By Ken Jaramillo and Subbu Meiyappan, Philips Semiconductors

PART ONE OF THIS SERIES REVIEWS BASIC SCAN-TEST TECHNIQUES. PART TWO PRESENTS 10 DESIGN PRINCIPLES TO FOLLOW FOR SUCCESSFUL IMPLEMENTATION OF THOSE TECHNIQUES.

10 tips for successful scan design: part two

ONCE YOU KNOW THE BASICS of scan design (see "10 tips for successful scan design: part one," pg 67), you're ready to take some action. The following list includes the most important issues to be aware of to guarantee successful adoption of scan techniques within your company or design group:

- Handle internal tristate buses with care and avoid bus contention by design.
- Make all clocks and asynchronous resets come from chip pins during scan mode.
- Ensure that all scan elements on a scan chain are in the same clock domain.
- Know the requirements and limitations of your chip testers.
- Handle mixing flip-flops triggered off different edges of the clock with care.
- Break all combinational-logic feedback loops.
- Handle all nonscan elements with care.
- Avoid design practices that lead to nonscannable elements.
- Handle multiple clock domains with care to avoid potential timing problems.
- Plan chip-level scan issues before you start block-level design.

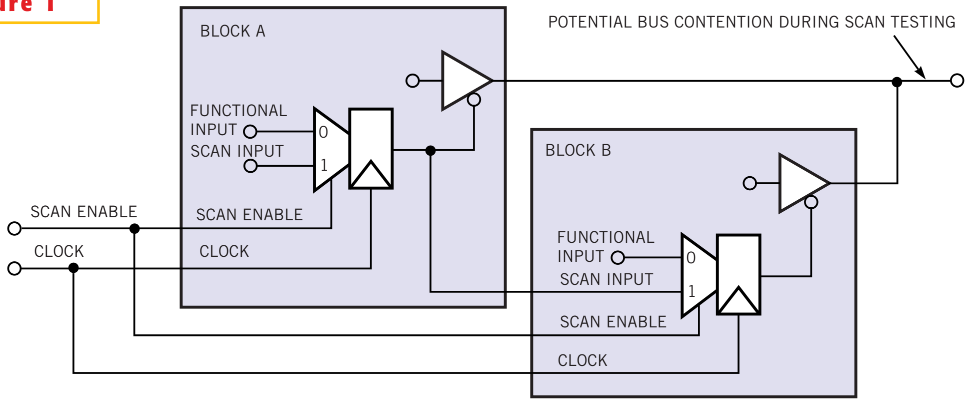
INTERNAL TRISTATE BUSES

Without a doubt, the biggest hurdle to overcome in system-on-chip (SOC) designs with respect to automatic test-pattern generation (ATPG) is the proper control of internal tristate bus structures. If possible, never implement designs with internal tristate bus structures. If you can't follow this rule, then implement the fewest possible internal tristate-bus structures and guarantee by design that no bus contention can occur on any internal bus during scan testing.

Two control problems require careful consideration. First, you must ensure that there is no contention on the tristate buses during scan-shift operations. Most scan-insertion tools can automatically perform this task during the scan-insertion phase. Second, you must ensure that there is no possible contention on the internal tristate buses during the capture cycles during scan testing.

With most designs, you can generate a scan-test pattern that causes bus contention on some internal buses. Several ATPG tools are intelligent enough to avoid generating patterns that cause bus contention. However, although the tools may be intelligent enough to avoid contention, this intelligence

Figure 1



When two flip-flops control the output enable for bus transceivers, bus contention can occur.

takes a fair amount of CPU effort. Depending on the design, the ATPG effort could be so CPU- intensive that it results in longer runtimes, fewer patterns, and lower fault coverage. If the ATPG tool cannot identify scan-test patterns that create bus contention and uses those vectors to test the device, then the part may be stressed during the production test. This production-test stress may cause the part to fail on the tester, cause damage, or cause a shortened life cycle. Therefore, avoiding contention on internal tristate buses is important.

Bus-contention problems in SOC designs occur at two levels. The first problem is within a design block that contains multiple drivers to a tristate port. The second problem is at the chip level, in which multiple blocks interface to the same bus. Consider the case of an internal PCI-bus structure. In normal operation, bus-arbitration logic via the request/grant pairs guarantees that only one master controls the bus at a time. During scan testing, however, the ATPG tool easily generates test patterns that would turn on multiple requests, grants, and output-enable signals for bus transceivers, thus forcing multiple devices onto the tristate bus simultaneously.

In **Figure 1**, two blocks drive a common internal tristate bus. The **figure** represents a single bit of the bus. In each block, scan flip-flops control the output enables for the bus transceivers. The last flip-flop in Block A's scan chain drives the first flip-flop in Block B's scan chain. If the ATPG tool generates a pattern that causes both flip-flops to shift in values of zero, then you have bus contention on this bit of the bus.

Although there are several approaches to these types of problems, the solutions are generally simple. (You can download the **appendix** "Internal PCI-bus-contention solution" from *EDN*'s Web site, www.ednmag.com/ednmag/reg/2000/02172000/04ms604.htm) It is important to recognize that if you use internal buses, you must guarantee by design that bus contention is impossible during scan testing.

