Adding Testability to an Asynchronous Interconnect for GALS SoC

Aristides Efthymiou
a.efthymiou@cs.man.ac.uk

John Bainbridge
jbainbridge@ieee.org

Douglas A. Edwards
d.edwards@cs.man.ac.uk

Department of Computer Science,
University of Manchester,
Oxford Road M13 9PL, UK

Abstract

Asynchronous circuits offer great potential for solving the interconnect problems faced by system-on-chip designers, but their adoption has been held back by a lack of methodology and support for fabrication testing of such circuits. This paper addresses this problem using a partial scan approach which achieves a test coverage of 99.5% on the CHAIN network-on-chip interconnect fabric which is used as an example. Test patterns are generated by a custom program automatically, given the topology of the interconnect. In comparison to standard, asynchronous, full-scan LSSD methods, area savings in the order of 50% are noted.

1 Introduction

The design of asynchronous circuits has been attracting more interest recently, as clock distribution on a large die becomes increasingly dif£cult. The ITRS road-map predicts that, as a solution to the clock distribution problem, Globally-asynchronous, Locally-synchronous (GALS) systems will become mainstream in the near future. In a GALS system, a number of synchronous islands of logic communicate asynchronously using a suitable interconnect. Unfortunately, the testability of asynchronous systems is considered one of their major drawbacks. This is a vital issue in enabling industrial adoption of this technology.

This paper describes novel design-for-testability and test-pattern generation techniques developed to produce a fully-testable version of CHAIN, an asynchronous SoC interconnect fabric [1]. The standard single stuck-at fault model is used and our method has been implemented in a computer program that, given the topology of a CHAIN interconnect, produces a sequence of test patterns that achieves nearly 100% fault coverage.

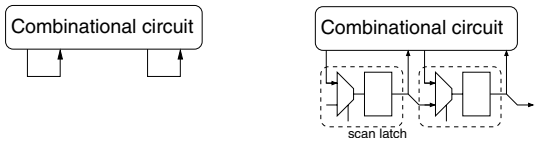
As asynchronous circuits resemble combinational circuits with feedback loops, previously developed techniques [2] insert scan-latches to break the feedback loops when the circuit is under test, as shown in £gure 1. Level sensitive scan design (LSSD) is used so that the asynchronous mode of operation is retained when both master and slave latches are transparent. Using this DfT logic, the circuit may be tested using conventional ATPG practices, as, for testing purposes, it possesses synchronous operation. Although this method does work, inserting scan latches at every feedback path present in the circuit incurs a signi£cant area overhead.

In this paper we exploit the regular topology of CHAIN interconnect circuits and a useful property of the C gate test patterns to produce an elegant solution to test pattern generation while using an area-saving, partial-scan method.

2 The CHAIN asynchronous interconnection

CHAIN is an architecture for system-on-chip interconnect using delay insensitive signalling and a message passing protocol. Connections are built from gangs of narrow signalling channels, each with its own acknowledge. Since completion detection over these narrow channels only requires the use of simple circuits and avoids the large trees of C gates found in conventional, wide datapaths, the links can operate at much higher speeds than the equivalent wide-datapath would be capable of.

The £rst CHAIN implementation uses a simple 1-hot code for the data encoding, with 5 forward going signals in each channel allowing either two-bits of data or an end-of-



(a) Asynchronous circuit (b) 'Scanned' equivalent

Figure 1. Scan latch insertion.

