Channel Masking Synthesis for Efficient On-Chip Test Compression

Vivek Chickermane, Brian Foutz, and Brion Keller

{vivekc, foutz, kellerbl}@cadence.com
Cadence Design Systems, 1701 North Street, Endicott, NY 13760, USA

Abstract

The effectiveness of on-product Test Compression methods is degraded by the capture of unknown logic states ("X-states") by the scan elements. This paper describes a simple but cost-effective solution called channel masking that masks the X-states and allows test compression methods to be widely deployed on a variety of designs. It also discusses various aspects of the channel masking hardware and the synthesis and validation methodology to support its use in a typical design flow. Results are presented to show its effectiveness on some large industrial designs.

1. Introduction

The increasing use of on-product Test Compression logic [Ref: 1-11] to reduce the Cost of Test has led to an increased interest in the study of unknown circuit states and their impact on DFT. Unknown or X states occur in VLSI designs for a number of reasons such as:

- Inability to accurately model some logic modules for Automatic Test Pattern Generation (ATPG). The un-modeled logic or black boxes are common sources of X values.
- Internal three-state logic that could go to high-Z.
- Un-initialized non-scan flip-flops/latches that cannot be reset prior to scan test application.
- Embedded RAMs that are not written into before reading. This is commonly avoided by having the memory Built-In Self Test (BIST) wrapper provide a RAM bypass so that the RAM is not used during logic test; however, there are some advantages in writing through RAMs to obtain a better delay test of the RAM and surrounding logic.
- At-speed delay tests in which not all paths can meet the desired timings. In fact, whenever applying delay tests or path tests, it is common to find paths that do not meet functional timing. Common reasons for these timing exceptions are setup and/or hold-time violations, multi-cycle or non-functional paths, etc., that may lead to incorrect capture of test response during device testing. Some of these paths may, in fact, not be used functionally, but that does not prevent them from causing problems during test.

The last point above relating to delay testing is rapidly becoming a major concern. Many companies moving to 130 nanometer designs and below are seeing great benefits to performing some kind of delay testing to their chips [Ref: 14]. Since adding delay tests can easily result in a 2-5x or more increase in the test vector counts, this is a prime motivator for choosing to use logic test compression. It should be emphasized that delay testing exposes many paths that may not make the target timing. It works fine for functional use, but can cause trouble when applying ATPG generated tests at speed. It may be possible to slow down the tests so that they work for all paths, but this creates two problems: (1) someone must determine at what speed the tests can safely be run, and (2) the quality of the tests is reduced when they are run at a slower speed. Path tests are notorious for wanting to test a specific path at speed and to ignore responses from all other paths. This is the ultimate stress test for compression logic to deal with. ATPG tools need to be very flexible in dealing with X states in order to create reliable delay (path and/or transition fault) tests.

Another concern relating to unknown values comes up during post-silicon debug of scan tests. There are a number of reasons why tests may not work at the tester, including timing problems, yield issues and incorrect ATPG models. When scan tests produce variable responses because of one or more of these issues, the test engineers need the means to change the test patterns to eliminate the tester failures. One approach is for the test methodology to allow the ATPG tools to re-simulate the

 0-7803-8580-2/04 $20.00 Copyright 2004 IEEE

tests while specifying a specific measure bit or multiple measure bits that must be predicted as X. This "force-X" approach allows the tools to re-compute the test response and the fault coverage for the failing test patterns while ignoring the specified measure bits. This "force-X" method needs to be supported by any embedded test compression method.

2. Impact of X-states on Embedded Test Compression Hardware

The capture of X-states can greatly degrade the efficiency of on-product DFT methods that utilize response analysis methods such as Signature Analyzers. Logic BIST (LBIST), Weighted-Random patterns (WRP), Deterministic BIST, On-Product Multiple Input Signature Register (OPMISR), SmartBIST etc., are well known methods [Ref: 1-10] that employ signature analyzers and are susceptible to the X-state problem. MISRs are Linear Feedback Sift Register (LFSR) based signature analyzers that observe test responses on a scan-cycle by scan-cycle basis and sequentially update the current signature by XOR-ing it with the parallel scan unload data. An example where a MISR is used for signature analysis is shown in Figure 1 with an OPMISR+ scheme [Ref: 3].