CLOCKS AND ASYNCHRONOUS RESETS

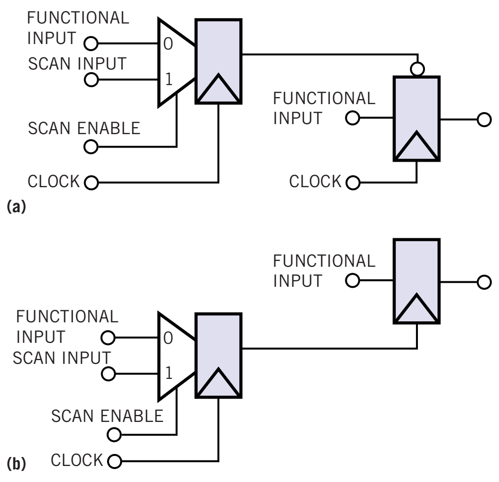
The next biggest issue when it comes to achieving high fault coverage is to ensure that all clocks and asynchronous resets come from chip pins during scan testing to allow the ATPG tool to control clocks and resets in the design. Neglecting this fact will cause the ATPG tool to consider each potential scan element that does not have a clock or reset coming from a chip pin as unscannable. The tool considers all unscannable cells as unknowns during pattern generation, resulting in reduced fault coverage.

This suggestion does not imply that all clocks and resets must come directly from pins. Rather, the ATPG tool must have total control of scan-element clock and reset signals. The tool must be able to control the clocks and be able to deassert the resets. The following examples demonstrate this situation.

Figure 2a shows one flip-flop in the scan chain driving the asynchronous set or clear pin of another flip-flop. You must avoid this design practice. As data shifts around the scan chain, the second flip-flop resets set or clear depending on the shift data. The ATPG tools cannot produce useful scan patterns for this type of circuit. Thus, the tool doesn't include the second flip-flop in the scan chain and considers this flip-flop as an unknown during pattern generation, resulting in a loss of fault coverage. If this type of logic exists, you should insert a multiplexer in the reset path of the second flip-flop. This multiplexer allows a chip pin to control the reset signal during scan test or disables the reset pin of the flip-flop.

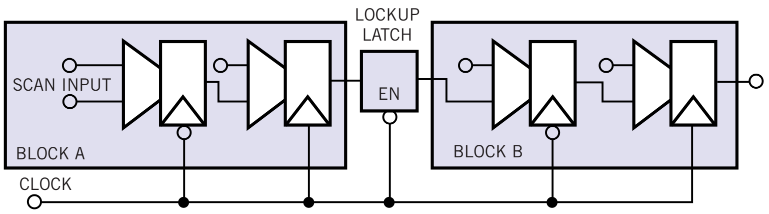
In general, you want to avoid any asynchronous flip-flop input that a chip-level reset pin can't disable. Therefore, if an asynchronous-reset input of a flip-flop ties to the output of some combinational logic that a chip-level reset pin cannot disable (the logic may have scan flip-flop outputs as its inputs or even chip-level

Figure 2



Two design practices to avoid are using one flip-flop in the scan chain to drive the asynchronous set or clear of another flip-flop (a) and using one flip-flop in the scan chain to drive the clock of another flip-flop (b).

Figure 3



If the design includes falling-edge-triggered flip-flops, place all of these flip-flops at the beginning of the scan chain for each block and insert lockup latches between blocks.

inputs), then you must insert a multiplexer. Note that if the offending signals that prevent a single chip-reset pin from disabling the flip-flop's asynchronous input are chip pins, then you can solve the problem by forcing the ATPG tool to drive these pins to constant values during pattern generation. This approach is easier than adding multiplex circuitry.

Figure 2b shows one flip-flop in the scan chain driving the clock input of another flip-flop. You must also avoid this design practice if possible. As data shifts around the scan chain, the second flip-flop's clock toggles, depending on the shift data. The ATPG tools cannot produce useful scan patterns for this type of circuit. Thus, the scan chain does not include the second flip-flop, and the tool considers this flip-flop as an unknown during pattern generation, resulting in a loss of fault coverage.

This type of design exists for circuits such as clock dividers. Therefore, if this type of logic exists, you have two options. First, you can insert a multiplexer in the clock path of the second flip-flop such that the clock input ties to one of the scan clocks only during scan-test mode. Because this approach introduces logic in the clock path, the clocks between the flip-flops are no longer synchronous. Therefore, you should insert a lockup latch in the scan chain before and after the second flip-flop to avoid any potential hold-time problems. If several instances of this circuit exist, you may want to create a clock that all these flip-flops can use during scan-test mode. In this case, you would need to place a lockup latch only before the first and the last of these flip-flops.

Second, you can insert a multiplexer in the second flip-flop's asynchronous reset path to tie this flip-flop active, which holds the flip-flop in reset only during scan-test mode. This approach is less effective than the first one but is better than having the ATPG tool consider the second flip-flop as an unknown.

In general, you want to avoid any clock input that a single chip-level clock pin can't control. Therefore, if a flip-flop's clock input ties to the output of some combinational logic that a single chip-level clock pin cannot control, then you need to insert multiplex circuitry, just as in the previous example. The logic may have scan flip-flop outputs as its inputs, chip-level inputs, and even chip-level clock inputs. If the offending signals that prevent a clock pin from controlling the flip-flop's clock input are themselves chip pins, then you can force the ATPG tool to drive these pins to constant values during pattern generation. This approach is easier than adding multiplex circuitry.