Proceedings of the 13th Asian Test Symposium (ATS 2004)
0-7695-2235-1/04 $20.00 © 2004 IEEE

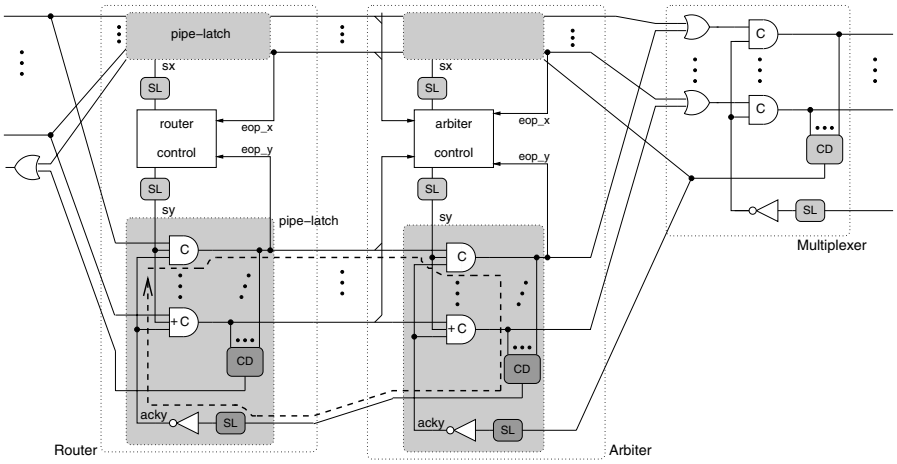


Figure 2. The CHAIN interconnect.

packet marker to be transmitted per communication handshake. Two such channels were used for each of the forward (command) and reverse (response) directions such that messages were transmitted over the CHAIN network serially, 4-bits at a time.

Fig. 2 shows the main elements of the CHAIN fabric: the router, arbiter, and multiplexer. All these blocks contain variations of the same circuit, a delay-insensitive pipeline latch, which stores and forwards the transmitted data.

Apart from the 'local' feedback loops inside the C gates, other feedback loops are also found in asynchronous circuits. Such a global loop, spanning two pipeline latches, is illustrated in Fig. 2. The blocks called *SL* indicate where scan latches are inserted to break these loops when the circuit is tested.

As a partial-scan strategy is employed (i.e. the C gates are not scanned), the state of the circuit has to be maintained from the application of one pattern to the next. Thus a second, parallel, slave latch is added to keep the state of the asynchronous circuit unchanged while patterns are being scanned in. Compared to standard scan latches used in asynchronous circuits [2], these are approximately 50% larger. This is the price paid for using a partial scan approach, but it is justi£ed by the lower total number of scan-latches required, which is approximately N times lower than in a full-scan approach for a CHAIN M-of-N pipeline latch.

During testing, *routes* from all transmitters to all receivers are set up by scanning in appropriate values to the scan-latches in the router and arbiter blocks; then patterns are transmitted from the source interconnection ports, while the output ports and the scanned acknowledge signals are observed. Depending on the interconnection topology, it is desirable to test as many routes as possible in parallel to minimize the testing time. Moreover, since some parts of the interconnection are shared among different routes, not all route combinations between transmitters-receivers need to be tested, as long as every pipeline latch is tested.

For each route, the interconnection is £rst tested by applying patterns aimed to expose stuck-at faults in the C gates of the pipeline latches, including those inside routers and arbiters. The second phase targets faults in the completion detection blocks of the pipeline latches. Finally, the control parts of the routing components are tested.

3 Testing the C-gate network

Achieving high test coverage requires generating sequential test patterns for asynchronous logic gates. In our experience conventional ATPG tools are unable to generate patterns with a high coverage for these circuits. We have developed a general method for generating patterns that yields 100% fault coverage for all types of C gates which is also applicable to other sequential gates.

All the parallel C gates inside the pipeline latches exhibit the same connectivity: an 'independent' input (a) coming from the upstream pipeline latch, one or two 'common' inputs from the scan-latches (b for the acknowledge signal, c for route select: sx or sy) and their outputs become the 'a-inputs' of the next stage. The intention is to use the same patterns for all parallel C gates in a pipeline latch, thus testing the same faults in all C gates simultaneously. This will also help to keep the number of patterns and the test time to a minimum.

Only three types of C gates can be found in a CHAIN interconnection: 2-input C gates (CM2), 3-input C gates (CM3), and 3-input asymmetric C gates (CM21) of which one input (marked with a '+') is active only for rising (output) transitions. Since all the C gates in a pipeline latch are intended to be tested in parallel using the same patterns, it is essential that the same patterns can be used for the CM3

Table 1. C-gate patterns.