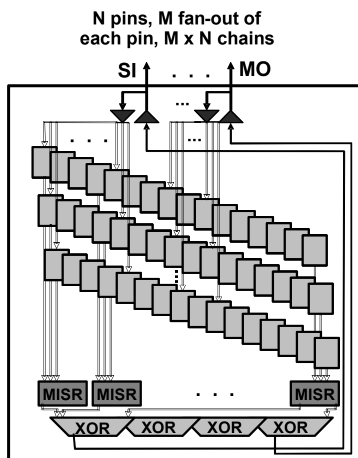

Figure 1: An OPMISR+ Implementation using multiple MISRs and XOR compactors

In this methodology, N scan-in pins are used. Each scan-in pin is fanned out to M internal scan chains. The NxM internal scan chains scan out into MISRs and are called "channels" for historical reasons [Ref: 4]. Each channel is associated with one MISR bit. Since a large multi-million gate design may have several hundred channels, multiple MISRs may be used. During test generation, the test response at the end of each system cycle is scanned into the MISR while concurrently the test stimuli for the next test is scanned in.

At the end of the scan operation, the MISR signature can be read from the parallel observation pins. Exclusive-OR (XOR) compactors known as MISR response compactors may be used to compact the MxN MISR signature bits down to N or fewer observation pins. An alternative implementation uses XOR compactors known as channel compactors between the channel outputs and MISR inputs.

In a MISR-based compaction scheme, the presence of a single X-state in any one of the scan channels will invalidate the MISR signature once it is captured. Every signature captured after that scan cycle will be unpredictable and the results will have to be excluded from the test coverage calculations. Surprisingly, compression schemes that do not rely on a MISR can also have problems with X states. The typical compression of scan channels to a smaller set of scan-out pins known as space compaction uses an XOR tree to compact all the scan channels to a smaller number of observable pins. For example in Figure 1 a space compactor is used between the MISR outputs and the chip observation outputs. If any one channel outputs an X, it is impossible to see the response from any of the other channels it is XORed with. It may take just a few channels to output a lot of X values to cause massive amounts of X values to appear at the observable pins. A scheme has been proposed in [Ref: 12] that sends each scan channel output to multiple XOR trees hoping that at least one of those trees will not see an X from any of the channels that feed it. This will work for relatively sparse X streams but causes a significant wiring congestion problem when there are 1000 or more channels needing to be XORed into multiple XOR trees.

There are two methods that can be used to address the issue of X-states in the context of embedded test compression methods

- X-suppression relies on the addition of control points that ensure that unknown values are never captured by the scan flip-flops. An example of widespread use of this is in the IBM DFT methodology [Ref: 2] to support WRP and LBIST techniques. This approach, while effective, requires a very disciplined design methodology and rules checking and does not permit late engineering changes and variability in the test environment.
- X-tolerance allows the X-states to be captured into the scan flip-flops but employs additional on-chip hardware and intelligent ATPG to ensure that the test results are not corrupted by the X-capture. The work in [Ref: 11-12] is an example where X-tolerance is supported for XOR-compactors, while the work in [Ref: 13] supports MISR-based compaction.

This paper will focus on on-chip hardware that supports X-tolerance by borrowing some ideas from X-supression for on-product signature analyzers such as MISRs and XORs or some combination of them. One key difference between this and previous work is that the proposed scheme is applicable to any general test compaction scheme and has very low hardware overhead. It is also very scalable in that the masking configuration can be adjusted to account for the number of captured X-sources in a given channel**.**

3. Channel Masking Logic

As discussed above, the capture of X-states into the scan elements degrades the ability of the MISR to gather a stable signature. The on-chip hardware to support X-tolerance in this work has been borrowed from a common debug method used for LBIST called channel masking [Ref: 15-16]. Every scan channel in a practical LBIST scheme is designed so that it can be masked by gating logic which ensures that the corresponding MISR bit always scans in a constant-0 value. The purpose of this channel masking hardware is to allow the test engineer some flexibility in deciding which channels should be observed in a given test session and which ones should be blocked. During LBIST debug, the test engineer can program the mask register so that only the logic module of interest will be observed during LBIST. This method has several advantages such as simplicity, low hardware overhead and the ability to make trade-offs between the complexity of the test software vs. complexity of the hardware.