SCAN ELEMENTS

Several factors determine the number of scan chains in a design. In general, you want to divide scan chains by clock domain. All flip-flops in a scan chain should use the same clock. However, some factors might make this situation undesirable.

First, each scan chain must have its own scan-input and -output pin. The more scan chains you have, the more pins you must set aside for test. If you don't dedicate pins for test, you must dedicate logic to multiplex the scan inputs and outputs with other chip pins. Also, the production tester that tests the chips has limitations that affect the number of scan chains a design can support. Finally, it is generally a good idea to equalize scan-chain lengths. Remember that each scan pattern is as long as the longest scan chain.

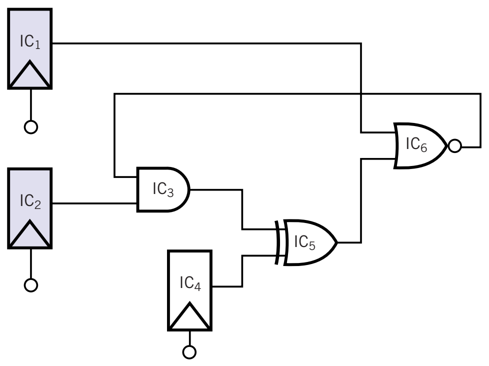
For example, consider a design with 10 scan chains, one chain being 1000 flip-flops long and the rest being two flip-flops apiece. Each scan chain requires a test pattern that is 1000 bits long for each test pattern, even though the chains that are two flip-flops long have patterns that contain 998 don't-care bits and only two real-test bits. So, it may be wise to break some of the longest chains into multiple chains. If you decide to combine scan chains based on different clocks—to equalize the scan-chain lengths or to compensate for tester limitations, for example—make sure you place lockup latches between the scan chains to avoid potential hold-time problems.

It is also a good idea to include lockup latches between chip-level blocks even if the blocks are in the same clock domains. Inserting these latches is unnecessary if you've done an accurate static-timing analysis, but using the latches minimize the chances for hold-time problems between blocks. Preventing hold-time problems is important because most of the effort relating to scan begins after you deliver a final netlist for placement and routing of the chip. Also, you typically develop static timing scripts for scan paths after the development of scripts to verify functional paths. Adding these lockup latches adds few gates to the design, and the latches help to avoid potential timing problems that you might not find until late in the design cycle.

REQUIREMENTS OF CHIP TESTERS

You have to know the limitations of your production tester before you can plan an effective strategy for scan. Two limits impact test. The first is test time. In general, you should design production tests to operate in less than 3 sec, roughly the cycle time of the device handler. Tests that take longer than 3 sec result in excess cost per chip. The second limit to keep in mind is tester memory. The entire test program must fit into the available memory of the tester. You can never reload test memory in the middle of the test. Dedicated scan-hardware constraints limit most testers to a certain number of scan chains. **Table 1** shows one possible configuration of an exam-

Figure 4



Combinational feedback loops, from IC_6 to IC_3 in this case, can produce an unstable output. You need to break this loop during scan mode for the ATPG tool to predict the operation of the circuit.

ple tester with 128 Mbits of memory.

In **Table 1**, the example tester supports a maximum of 32 scan chains. The number of scan chains you choose determines how much memory you have to work with, which directly impacts the number of test patterns the tester can support. Most testers work on even numbers of scan chains. For example, if a design has nine scan chains, the available memory per chain is still 8 Mbytes; the remaining memory is inaccessible for the scan test.

TABLE 1—POSSIBLE TESTER MEMORY CONFIGURATIONS

Maximum number of scan chains	Available memory per chain (Mbits)
One	128
Two	64
Four	32
Eight	16
16	8
32	4

The following example shows how to determine the number of allowable test patterns you can generate based on the tester's memory limits:

Tester_Memory_Per_Chain> (#Scan_Patterns·Max_Scan_Chain_Length)+Max_Scan_Chain_Length.

Tester_Memory_Per_Chain=Total_Tester_Memory/Number_Of_Scan-chains.

Number_Of_Scanchains is in multiple of two increments (except if you have only one scan chain).

1 Mbyte of memory=1,048,576 bits.

#Scan_Patterns<(Tester_Memory_Per_Chain−Max_Scan_Chain_Length)/Max_Scan_Chain_Length.

For example, if the amount of memory available on the tester is 256 Mbytes, your design has eight scan chains, and your longest scan chain is 3000 flip-flops long, then

Tester_Memory_Per_Chain=256 Mbits/8=33,554,432 bits=32 Mbits.

#Scan_Patterns<(33,554,432−3000)/3000=11,183 ATPG patterns.

This example shows how to calculate tester-memory limitations. These calculations depend on the type of tester you use. You should consult the test-engineering personnel of your tester's manufacturer for details.

