Test Pattern Generation and Partial-Scan Methodology for an Asynchronous SoC Interconnect

Aristides Efthymiou, *Member, IEEE*, John Bainbridge, *Member, IEEE*, and Douglas Edwards

***Abstract*—Asynchronous design offers a solution to the interconnect problems faced by system-on-chip (SoC) designers, but their adoption has been held back by a lack of methodology and support for post-fabrication testing. This paper first addresses the problem of testing C-elements, an important building block of asynchronous circuits. A simple method for generating test patterns is described which is shown to be applicable for a wide range of implementations. Based on the C-element testability, a partial scan technique was developed that achieves a test coverage of over 99.5% when applied to an asynchronous, network-on-chip, interconnect fabric. Test patterns are automatically generated by a custom program, given the interconnect topology. Area savings of at least 60% are noted, in comparison to standard, asynchronous, full-scan level-sensitive scan devices (LSSD) methods.**

***Index Terms*—Asynchronous circuits, ATPG, globally-asynchronous, locally-synchronous (GALS), scan-testing, stuck-at fault testing.**

I. Introduction

The ITRS road-map predicts a widespread adoption of globally-asynchronous, locally-synchronous (GALS) systems to solve the increasingly difficult task of distributing a single clock signal on a chip. Consequently, it is expected that asynchronous circuits will start to make their way into mainstream, commercial integrated circuits (ICs). One barrier to widespread acceptance of asynchronous technology is an appropriate test methodology. Techniques for integrating testability into synchronous system design are not directly applicable to asynchronous systems. Problems arise from redundant logic inserted to deal with hazards and races, the distributed nature of self-timed control, multiple feedback loops and the extensive use of state-holding gates such as C-elements. The major problem is the absence of a global clock making it difficult to single-step through a sequence of states. Traditionally, feedback loops are broken with scan latches; in the context of an asynchronous system, this technique may lead to an excessive area overhead and to an adverse impact on performance.

One of the major contributions of the presented work, which enabled the above high fault coverage, is the development of a new method for generating test sequences for C-elements and other circuits with one feedback loop. We show that, at least for a 2-input C-element, the produced test sequence is the shortest possible, that it avoids introducing hazards into the test sequence and that it is applicable to a wide range of possible implementations.

The second contribution of this paper is a novel design-for-testability and test-pattern generation technique developed to produce a fully-testable version of CHAIN, an asynchronous interconnect fabric [1] which is particularly well-suited for GALS systems-on-chip. By identifying pipeline structures and distinguishing between global signal routes and local paths, the number of test patterns can be greatly reduced by applying a common set of global patterns. As a practical outcome of this work, a computer program has been developed which, given the topology of a CHAIN interconnect, produces a sequence of test patterns that achieves 99.5% fault coverage, under the standard single stuck-at fault model.

Even in deep-sub-micron technologies, the stuck-at fault model is still considered the *de facto* fault model. Acceptable coverage rates vary according to the production volume and the intended market, but a coverage rate of over 99% is not untypical for commercial synchronous products. For asynchronous circuits such a high coverage is still considered a challenge. Thus this work deals with bridging the gap in testability for asynchronous circuits; achieving high fault coverage for more advanced fault models is left as a future challenge.

In particular, delay faults can be an issue in asynchronous circuits, for example when they break timing assumptions, such as isochronic forks. This work does not consider delay faults, in general; they are also left for future work. However, since a delay insensitive protocol is employed, there are few timing assumptions in CHAIN interconnects, so we do not expect such faults to be significant.

The rest of the paper is organized in three main parts. An overview of asynchronous circuit testing, including relevant past work and our overall methodology is given in Section II. Our work on generating test-pattern sequences for C-elements is described in Section III. The third part discusses design-for-testability and test-pattern generation for the CHAIN interconnect. Section IV introduces the CHAIN interconnection fabric and discusses our test strategy. Sections V and VI describe the procedure for testing the basic components of the interconnect, while Section VII describes the pattern generation program. The eval-

Manuscript received December 7, 2004; revised August 22, 2005. This work was supported in part by the European Union (EU) through the IST-2002-37796 ASPIDA project. CHAIN, an asynchronous interconnect fabric, was created with support from EPSRC and Theseus Logic Inc.

A. Efthymiou is with the School of Informatics, University of Edinburgh, Edinburgh EH9 3JZ, U.K. (e-mail: aefthymi@inf.ed.ac.uk).

J. Bainbridge is with the Silistix Ltd., Armstrong House, Manchester Technology Centre, Manchester M1 7ED, U.K. (e-mail: jbainbridge@ieee.org).

D. Edwards is with the School of Computer Science, University of Manchester, Manchester M13 9PL, U.K. (e-mail:doug@cs.man.ac.uk).