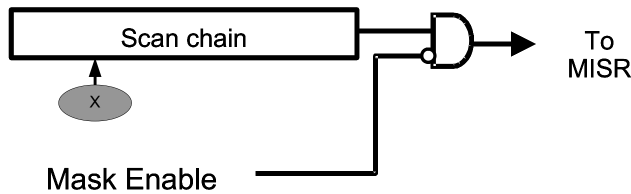

Figure 2: WIDE0 Channel Masking with a Single Global Signal

Figure 2 shows the simplest implementation, called a WIDE0 Channel Mask. Each channel tail is gated by a global Mask Enable signal (which is a Test Primary Input) before it feeds the MISR. The single Mask Enable signal gates all the channels simultaneously. The ATPG can block a scanned out value on a scan-cycle by scan-cycle basis. Figure 3 shows three scan Channels carrying test responses including some X states feeding a MISR via a WIDE0 masking scheme. The Mask Enable data that is stored on the ATE is juxtaposed with the scan chain data to illustrate its working. Every scan bit position that carries an X needs a 1 value in the mask data to enforce the masking. This is shown by the shaded regions. For example during the first shift into the MISR the X in Channel 1 is likely to get captured by the MISR. This can be prevented by setting the mask enable data to 1 for the first bit position from the right. However any non-X response data that happens to be in the same bit position is also masked such as the 0 values in Channels 0 and 2 in the right-most bit position. Furthermore Channel 2 that captures only non-X data will lose 7 valid response bits that may be detecting faulty responses. When the mask enable is asserted to block an X from a channel any valid response bits that are also blocked from other channels are considered to be *over-masked*. This example shows one limitation of the simple masking approach: all the channels are blocked simultaneously, thereby over-masking many more bits than would otherwise be required and thus reducing the quality of the test.

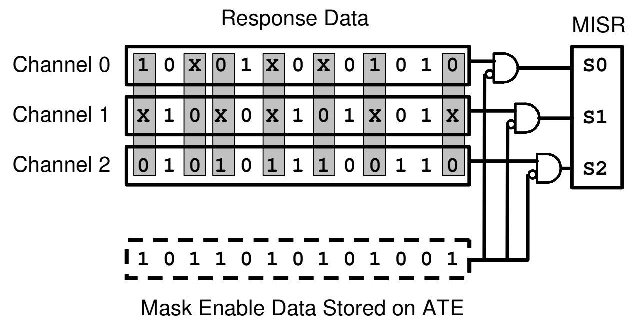

Figure 3: An example illustrating WIDE0 masking

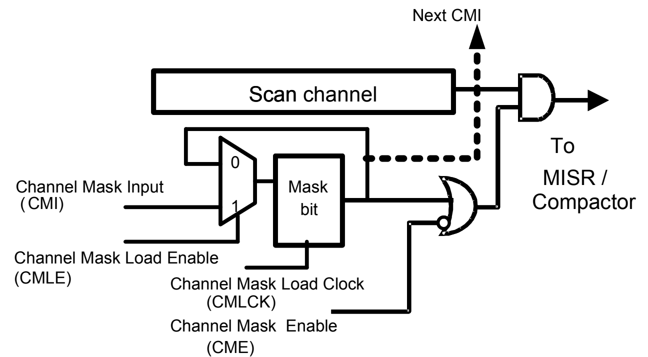

Figure 4: Channel Masking with one Mask Register bit per channel (WIDE1 Masking)

The next step up in quality can be achieved by reducing over-masking with the addition of one mask register bit per channel. This implementation is called a WIDE1 channel mask and is shown in Figure 4. The mask bits can be pre-loaded via one or more Channel Mask

Input (CMI) pins when the Channel Mask Load Enable (CMLE) pin is asserted. The ATPG is given the means to automatically mask out selected channels on a scan cycle by scan cycle basis. A global Channel Mask Enable (CME) pin determines if the mask bits should be applied to the current scan cycle. The mask bit is a storage element with a clock CMLCK which can be shared with other test clocks.

4. ATPG Process

The test generation and application process is enhanced to support the channel masking hardware as follows:

```
For the i'th test {
    (1) Assert the Channel Mask Load Enable;
    (2) Scan in the mask data for the (i-1)'th test response via the CMI pin(s). The Mask bit for a channel is loaded with 0 if any bits of that channel have captured an X. The Mask bit for an X-free channel is loaded with a logic value 1;
    (3) De-Assert the Channel Mask Load Enable;
    (4) Assert the chip Scan Enable to enter the scan state;
    (5) Load the scan channels with the scan load data for test (i) as determined by ATPG; Concurrently the test response for the (i-1)'th test is being analyzed and compacted by the MISR;
    (6) At each scan cycle if an X from test response (i-1) is about to be scanned into the MISR then block it by asserting the Mask Enable;
    (7) Measure the MISR signature from the parallel observe PO's;
    (8) Reset the MISR;
    (9) De-assert the scan enable and exit the scan state;
    (10) Set the PI values as determined by the ATPG for test (i);
    (11) Run the system cycles to launch test data and capture the test response;
    (12) Measure the PO response;
    (13) Set (i) to (i+1);
}
```

Several points need to be highlighted based on the above algorithm.