CM2			CM3 (CM21)				
a	*b*	*q*	*a*	*b*	*c*	*q*	
1	1	1	1	1	1	1	global
1	0	1	1	0	0	1	
0	0	0	0	0	0	0	
0	1	0	0	1	1	0	
1	1	1	1	1	1	1	
0	1	1	0	1	0	1	local
0	0	0	0	0	0	0	
1	0	0	1	0	1	0	
			0	0	0	0	
			1	1	0	0	
			1	1	1	1	
			0	0	1	1(0)	

and CM21 gates, which are simultaneously present in the same pipeline latches.

Our experiments showed that the faults in CM21 are covered by patterns generated for CM3, i.e. the patterns for CM3 are a superset of those for CM21. As can be seen in table 1, their only difference is that the pattern {a,b,c} = {0,0,1} with a previous output of 1 generates a 0 for CM21 (c is the '+ input') while it remains 1 for CM3. This difference complicates pattern generation slightly, as an 'all-clear' pattern needs to be added to reset all the C gate outputs to the same value. Although this feature was not exploited here, it is interesting to notice that ignoring the third input (c), the test patterns for 2-input C gates are also included in those for the 3-input C gates.

3.1 Test patterns for the C-gate network

The sequences of patterns for the C gates are shown in table 1. Many of the patterns common to all three types of C gates have the property that the produced gate output value is the same as that of their 'independent' input a. These are the top £ve patterns in table 1. The importance of these patterns is that they can be used to exercise all the pipeline latches along a link route (from an initiator to a target) simultaneously, since the same values are generated at the a-inputs of all the C gates in all the latches in the link route. This saves a signi£cant amount of test time. As they are used for the whole link route, these patterns are called 'global patterns' in this paper. Out of the many possible ways to order the test patterns of a C gate in a sequence, we intentionally selected the sequence of table 1 so that as many 'global patterns' as possible are placed consecutively in order to take full advantage of the above property.

For the remaining C gate patterns, called 'local' here, each pipeline latch must be tested independently. To apply the local patterns to a speci£c latch, the upstream latches in the link route must be made to generate the appropriate values, while the ones downstream must be set to propagate the results to the end of the link route.

Generating the needed a-input values for the pipeline latch under test is simple: If a value α must be produced at the a inputs of a pipeline latch, all the upstream latches should have value α scanned-in to their b and c inputs and the initiator at the source of the link route should also drive its ports to the same value α. This will cause each C gate to change its output to α in a ripple fashion.

3.2 Fault propagation through C gates

Propagating values down a route involves propagation through a number of consecutive C gates, which is not a trivial issue since C gates are sequential circuits.

In the 2-input C gate of Fig. 3, for example, assume that a fault upstream sets $A = \alpha$ and this value must be propagated through the gate for the fault to be detected. To propagate α from A to the output Q, the other input (B) should also be set to α. But if the fault is not present and the inverse value actually appears at A, it will not be propagated because it is blocked by the value of the other input. Then the output will remain at whatever value was stored in the C gate previously which could be wrongly detected as a fault.

Thus in order to effectively propagate a fault, the previous value of the C gate output must be taken into consideration. If the previous C gate output Q and the good machine value expected at A are equal, then B should be set to the inverse value; in case there is a fault, it will propagate through the C gate, change the output and thus will be detected. If Q is the inverse of the expected value at A, then B should be set to the expected value; if there is a fault, there will *not* be a change in the output value and thus the fault will be detected.

The above method is directly generalisable to 3-input C gates and asymmetric C gates. In a link where a value must be propagated through multiple C gates the above is simply applied to each gate down to a primary output.

3.3 Testing of completion detection circuits

The test process described so far covers all the faults in the pipeline-latch C gates but not the faults at the completion detection blocks, which generate the interconnect acknowledgment signals.

