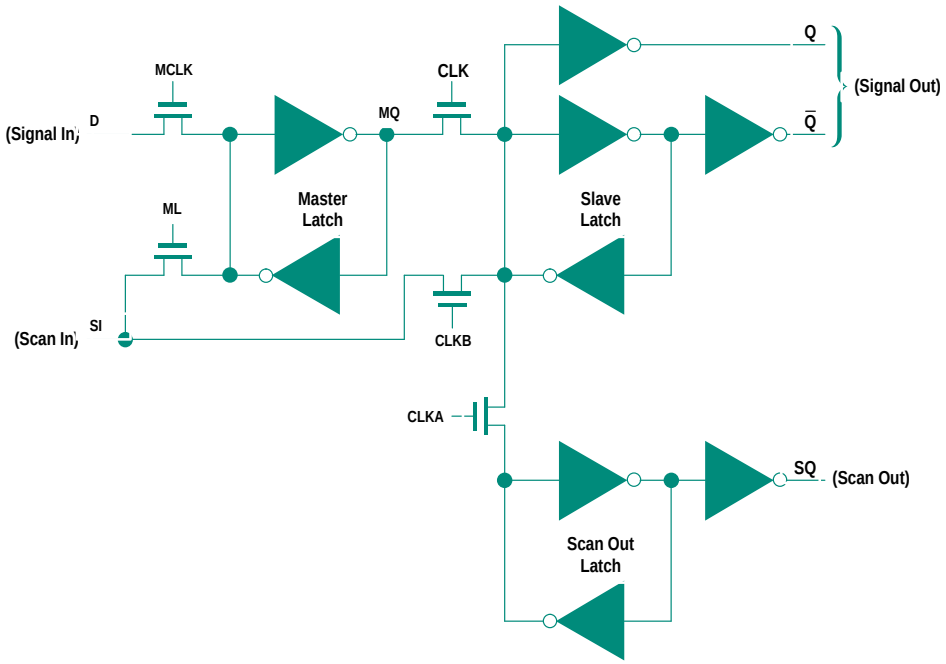

Fig. 2. Scannable flip-flop. This is a portion of our implementation of a scan cell. The MCLK, ML, CLKA, and CLKB signals are controlled by the TAP test logic and are derived from the TAP input signals TMS, TRST*, and TCK.

previous state of the TAP circuit by checking certain bits in the mode register. For example, one error that typically takes a long time to correct occurs when the bit in the mode register that controls I/O direction (PSCAN in Fig. 3) is not set properly during initialization. Forgetting to set this bit causes TSTDEN (test data enable) to go high during a scan. Later when data is supposed to be coming out of the chip (via the I/O pin) and the tester is driving a value into the chip, a bus fight occurs. We want TSTDEN to be low, which puts the gate in tristate mode when the tester is driving a signal onto the pin. Anytime the wrong data is output because of something that is done or not done early in the test cycle, it always takes a long time to debug. Debugging time can be saved if each state (bits in the mode register) is closely monitored.

Technique 4

Check the value of each pin before each system clock. One bug that occurs frequently is that test vectors will run smoothly during simulation, but cause bus fights when they are run on the tester. From Fig. 3 we can see that if BUSIN and the tester drive an identical value onto the I/O pin at the same time, a problem occurs that can only be caught by the tester and not by simulation. This problem can be eliminated if we stop the simulation before each system clock and make sure that the I/O pin is driven by either BUSIN when the pin acts as an output (i.e., the pins drive the values to the BUSOUT), or by the tester when the pin acts as an input, but not both at one time. This task can be easily accomplished by using the Verilog-XL command $showvars.

Technique 5

Verify that the process of creating the operational test release (OTR) vectors for the tester is correct. Fig. 4 shows the additional steps we take to create and verify the OTR vectors. The first thing we do is take the raw waveform database described above and use the conditioners* ALIGN and WINDOW to create a more refined waveform database. We use the ALIGN conditioner to align each signal to be edge triggered. The WINDOW conditioner is used to select certain cycles or windows within a cycle during which output data is valid, which results in monitoring pins only at the times we care about. The next step is to verify that the windowed data is okay. This involves the same process we went through in steps 2 and 3 of Fig. 1. If everything is okay, the waveform database is converted to OTR vectors to run on the tester.

* Conditioners are functions that provide a way to modify waveform data generated via $tds_monitor. The ALIGN and WINDOW conditioners come from TSSI Inc.

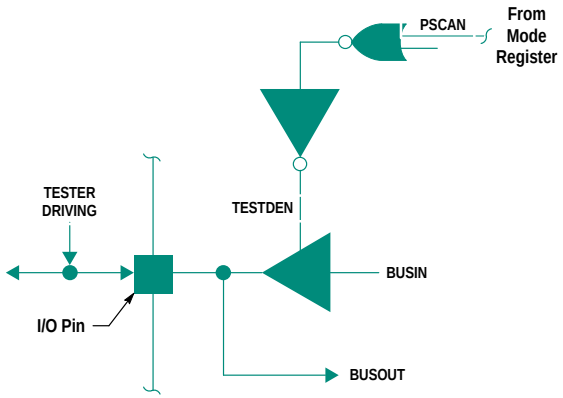

Fig. 3. A simplified diagram of the circuitry around an I/O pin.

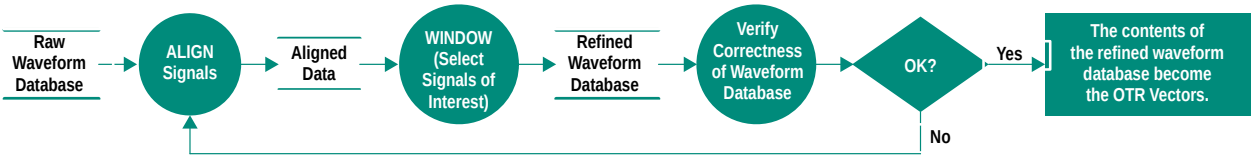

Fig. 4. The final steps in creating the OTR vectors.

Conclusion

These five techniques have proven to be a success in the Chelmsford systems lab by shortening the time it takes us to produce final test vectors. These techniques can also be applied to non-ATPG OTR vectors, since we can create vectors manually to meet different needs and put them into ATPG format. We characterized the entire analog circuitry embedded in one chip by controlling the proper bits on the scan chain.

Acknowledgments

I would like to acknowledge all those from the Chelmsford systems lab and the Corvallis Integrated Circuit Business Division lab who contributed to the slave memory controller's operational test release. Also, special thanks to my managers for their continuous recognition and support.