- The mask_enable signal will operate as if it was a scan input, as far as the tester interface is concerned. This means that mask_enable signals will consume ATE scan pin resources and thus will reduce the number of scan pins available for loading test data into the design – i.e., the masking-per-cycle information is considered to be additional test data.
- If test response *(i-1)* leads to an X being captured into channel *j*, then the mask register value for channel *j* will be a 0 in the *i'th* test.
- The converse of the above is for a channel *j* that is X-free. The mask bit will be at a 1 and therefore all the captured responses in channel *j* are guaranteed to be analyzed by the MISR, regardless of the X's captured by the other channels.
- If the X value is captured in position *p* from the tail of channel *j*, then the mask enable signal will be at 1 during the *p'th* scan cycle.

The diagram in Figure 5 below illustrates the above points. Channel 2 is X-free; its mask register will be pre-loaded to a value 1 during the channel mask load state. Its contents can never be masked out and are safely transported to the MISR.

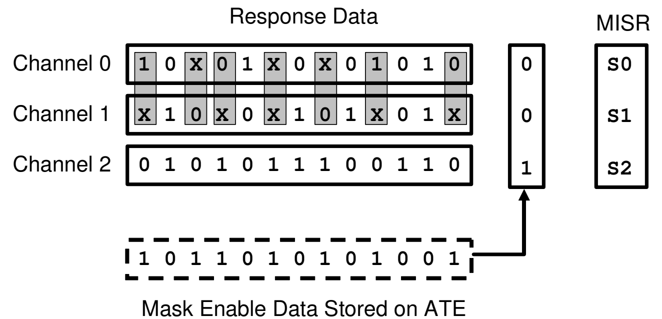

Figure 5: An example illustrating WIDE1 masking

Consider now Channel 0 and Channel 1, both of which captured an X. Both their mask registers have to be pre-loaded with a 0. When the X from Channel 1 is at the tail position and about to be scanned into the MISR, the mask enable is asserted to mask the X. This allows the valid data in Channel 2 to pass into the MISR. The cases where over-masking occurs is shown by the shaded bit positions in Figure 5. There are 6 over-masked bits in this example when compared to the 13 over-masked bits in Figure 3. This scheme over-masks the legitimate test response value 0 in Channel 0 and prevents it from being scanned into the MISR during the first shift. Likewise at the fourth shift the X in Channel 1 has to be blocked; this will mask the legitimate test response in Channel 0. This simple example illustrates the trade-off. With the addition of 1 mask bit per Channel the over-masking has been reduced considerably but there is still some room for improvement.

5. Upgrading to a WIDE2 Channel Mask

The next iterative improvement in the masking logic is to have two mask bits per channel. This implementation, known as WIDE2, is shown below in Figure 6. This utilizes two mask-bits per channel: R0 and R1. It also utilizes two streams of mask enable data stored off-chip and controlled by the two mask enable signals CME0 and CME1. Cumulatively this provides four possible states: 3 channel masking states and one non-masking state as shown in Table 1. Each channel's masking state can be classified into four categories:

1. Never masked. By pre-loading the mask bits to be R0=1 and R1=1 this channel is always observable unless overridden by the All-Masking state.
2. R0-maskable. By pre-loading R0=0 and R1=1 this channel is masked whenever CME0=1 and CME1=0
3. R1-maskable. By pre-loading R0=1 and R1=0 this channel is masked whenever CME0=0 and CME1=1
4. R0R1-maskable. By pre-loading R0 and R1 both to 0, this channel is masked on any masking cycle.

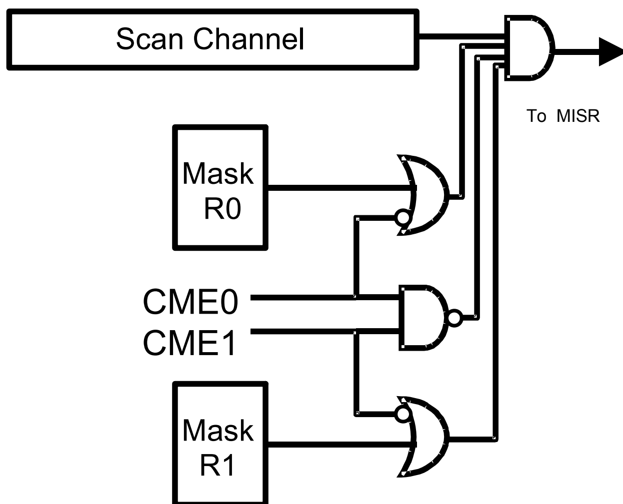

Figure 6: WIDE2 channel mask

Table 1: Four channel masking states using Mask bits R0 and R1 and 2 Channel Mask Enable pins

CME0	CME1	Mask Bit(s)
0	0	None
1	0	R0
0	1	R1
1	1	All bits

Figure 7 shows an application of WIDE2 masking on the same test response stream as above. Channel 2 is X-free and is assigned the mask state 11. Channel 0 is assigned to be R0-maskable while Channel 1 in the middle is R1-maskable. When an X is at the tail position of Channel 1, at the next scan cycle the Mask R1 configuration permits only Channel 1 to be masked while the other two channels are not affected. The ATPG has the flexibility to map any X-capturing channel to be R0 or R1 or R0R1 maskable. We see that the WIDE2 scheme is superior to WIDE1 for the bit positions 1, 4, 6, 10, 11, and 13 since no over-masking occurs. However over-masking does occur at position 8 where valid response data in Channel 2 may be over-masked. WIDE2 works the best when the captured X's are well distributed along the length of a channel.