MIXING FLIP-FLOPS

ATPG tools require that you place all falling-edge-triggered flip-flops at the front of a scan chain. If you place a falling-edge-triggered flip-flop after a rising-edge-triggered flip-flop in the scan chain, a single clock cycle will clock scan data through both flip-flops. This situation causes some loss of coverage because the two flip-flops always have the same scan data value after a shift cycle.

Fortunately, handling a situation in which several blocks within a chip may have falling-edge-triggered flip-flops doesn't mean that you have to place all falling-edge-triggered flip-flops at the front of the entire scan chain, which consists of multiple chip-level blocks. Whenever a falling-edge-triggered flip-flop follows a rising-edge-triggered flip-flop in a scan chain, you must insert a lockup latch between them. The lockup latch prevents data from shifting through both flip-flops in one clock cycle. To avoid having an excessive amount of lockup latches, it is still advisable to place all the falling-edge-triggered flip-flops at the beginning of the scan chain for each block. Then, you need to place lockup latches only between blocks. **Figure 3** shows two chip-level blocks, A and B, each containing falling-edge-triggered flip-flops. The blocks' scan ports connect together via lockup latches at the chip level.

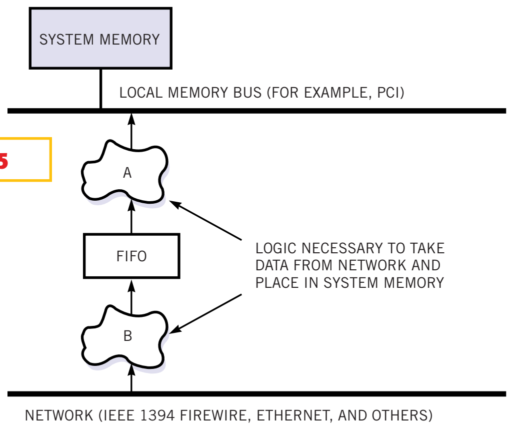
A few ATPG tools have difficulty handling falling-edge-triggered flip-flops during capture cycles and may require special commands to inform the tool how to handle the flip-flops. You should investigate how your ATPG tool handles this situation. You may even consider changing the circuit so that these flip-flops trigger off the rising edge of the clock during scan mode rather than off the falling edge. But remember, modifying the clock inputs of these flip-flops effectively puts them in a different clock domain, requiring that you add lockup latches before and after these flip-flops in the scan chain.

COMBINATIONAL-LOGIC FEEDBACK LOOPS

Designs that contain combinational-feedback loops have inherent testability problems. Combination-feedback loops may introduce internal logic states to a design that scan-storage elements cannot control. Consider a circuit with three flip-flops and a combinational feedback loop from IC_6 to IC_3 (**Figure 4**). This feedback loop causes the problem. If you initialize the flip-flops to values of $IC_1=0$, $IC_2=0$, and $IC_4=1$, then the output at IC_6 will be a stable high. If IC_2 changes to a logic high, then the output at IC_6 oscillates between 0 and 1. Because of this oscillation, ATPG tools cannot predict the operation of the circuit. To generate patterns, the ATPG tool would have to break this loop, which would reduce overall fault coverage. ATPG tools have a few methods for breaking combinational feedback loops. Some of these methods are less harmful to fault coverage than others, but all of them result in some loss of coverage. Therefore, you should avoid combinational feedback loops whenever possible. Most ATPG tools inform users of all the combinational-feedback loops present in a design.

If you cannot avoid these feedback loops, then you should break the feedback loop by inserting an additional flip-

Figure 5


SYSTEM MEMORY
LOCAL MEMORY BUS (FOR EXAMPLE, PCI)
A
FIFO
B
LOGIC NECESSARY TO TAKE DATA FROM NETWORK AND PLACE IN SYSTEM MEMORY
NETWORK (IEEE 1394 FIREWIRE, ETHERNET, AND OTHERS)

You need to carefully handle the FIFO to be able to test the surrounding logic blocks, A and B.

flop that is present in the feedback path only during scan-test mode. This modification results in the highest fault coverage. If you cannot insert a flip-flop, then insert a multiplexer in the feedback path that drives a constant value during scan-test mode. This approach results in lower coverage than the flip-flop option but higher coverage than if you allow the tool to break the loop by assuming an unknown value as a result of the loop.

NONSCAN ELEMENTS

Scan-insertion tools consider all cells that do not have a scan-equivalent cell as black boxes and do not insert them into a scan chain. ATPG tools consider sequential cells that are not on scan chains as black boxes. Therefore, you must treat all nonscan sequential elements with care to avoid loss of fault coverage. Examples of nonscan elements are latches, RAM, and design blocks that do not include scan.

Although latch-based designs are popular for gate and power savings, most scan/ATPG tools do not handle these designs optimally. ATPG tools can understand the behavior of latches that the design holds transparent, but these tools model the behavior of latches when they are not transparent as unknowns. If the latch data feeds into other logic that a scanned register then captures, poor fault coverage could result. In general, you should keep all latches transparent during scan testing, but you should investigate how your scan/ATPG tool handles latches.