Digital Object Identifier 10.1109/TVLSI.2005.862722

1063-8210/$20.00 © 2005 IEEE

uation of the area impact of our method in comparison to standard full-scan is presented in Section VIII. Finally, Section IX concludes the paper.

II. Testing Asynchronous Circuits

A variety of methods have been proposed for testing asynchronous circuits. The interested reader is referred to [2] for a survey of classic past work on such methods. We only mention work directly relevant to our research here.

One of the most unconventional methods is self-checking [3], [4] (or self-diagnostic [5]), where faults are detected because they cause the asynchronous circuit to halt when a handshake protocol is violated. Although this method can be applied to a large class of asynchronous circuits, Brzozowski and Raahemifar [6] have shown that it ignores internal faults in some asynchronous building blocks, such as the C-element.

C-elements are effectively a special form of set–reset latches, e.g., a 2-input C-element waits (its output remains unchanged) until both its inputs assume the same logic value, then sets its output to that value. When constructing a C-element from basic logic gates, such as those present in a common standard-cell library, it has been shown that approximately 50% of the possible internal stuck-at faults do not actually cause an asynchronous circuit to halt [6]. Thus the self-checking technique does not offer a high fault coverage for the conventional (input) stuck-at fault model when internal faults are also considered.

Conventional, scan-based methods have also been proposed [7]–[11]. We discuss the differences of our approach to most of these methods below. Our approach builds upon the work by Philips Research [9], [12], [13] which is briefly explained below.

A. Feedback Loop Scanning

Asynchronous circuits can be viewed as combinational circuits with feedback loops. The feedback loops are "broken," in one of many possible ways, with level-sensitive scan devices (LSSD) [14], as shown in Fig. 1. It is essential that level-sensitive latches are used so that the original, asynchronous, operating mode of the circuit is still available by keeping both latches transparent; this would not be possible with the common edge-triggered flip-flops.

In test mode, the asynchronous circuit operates synchronously: after a pattern has been scanned in, it is applied to the circuit and the outputs are latched and scanned out; as the loops are broken, the asynchronous FSMs are single-stepped. This greatly simplifies the generation and application of test patterns and the overall method can be easily automated.

B. Partial-Scan Testing

Although the above method does solve the testability problem for most asynchronous circuits, inserting scan latches at every feedback path present in the circuit incurs a very significant area overhead. Typically, the number of sequential elements in an asynchronous circuit is much higher than the number of state flops in a synchronous circuit. Thus partial-scan techniques [15] should be employed to keep the testing area overhead to a minimum.

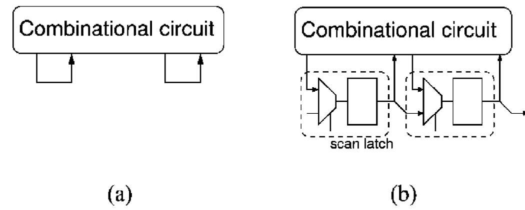

Fig. 1. Scan latch insertion in asynchronous circuits. (a) Asynchronous circuit. (b) "Scanned equivalent."

Unfortunately, there is no known general way of selecting which feedback loops to scan. This tends to depend on the specific topology of each circuit. The few publications on partial-scan testing of asynchronous circuits for stuck-at faults [7], [16] make assumptions about how the circuits are constructed and thus can only work for the specific design methodologies and tools. In particular, the methodology in [16] was developed specifically for the Tangram design environment [17] although it could be extended for use in other asynchronous synthesis environments. Difficult to control state elements are identified and the Tangram source code is manually edited to enable them to be scanned.

Khoche and Brunvand [7] introduce a scan-based approach for a specific macromodule-based design methodology where they scan all Select, Toggle, and most Call macromodules. The method achieves very good results but it can only be applied in their specific design methodology. In addition they modify every C-element in the system to include an OR gate and two extra inputs. Although this is not as expensive as adding a scan latch, it still increases the DfT area overhead considerably.

Some of the ideas underlying our solution for adding testability in the CHAIN interconnect were first proposed in [8], where a partial-scan delay fault testing methodology is presented for asynchronous circuits. Similarly to our approach, they differentiate between two types of feedback loops (defined below) and only scan the global ones. Our approach uses a different fault model (stuck-at) and the regular structure of CHAIN leads to a significantly simpler solution to test pattern generation.

C. Proposed Approach

Asynchronous feedback paths can be categorized as local and global depending on whether the path is internal to an asynchronous sequential gate (C-element, SR-latch, etc.), or spanning a number of asynchronous gates [8]. This paper expands on our preliminary work [18], [19] where partial-scan LSSD is employed in such a way as to only scan global feedback loops (GFLs), rather than every asynchronous sequential gate.