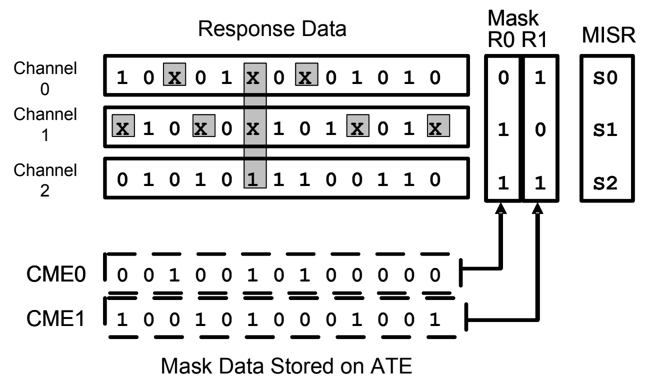

Figure 7: An example illustrating WIDE2 masking

One could conceivably implement a WIDE3 or higher masking scheme to even further reduce over-masking. However the hardware overhead and the fact that over-masking is not always going to lead to loss of test-coverage make this impractical. A more pragmatic scheme would be to selectively use WIDE0, WIDE1 or WIDE2 or no masking based on the analyses of the X-responses captured by the different channels. For example, in Channel 2 if no masking were used then the over-masking would be reduced to 0.

6. Channel Mask Synthesis Considerations

The WIDE0, WIDE1 and WIDE2 mask types can be deployed in a number of ways. If the design is very carefully implemented to ensure that X states seldom occur, then the WIDE0 type could be used for all the channels. This would provide an adequate safety mechanism in the unlikely event that an X value is captured. If X states are likely to occur sporadically affecting less than 1% of the scan bits in a channel then

the WIDE1 implementation can be used for all the channels. The WIDE2 mask type is a good solution when the number of X-states is likely to exceed 1%. Experiments on several industrial designs have shown that WIDE1 is a good default for most designs while WIDE2 can be selectively deployed on designs once it is determined that the X-states are prolific. Since the focus of this paper is on the presentation of the concept of channel masking and its implementation, a comprehensive comparison of the three schemes and experimental data will be the subject of a future paper.

The adoption of synthesis methods permits the deployment of a whole family of channel mask configurations. For example, some channels could be assigned WIDE2 mask types, while others could be assigned WIDE1 mask types and the remainder assigned WIDE0. An even more effective scheme would be to share a mask type such as WIDE2 with multiple channels. We will now discuss a simple methodology to obtain such a configuration. The analysis starts on a design where the channels have already been created based on physical floor-planning info. X-state analysis is run to determine the X-sources in the design and forward propagate them to scan elements. Calculate the number of X-corruptible scan elements in each channel.

1. For any channel that has no X-corruptible scan elements, assign a WIDE0 mask. This allows the minimum amount of safety to support post-silicon test pattern debug with minimal hardware overhead.
2. For any channel that has greater than 0 but fewer than 1% X-corruptible bits assign:
    a. If a physically neighboring channel also has greater than 0 but fewer than 1% X-corruptible bits then share a WIDE2 mask with that neighbor
    b. If there is no such neighboring channel, assign it a WIDE1 mask
3. For any channel that has greater than 1% X-corruptible bits assign it a dedicated WIDE2 mask

The above configuration can be updated for a more conservative design methodology to use the following mask mapping: use WIDE1 for (1) and dedicated WIDE2 for all others. The strategy can be easily tuned based on user experience. Once the channel mask configuration has been determined the implementation requires a few other design considerations:

1. Channel Mask Load Enable (CMLE) is very similar to a scan enable for the channel mask register and is enabled during the channel mask load state. It cannot be shared with any other test input for the test compression mode but it can be shared with a functional pin using the proper pin sharing logic.
2. The Channel Mask Load Clock (CMLCK) clock loads the mask register flip-flops and can be shared with a test clock such as a MISR clock. In any test compression mode, CMLCK will shift only in the channel mask state so it has to be kept mutually exclusive from the shift clock for the logic under test and the MISR shift clock. This can be achieved by gating any common test clock with CMLE. CMLCK can be shared with any system functional pin with the use of appropriate pin sharing logic.
3. Channel Mask Input (CMI) is similar to a scan-in pin that is used to serially load one or more mask bits. By default CMI pins are shared with system scan-in pins. This is possible since the channel mask load state does not overlap with the normal system scan operation. The mask bits are configured into mask shift registers with the CMI serving as the scan in pin. In Figure 8, one can have N mask shift registers each with M shift stages. Each of the N scan-in pins serves as a CMI.
4. As can be seen from Figure 8 the scan-in pins are connected internally to a test bus. This test bus is used to load test stimuli; it is used to load mask register data; and the bus is also used to unload MISR signatures.

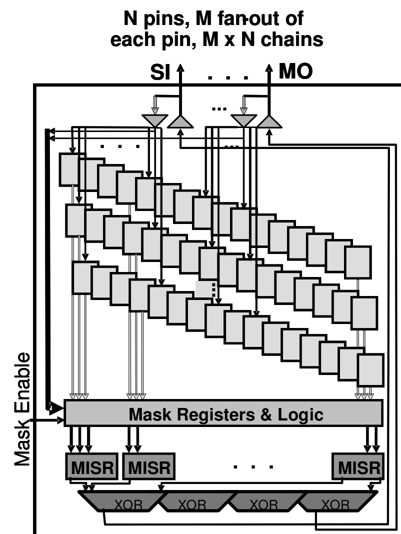

Figure 8: An OPMISR+ Scan Configuration with Channel Masking Logic

7. Experimental Results

In this section, results on a 7M gate industrial design with 250K scan flip-flops are shown to demonstrate the

effectiveness of channel masking. This design was chosen specifically since it was deemed hard-to-test, requiring over 19 CPU hours to achieve close to 99% fault coverage and over 62 CPU hours to exceed 99.5% fault coverage. Furthermore, this design had many unmodeled embedded mixed-signal and memory cores and X-suppression was not feasible due to design considerations. An OPMISR+ methodology similar to the one shown in Figure 8 above was selected to reduce test data volume and test application time.

Table 2 summarizes the results obtained when using a low ATPG effort. The original design supported 8 scan chains with 27,000 scan bits per scan chain and required 7000 test patterns to achieve 98.93% fault coverage in 19 hrs and 30 mins. Column 5 describes the number of response bits that were masked for the compression modes using WIDE1 masking, while column 6 shows the magnitude of test data that would have to be stored on the ATE. Column 7 shows the test application time as a percentage the time required for regular full scan. A number of different test compression schemes all generating 7000 patterns were experimented with to study the reduction in test data volume and test application time. Furthermore these experiments were used to evaluate the effectiveness of the channel masking hardware using the suggested configuration in the previous section. In the OPMISR configuration 16 channels were used while the OPMISR+ schemes used 128, 512, and 1024 channels. The results show that in all cases the test data volume and test application time reductions obtained were equal to or better than the theoretical predictions, while the fault coverage was very close to the full scan results. This was true despite the large number of x-states captured that ranged from 150K bits in the OPMISR case to 688K in the OPMISR+ with 128 channels. This experiment shows that the channel masking approach described in this work is very effective and provides test compression results that match expected results.

Table 2: ATPG results using different scan configurations and “Low” ATPG effort

Mode	Scan Chains/ Channels	Pattern Count	Max Scan Bits	#Bits masked	Test Data Volume / (%)	Test Time Percentage	Fault Coverage	CPU Time – hr:min
Full Scan	**8**	**7000**	**27000**	**NA**	**5103M (100%)**	**100%**	**98.93**	**19:30**
OPMISR	**16**	**7000**	**13500**	**150K**	**1607M (31.5%)**	**50%**	**99.04**	**18:18**
OPMISR+	**128**	**7000**	**1687**	**688K**	**201.7M (3.95%)**	**6.3%**	**98.96**	**17:30**
OPMISR+	**512**	**7000**	**422**	**220K**	**54M (1%)**	**1.7%**	**98.75**	**22:30**
OPMISR+	**1024**	**7000**	**211**	**354K**	**32.7M (0.6%)**	**1%**	**98.84**	**22:55**