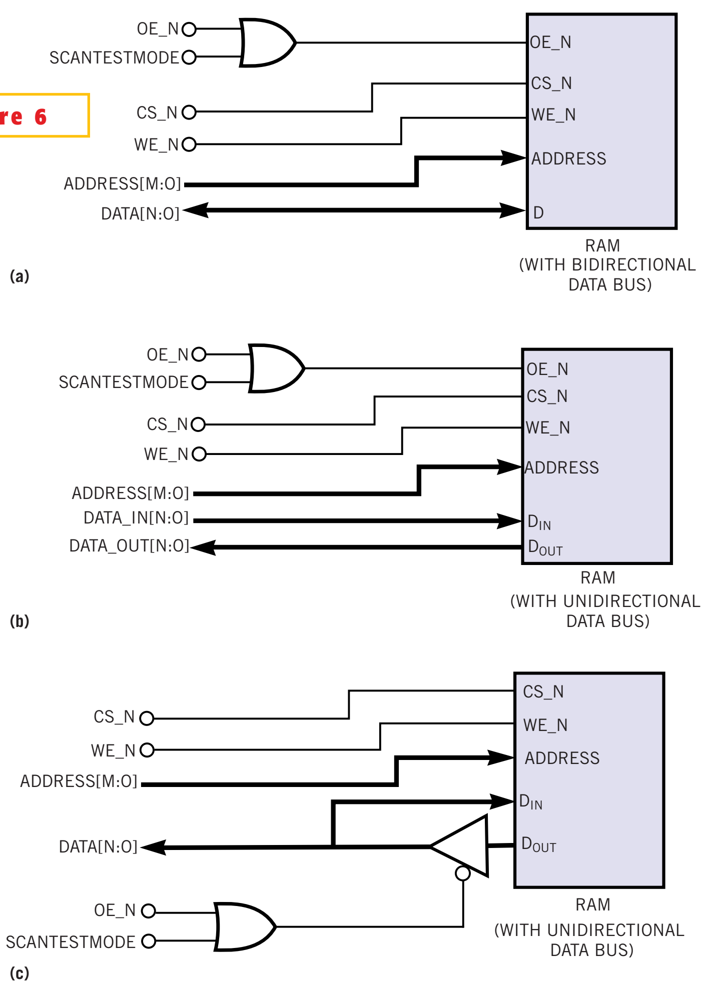
RAM cells have more complex failure modes than do the simple "stuck-at" modes for standard-cell logic, such as flip-flops, latches, and combination-logic gates. Thus, scan techniques do not verify RAM circuits during production testing. Instead, you use RAM built-in self-test (BIST) to verify RAM cells. This technique involves writing several patterns into the RAM array to check for the various failure modes of RAM cells. BIST is a well-known technique. (See **Reference 2** for an introduction to BIST and testing of RAM and ROM.)

Because you test RAM via BIST, which achieves high fault coverage, ATPG tools need not test and fault-grade RAM. However, even though BIST logic makes RAM fully testable, a large reduction in test coverage of the surrounding logic may result. This reduction is commonly known as the shadow effect. Imagine a FIFO array for which surrounding logic pushes data in one side and pulls data out the other. In this case, the test doesn't include the logic surrounding the FIFO array unless you use special techniques to make the FIFO ATPG-tool-friendly. **Figure 5** shows an example of a RAM array (FIFO) for a networking application. The logic in A and B grabs the data off the network and places it in system memory. If you don't handle the FIFO carefully, you could lose a lot of fault coverage. You need to design the FIFO so that you can observe the outputs from the logic in B and control the inputs to logic A during scan-test mode.

Several techniques can increase the observability of logic immediately before the RAM and increase the controllability of logic immediately after the RAM. Support for these techniques varies among ATPG tools. So, investigate your ATPG tool's capabilities before you decide how to handle RAM. Keep in mind that there are many types of RAM blocks. Some have bidirectional data buses, and others have unidirectional data buses. Some are synchronous, and others are asynchronous. The best technique to use depends on your application and your vendor library. The approach you ultimately choose depends on your design architecture (a single bidirectional bus or two unidirectional buses), your timing budget (whether the design can withstand additional logic in the datapaths), and your ATPG tool (whether it supports modeling of RAMs).

There are five ways to deal with RAM. The first and easiest approach is to isolate the RAM block by deasserting its output-enable signal during scan mode. This method adds no observability or controllability for the RAM, but it gets the RAM off the bus so that it does not

Figure 6



Deasserting the output enable during scan-test mode isolates RAM with bidirectional (a) and unidirectional data buses (b, c).

interfere with the other blocks on the data bus. Implementation depends on the type of RAM the design uses. **Figure 6a** shows the logic necessary to isolate a RAM that has a bidirectional bus using the RAM's output-enable signal. **Figure 6b** is similar to **Figure 6a** except the RAM has separate data-in and data-out buses, and the design uses these buses separately. **Figure 6c** is similar to **Figure 6b** in that the RAM has separate data-in and -out buses, but this implementation combines the data buses into one bidirectional data bus.

The second approach is to isolate the RAM block by inserting a multiplexer to drive the data signals during scan mode. The drive values can be either constant or some combination of the input-control signals. This approach is useful only for RAM blocks with unidirectional read-and-write data buses, but it is more sophisticated than the previous approach, which simply disables the RAM. In this case, you allow the test to drive a constant pattern onto the output data bus. Thus, this approach adds controllability to the logic immediately after the RAM because the ATPG tool can affect the output data of the RAM, although only in a simple way. **Figure 7a** shows an implementation using unidirectional data buses; **Figure 7b** uses a bidirectional data bus. Both figures show a RAM block that has unidirectional data buses.