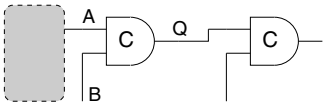


Figure 3. Fault propagation through C gates.

Table 2. Area overhead comparison

	1-of-4					3-of-6				
	partial		full		f/p	partial		full		f/p
# scan cells	35		99		2.8	67		147		2.2
Total area (μm^2)	17225		30755		1.8	28316		45228		1.6
DfT area (μm^2)	7399	(43%)	20929	(69%)	2.8	12814	(45%)	29726	(66%)	2.3
Boundary area (μm^2)	+9403	(63%)	+9403	(76%)	1.0	+10940	(61%)	+10940	(72%)	1.0

In case of a 1-of-N data encoding implementation, the completion detection block is just an (N+1)-input OR gate, probably built out of low fan-in gates, if N is large. The test patterns applied for the C-gate part of the pipeline latches cover all faults in the OR gate except for the stuck-at-0 faults at the individual inputs of the OR gate. To test these faults, a series of pattern pairs is required in which, £rst, only one of the N+1 wires is driven high at any time, followed by a pattern where all wires are cleared. As these patterns are 'global', all such faults in an end-to-end route are tested with one set of 2(N+1) patterns.

When M-of-N encoding is used, completion detection logic is more complicated and uses C elements, some of which may have to be scanned. For example for a 3-of-6 completion detection block 21 patterns are required in comparison to 10 for 1-of-4 encoding.

4 Evaluation

We have examined two interconnections with 1-of-4 and 3-of-6 encoding. Both are implemented in structural Verilog mapped for the UMC 0.18μm process using the VST standard-cell library, which does not contain any special, asynchronous gates. Since the added DfT circuits are placed locally and the only global wiring they require is a single scan-chain, the impact of routing is ignored in this evaluation. We believe this is the worst case comparison, as the area overhead of the DfT circuits will not be amortised over the total (cells and routing) area of the interconnection.

For the 1-of-4 implementation, a total of 35 scan latches are required; this number rises to 89 when the "boundary" scan-latches at the input and output ports are included. This represents a DfT area which is 43% of the total (63% including the boundary latches). In comparison, a full-scan approach [2] would require 99 scan latches (69% of the total area), or 153, incl. boundary scan (76% of total). For 3-of-6 encoding the absolute number of scan latches is higher and similar percentages were measured. Table 3.2, summarises these results.

5 Conclusions

Full-scan, LSSD is a viable approach for testing asynchronous systems and test patterns can be produced using conventional ATPG tools. Unfortunately the test structures that need to be added have a very large area overhead in a typical asynchronous system. Partial-scan testing can be employed to reduce this overhead, but there is no automatic way to decide where scan latches should be inserted.

In this paper scan-latches are inserted only to break global feedback loops, thus reducing the area overhead. The application of the method and the pattern generation process is shown in detail for an asynchronous interconnect fabric. Such interconnections are very likely to be used in the near future to build globally-asynchronous, locally-synchronous systems-on-chip.

Nearly 100% fault test coverage is achieved with a relatively moderate increase in area, for an asynchronous circuit. In comparison full-scan LSSD produces a circuit occupying about 50% more area for the same interconnect, while there is no noticeable difference in the delay overhead caused by the two methods.

Acknowledgements

This work was partially funded by the EU through the IST-2002-37796 ASPIDA project. CHAIN was created with funding from EPSRC and Theseus Logic Inc. The authors are grateful for this support.

References

[1] W. J. Bainbridge and S. B. Furber, "CHAIN: A delay insensitive CHip Area INterconnect," *IEEE Micro*, vol. 22, no. 5, pp. 16–23, Sept/Oct 2002.

[2] K. van Berkel, A. Peeters, and F. te Beest, "Adding synchronous and LSSD modes to asynchronous circuits," in *Proceedings of the International Symposium on Asynchronous Circuits and Systems*, Apr. 2002, pp. 146–155.