A second experiment was run using a high ATPG effort to achieve the highest possible fault coverage to see if there would be any negative impact due to masking when test compression was used. The results are presented in Table 3. The regular scan mode achieved 99.55% fault coverage in over 62 CPU hours with 7173 test patterns. In phase two an incremental approach was used to achieve the highest fault coverage. The inserted test compression hardware supported three test modes: OPMISR+ with 1024 channels, OPMISR with 16 channels, and regular scan with 8 scan chains. First the opmisr+ mode which has the highest test compression effectiveness was run until the coverage flattened out. The test data was committed and the remaining faults were tested using the OPMISR test mode until the coverage flattened out. Finally, any remaining faults from the OPMISR mode were tested in the regular full scan mode. The fault coverage was thereby incrementally increased from 99.54% to 99.68% to 99.85%. This scheme brought about the best fault coverage with a total CPU usage of about 29 hrs while reducing test data volume to less than 9% of the original test data volume and 11% of the original test application time. Here again we see that the large number of X-states masked (over 6.9M in the OPMISR+ case) had

no appreciable negative impact on the overall results obtained. The channel masking and test compression hardware overhead was less than 0.7% which is very insignificant when compared with the high fault coverage and compaction that was achieved on a very hard-to-test design.

Table 3: Incremental ATPG results using "Max" ATPG effort

Mode	Scan Chains/ Channels	Pattern Count	Max Scan Bits	#Bits masked	Test Data Volume / (%)	Test Time Percentage	Fault Coverage	CPU Time – hr:min
Full Scan	**8**	**7173**	**27000**	**NA**	**5229M / 100% (base)**	**100% (base)**	**99.55%**	**62:48**
OPMISR+	**1024**	**5400**	**211**	**6990K**	**25.2M / 0.5%**	**0.76%**	**99.54%**	**25:50**
OPMISR	**16**	**686**	**13500**	**330K**	**157M / 3.0%**	**4.78%**	**99.68%**	**1:49**
Full Scan	**8**	**391**	**27000**	**NA**	**285M / 5.4%**	**5.45%**	**99.85%**	**1:24**

The third experiment that was run used three different industrial circuits named Alpha, Eta, and Tau. These are smaller designs than used before and are in the 700K-1M gate range with the scannable flip-flop count in the 25K-50K range. Each design was built with four different DFT configurations: regular full scan with 8 scan chains and 3 OPMISR+ test compression configurations using WIDE0, WIDE1, and WIDE2 masking respectively. Alpha and Eta used 256 scan channels for compression (32x reduction in test application time) while Tau used 128 channels (16x reduction in test application time). Table 4 summarizes the results obtained. Column 2 shows the scan/masking configuration for each design: full scan with no masking, and WIDE0, WIDE1, and WIDE2 masking. Column 3 shows the number of scan-in/scan-out/channel mask enable test pins and column 4 shows the ATPG vector count. Column 5 has the length of the longest scan chain/channel and also the length of the longest channel mask register that has to be loaded from the CMI pin which is only applicable for the WIDE1 and WIDE2 modes. Column 6 shows the number of patterns for which the mask register had to be loaded while column 7 shows the total number of mask bits in the design. Column 8 shows the number of response bits in the ATPG patterns that had to be masked. For the full scan design this column represents the number of X values in the response data. Column 9 shows the test coverage obtained.

The Alpha design demonstrated the most interesting results mainly due to the large number of X's that were captured even in the full scan mode. The WIDE0 configuration had a very large number of response bits masked in the patterns generated (93.4%). By stepping up to WIDE1 and then WIDE2 masking the number of bits masked fell significantly. The number of test patterns fell from 8490 for WIDE0 to 7051 for WIDE1 and eventually 4331 for WIDE2. The full scan pattern count is at 3960 but this has to applied with a configuration where the longest scan chain is 5632, while the OPMISR+ cases have 32x shorter maximum scan channel length.

The Eta and Tau designs are better from a DFT perspective, with fewer X-states which impact very few of the scan channels. As a result, there is no significant reduction in test patterns when going from WIDE0 to WIDE1 and WIDE2 and equivalent test coverage is obtained with just a small reduction in patterns. However, one does observe a reduction in the number of bits masked. Presumably the over-masking did not impact fault detection to a level that required significant additional ATPG effort.

The results from Table 4 highlight two important points. The first is that for designs with a significant amount of X-states in the response data a scheme such as WIDE2 is vital to achieve the desired test coverage with a high degree of compression. For more carefully designed circuits with few X-states a scheme such as WIDE0 or WIDE1 is sufficient. Overall a scalable and flexible masking scheme such as presented in the previous section provides the best results with the lowest overheads.

8. Conclusions

This paper described a simple but cost-effective solution called channel masking that masks the X-states and allows test compression methods to be widely deployed on a variety of designs. The channel mask hardware is very simple and three different mask types were presented that can be deployed in a flexible manner. Results were presented on a hard-to-test industrial design to show its effectiveness using multiple test modes and ATPG effort levels to show that test compression can continue to be effective when coupled with the proposed channel masking hardware scheme.