Third, you can place the RAM block into a transparent mode during scan test (**Figure 8**). In this mode, you essentially route data-in to data-out. Note that this approach is only useful for RAM blocks with unidirectional read-and-write buses and with designs that use the buses separately. This solution not only provides controllability of the logic immediately after the RAM, as in the previous approach, but also provides observability to the logic immediately before the RAM.

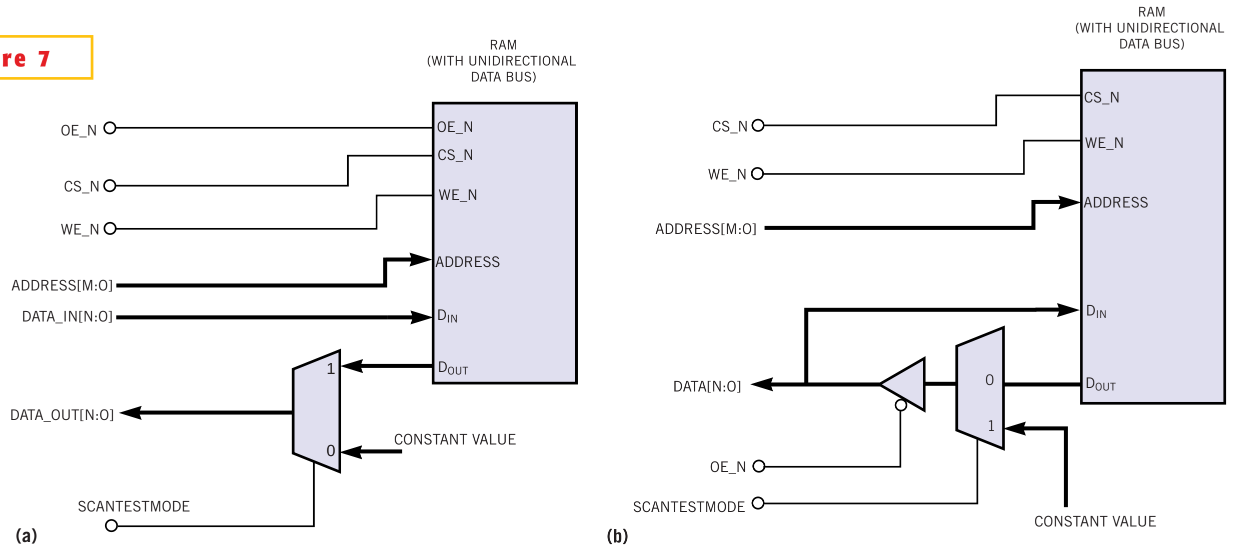
A fourth approach is to write RAM data before scan test and use the RAM contents to generate test data for the surrounding logic during scan test. To avoid disturbing the RAM contents during scan test, disable the RAM write signal during scan-test mode. This method requires the ATPG tool to support a functional RAM model. Some ATPG tools allow for only a partial initialization of the RAM array.

This approach adds a lot of controllability to the logic immediately after the RAM but no observability to the logic before the RAM. Another drawback is the length of time that initializing the RAM array might require. Yet another potential drawback is that this approach requires the ATPG tool to support a functional RAM model. Because many ATPG tools support RAM models, this issue may not be critical. **Figure 9a** shows the implementation for which you must initialize the entire RAM before scan testing and then allow the contents of the RAM to test the surrounding logic. **Figure 9b** shows the implementation for which you need initialize only a single location within the RAM before scan testing. This approach saves time on initialization but provides less controllability than the logic in **Figure 9a**. You can create an approach between these two that requires initialization of only a certain number of locations, for example, the bottom 1 kbyte of memory.

The fifth approach is to leave the RAM as is and let the ATPG tool functionally exercise it to generate logic values to test the surrounding logic during scan test. This method requires that the ATPG tool supports a functional-RAM model. This approach requires no changes to the hardware and provides the most observability of the logic before the RAM and controllability of the logic after the RAM. However, this approach requires an ATPG tool that can model RAMs and a little more effort to learn how to use it.

Portions of your chip may not include scan circuitry. Examples are older versions of blocks and third-party intellectual property. You can test these blocks by using canned test vectors, logic BIST, or other testing methods. However, you need to be careful that the lack of scan in these blocks does not hurt the ability to test other blocks in the chip. You can use multiplexer-isolation techniques to separate any nonscan blocks from the scan section of the design during scan mode. Well-designed scan and nonscan isolation with appropriate control logic re-