Acknowledgements

The authors would like to thank C.V. Krishna and other colleagues at Test Design Automation for their help in supporting this work and their careful review of this paper.

References

[1] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan and J. Rajski, "Logic BIST for Large Industrial Designs: Real Issues and Case Studies," Proc. International Test Conference, pp. 358–367,

[2] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller and B. Koenemann, "OPMISR: The Foundation for Compressed ATPG Vectors," Proc. International Test Conference, pp. 748–757, 2001.

[3] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, A. Ferko, B. Keller, D. Scott, T. Onodera and B. Koenemann, "Extending OPMISR beyond 10x Scan Test Efficiency," Design & Test of Computers, IEEE, pp. 65–73, Sept–Oct 2002.

[4] P.H. Bardell and W.H. McAnney, "Self-Testing of Multi-Chip Logic Modules," Proc. International Test Conference, pp. 200-204, 1982.

[5] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, D. Wheater, "A SmartBIST Variant with Guaranteed Encoding," Proc. Asian Test Symposium, pp. 325–330, 2001.

[6] I. Bayraktaroglu and A. Orailoglu, "Test Volume and Application Time Reduction Through Scan Chain Concealment," Proc. Design Automation Conference, pp. 151-161, 2001.

[7] K. Lee, J. Chen, and C. Huang, "Using a Single Input to Support Multiple Scan Chains," Proc. of the Int. Conf. on Computer-Aided Design, pp. 74–78, November 1998.

[8] I. Hamzaoglu and J. H. Patel, "Reducing Test Application Time for Full-Scan Embedded Cores," Digest of Papers, 29th International Symposium on Fault-Tolerant Computing, pp. 260–267, June 1999.

[9] F. Hsu, K. Butler and J. Patel, "A Case Study on the Implementation of the Illinois Scan Architecture," Proc. International Test Conference, pp. 538–547, 2001

[10] B. Koenemann, C. Barnhart, B. Keller, "Real-Time Decoder for Scan Test Patterns," US Patent 6,611,933, Filed April 2000, approved August 2003.

[11] J. Rajski, et. al., "Embedded Deterministic Test for Low-Cost Manufacturing Test", Proc. International Test Conference, pp. 301-310, 2002

[12] S. Mitra, K.S. Kim, "X-Compact: An Efficient Response Compaction Technique for Test Cost Reduction", Proc. International Test Conference, pp. 311-320, 2002

[13] P. Wohl, J.A. Waicukauski, S. Patel, M. Amin, "X-Tolerant Compression and Application of Scan-ATPG patterns in a BIST architecture", Proc. International Test Conference, pp. 727-736, 2003

[14] B. Keller, M. Tegethoff, T. Bartenstein, V. Chickermane, "An Economic Analysis and ROI Model for Nanometer Test", Proc. International Test Conference, 2004

[15] P. Gallagher, V. Chickermane, and S. Gregor, "A Building Block BIST Methodology for SoC Designs: A Case Study", Proc. International Test Conference, pp. 111-120, 2001

[16] B. Keller and T. Snethen, "Built-In Self-Test Support in the IBM Engineering Design System", IBM Journel of Rsearch and Development, pp. 405-415, March 1990

Table 4: Comparison of Masking Schemes for Three Different Designs

Design	Mode	SI/SO/ CME Pins	Pattern Count	Max Scan Bits / +mask	Number of channel mask loads	Mask bits	#bits masked (percentage of total data)	Fault Coverage
Alpha	Full Scan	8/8/0	3960	5632/-	NA	NA	3.5M (2.0%)	99.22%
	Wide0	16/0/1	8490	175/0	0	0	355.9M (93.4%)	98.94%
	Wide1	16/0/1	7051	175/16	6699	256	28.2M (8.9%)	98.97%
	Wide2	16/0/2	4331	175/32	4009	512	16.5M (8.5%)	98.97%
Eta	Full Scan	8/8/0	2854	4319/-	NA	NA	44.9K (0.001%)	99.66%
	Wide0	16/0/1	3427	135/0	0	0	15.5M (13%)	99.42%
	Wide1	16/0/1	3406	135/16	1631	256	212.4K (0.18%)	99.39%
	Wide2	16/0/2	3370	135/32	1609	512	211.6K (0.18%)	99.36%
Tau	Full Scan	8/8/0	5579	2784/-	NA	NA	129.8K (0.1%)	98.64%
	Wide0	16/0/1	5797	174/0	0	0	17.9M (13.9%)	98.39%
	Wide1	16/0/1	5760	174/8	3053	128	521.5K (0.4%)	98.37%
	Wide2	16/0/2	5738	174/16	3011	256	477.9K (0.37%)	98.40%