Figure 7

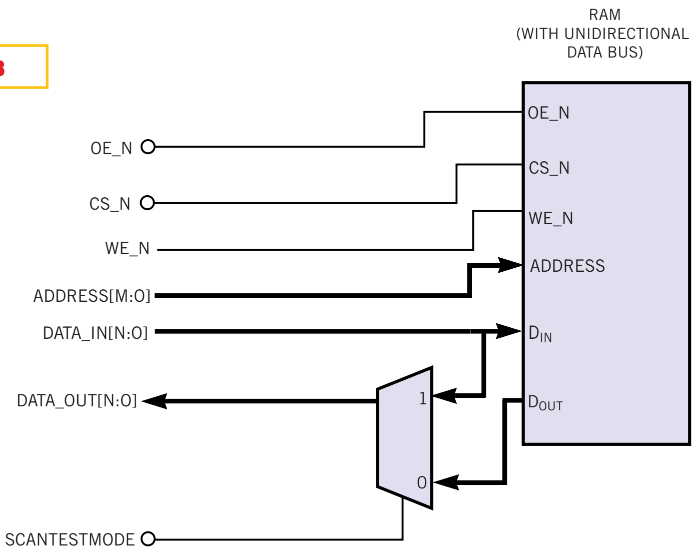


You can isolate RAM during scan test by driving output data to a constant value in the case of both unidirectional (a) and bidirectional (b) data buses.

sults in fast and trouble-free test-program generation with high fault coverage. To increase the fault coverage that you can obtain from scan ATPG software, the isolation circuitry of nonscanned blocks should set the outputs of such blocks to known logic states during scan mode.

Examples of nonscanned flip-flops are flip-flops that have no scan equivalent and thus can't be part of the scan chain. Nonscanned flip-flops can also be flip-flops that you design into the circuit such that you could not place them on a scan chain because of improper generation of clock or reset inputs (see tip 2). You should first try to fix the cause of the problem. For example, if the flip-flop has no scan-equivalent cell in the ASIC library, then change the design so that it uses a scan-type flip-flop instead. If problems exist with the generation of the clock or reset inputs, then change the design according to the recommendations in tip 2. If you can't modify the design, the last resort is to design access to the preset or clear connectors such that the system, or tester, holds the flip-flops in known states, either preset active or clear active, during scan mode. This approach somewhat reduces fault coverage because it limits the controllability of input nodes of the flip-flop and of the logic downstream from the flip-flop.

Figure 8



Routing data in to data out during scan test places RAM in transparent mode.

NONSCANNABLE ELEMENTS

The ASIC vendor and library you choose dictate the types of scan-equivalent cells you can design with. Because most vendor libraries have a rich variety of standard logic cells to choose, engineers typically write HDL code to produce sequential logic without paying much attention to the types of available cells. However, the scan-insertion tools pick flip-flops that come from a subset of this library. Currently, only one cell lacks vendor libraries when it comes to scan: flip-flops with both asynchronous set and asynchronous clear inputs. Therefore, avoid designing functions that require these types of cells. Scan flip-flops usually have either an asynchronous set or an asynchronous clear but not both.

MULTIPLE CLOCK DOMAINS

It is important to handle multiple clock domains with care. A clock domain is a grouping of sequential elements all tied to the same clock line. The same clock tree must generate this clock line. If two flip-flops use clocks that come from the same clock tree but a different branch of the tree, you still consider them within the same clock domain as long as you carefully watch your clock skew. However, if one flip-flop takes the clock directly from the clock tree and another flip-flop has to modify the clock via combinational logic, then the two flip-flops are in two clock domains. The only way you could consider these clocks as in the same clock domain is if you carefully watch the clock skew between them, assuming that they are next to each other in the scan chain.

Keeping an eye on clock skew is important because, although it's usually easy to meet setup-timing requirements during scan testing, due to slow scan-clock frequency, hold-time problems are common. As you internally route the blocks' scan chains—meaning that you route the scan chains at the block level and connect them at the chip level—hold-time problems aren't prevalent for flip-flops in the same clock domain. Timing problems usually occur between blocks, due to the distance between logically adjacent scan flip-flops, which causes excessive clock skew, and within blocks, in which clock-gating occurs. Any gating of the clock—whether for power savings or for multiplexing to handle issues with the second scan-test technique, for example—introduces skew in the clock line.

You can avoid potential hold-time problems by ensuring that a scan chain consists only of flip-flops from the same clock domain. If this situation is not feasible, then you should add lockup latches between the adjacent flip-flops on a scan chain that are in different clock domains. To avoid having too many lockup latches and a confusing scan-chain interconnect, you should analyze your designs beforehand to avoid any gating of clocks. If you can't avoid clock gating, then attempt to minimize it. Try to develop a scheme that localizes all of the clock gating so that these flip-flops use the same clock during scan mode, meaning they are within the same clock domain for scan purposes. Then, you can place the flip-flops together, and you need to place lockup latches only before the first and after the last flip-flop in this group.

CHIP-LEVEL-SCAN ISSUES

This final set of tips is simply a collection of issues that you need to think about at the chip level to ensure the correct handling of chip-level and block-level scan issues.

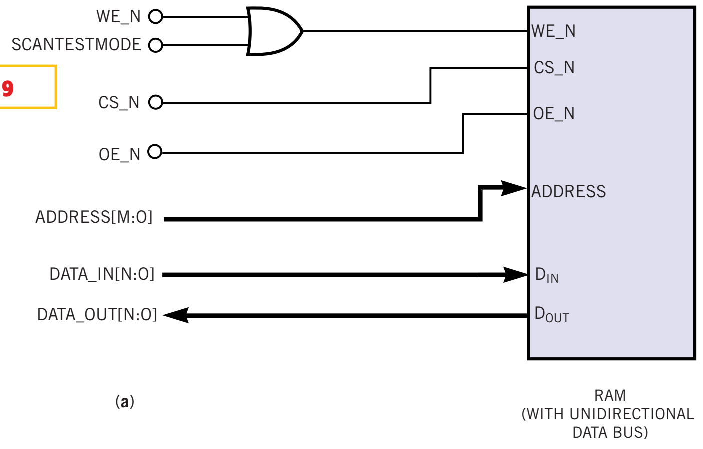
- Route the scan chains at the block level and connect them at the chip level. To avoid potential hold-time problems, consider using lockup latches at the chip level to hook up the scan chains between blocks.
- Dedicate pins to handle the scan chains using scan-enable, scan-input, scan-output, and scan-test-

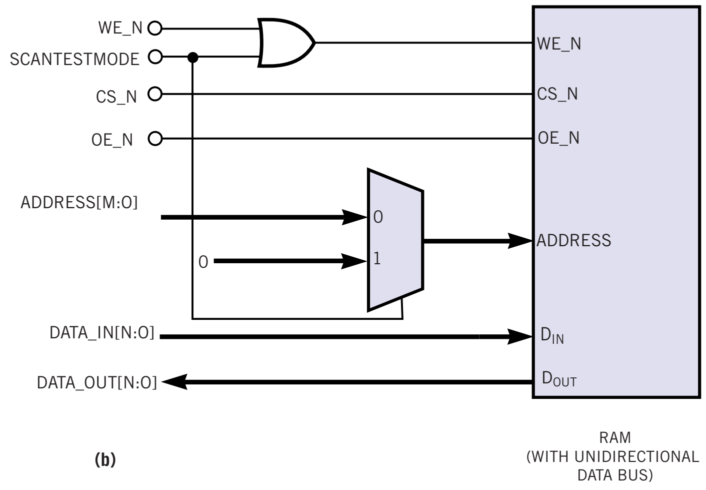
mode signals, or design logic to multiplex the scan pins with the normal functional I/O.

- Preplan all the various test modes you need and determine how to get the chip into those modes. You can either use spare pins or design in logic, such as test-access-port controllers (**Reference 1**). This test logic needs to at least generate a "scantestmode" signal to alert logic in the chip when scan-test mode is active. All of the various multiplex logic that this article mentions uses this signal to bypass nonscan blocks, to multiplex clock signals to clock pins of flip-flops, and to multiplex reset signals to set or clear pins of flip-flops.
- Buffer the scan-enable signal to provide the maximum scan-testing frequency. Remember that every flip-flop uses the scan-enable signal. If you're not careful, you could end up with a large ramp time on this signal. A few 4× drive buffers in parallel should be able to drive more than 20,000 gates to run the scan vectors at 20 MHz. If higher speeds or more flip-flops are involved, then a little more buffering is necessary. Although 1 MHz is a typical scan frequency, it may be necessary to increase the frequency, to 10 or 50 MHz, for example, for the production-test vectors to run in a reasonable amount of time. The test frequency depends on the complexity of design, how scan friendly the design is, and how many scan patterns are necessary to achieve the desired fault coverage. The design complexity limits the speed of the test. A higher number of scan patterns generally warrants a higher test frequency, so that the test time remains as short as possible.

Handle bidirectional I/O with care. Bidirectional I/Os can cause problems on testers, depending on how freely the ATPG tool chooses to operate them. Frequently, the default setting of the ATPG tool allows it to generate vectors in which the bidirectional I/Os change direction as a result of the capture clock. Production testers do not generally support this activity. To be safe, you should instruct the ATPG tool to generate scan patterns that do not change the direction of bidirectional I/Os as the result of a capture cycle or do not cause any contention as a result of the bidirectional I/Os changing to outputs. □

Figure 9





Another way to test surrounding logic is to preinitialize the entire RAM (a) or one location in RAM (b) and then use the contents to generate test data for the logic. You can use the "scantestmode" signal to disable the RAM's write signal during scan test.

REFERENCES

1. IEEE Standard 1149.1-1990, IEEE Standard Test Access Port and Boundary Scan Architecture, New York, IEEE, 1990.
2. "ASIC/IC Design-for-Test Process Guide," Mentor Graphics, www.mentorgraphics.com

AUTHORS' BIOGRAPHIES

Ken Jaramillo is a principal engineer at Philips Semiconductors, where he has worked for four years. He has worked as designer and architect of ASICs, FPGAs, and boards for products including avionics, high-speed serial buses, high-performance gaming platforms, PCI-bus-related products, high-performance PC audio, and high-speed networking products. He has a BSEE from the University of Missouri (Kansas City) and a BSCE from the University of Missouri (Columbia).

Subbu Meiyappan is a senior design engineer at VLSI Technology, a subsidiary of Philips Semiconductors. He has worked for the company for three years, designing, developing, synthesizing, simulating, and validating high-performance intellectual-property blocks for PCI, ARM-ASB-based devices, and high-performance ASICs. He has a BE from Annamalai University (Annamalai Nagar, India) and an MS from Tennessee Technological University (Cookeville, TN).