In order to test an asynchronous circuit it is *essential* to be able to observe and control the global feedback loops by adding scan latches. For correctly designed asynchronous sequential gates, a single change in the inputs can only cause a single (output) state transition, if at all; for another transition to occur, at least one of the inputs must change again. If the GFLs in the circuit are not "broken" in test mode by scan latches, state transitions could make their way through some GFL back to a sequential gate's input, thus causing further output transitions with only one change in the primary inputs of the circuit. Thus scanning

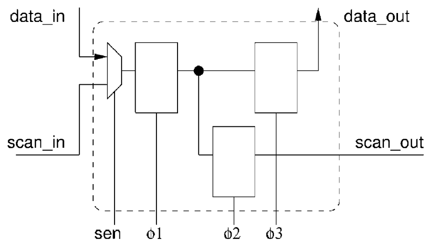

Fig. 2. Scan latch design.

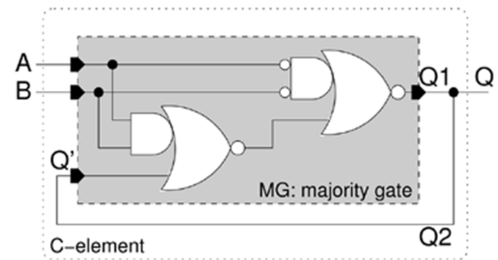

Fig. 3. An implementation of the C-element by adding a feedback path to a majority gate.

all the GFLs is essential to solving one of the major problems of testing asynchronous circuits which is that state transitions are not easily controllable.

Since not all state-holding gates are scanned, the generation of appropriate values in internal nodes to test for faults requires sequential patterns, which tend to depend on the circuit topology. In this work we exploit the regular topology of CHAIN interconnect circuits and a useful property of the test patterns of C-elements to produce an elegant solution to test pattern generation.

For sequential testing, the values held in the "unscanned" state-holding elements must not change from the application of one pattern to the next. This is easy to achieve in synchronous circuits, as the unscanned latches are held opaque by the clock and any values appearing at the circuit inputs do not affect the state stored in them. On the contrary, changes at the inputs of asynchronous circuits can modify the values held in unscanned state-storing elements. Therefore, we must not allow any changes at the inputs while new values are being scanned in and this is reflected in the design of the scan latches shown in Fig. 2. Compared to standard LSSD, a second, parallel, slave latch is added which keeps `data_out` unchanged while patterns are being scanned in.

Compared to standard LSSD scan latches used in asynchronous circuits [9], the *dual-slave* latches that we employ are approximately 50% larger. This is the price paid for using a partial scan approach, but it is justified by the lower total number of scan-latches required compared to the full-scan approach as will become apparent in Section VIII.

The use of scan latches introduces three global phase signals: ϕ_1, ϕ_2, ϕ_3, and a scan-enable signal, *sen*. Five operating modes (based on [9]) are defined.

- *Asynchronous:* $sen = \phi_2 = 0$, $\phi_1 = \phi_3 = 1$. This is the normal operating mode of the circuit.
- *Flush:* $sen = 1$, $\phi_3 = 0$, $\phi_1 = \phi_2 = 1$. This mode can be used to quickly test the scan-path.
- *Scan:* $sen = 1$, $\phi_3 = 0$, 2-phase nonoverlapping clocking for ϕ_1, ϕ_2. This mode is used for scanning in and out values captured on the scan latches.
- *Evaluation:* $sen = 0$, $\phi_3 = 1$, $\phi_1 = \phi_2 = 0$. This mode applies the value previously loaded in the scan latch to the asynchronous circuit. Note that the feedback loop is disconnected during evaluation; otherwise there might be multiple state changes in an asynchronous circuit with global feedback loops.
- *Load:* $sen = 0$, $\phi_1 = 1$, $\phi_2 = \phi_3 = 0$. Loads the output of the asynchronous circuit onto the scan latch. Changing from *evaluation* to *load* must make sure that ϕ_1 and ϕ_3 are never high at the same time.

To apply a pattern, the scan mode is used for a number of cycles to load the scan latches with the pattern values. This is followed by the evaluation and then the load operations. Finally the results are scanned out while new patterns are being scanned in.

III. Testing Muller C-Elements

Muller C-elements [20] are an important building block in asynchronous circuits, so their testability is of key importance. Unfortunately, standard ATPG tools are unable to generate patterns for circuits that retain state through the presence of local feedback loops. Experiments we performed using commercial ATPG tools to automatically generate patterns for C-elements implemented using standard cells of the UMC/VST 0.18 μm library showed that the highest achievable test coverage was 88.9%, even when manually adding extra test signals and/or gates.

A. Proposed Pattern Generation Method

Fig. 3 shows an implementation of a C-element using two complex gates from a standard cell library. The implementation is essentially a *majority gate*, highlighted in grey in the figure, with the output connected back to one of the inputs. The key observation here is that if the feedback loop is removed, the sequential C-element is transformed into a combinational circuit for which ATPG programs can trivially produce patterns to cover all faults, assuming the circuit is irredundant.

Theorem 1: For a 2-input C-element constructed by adding a feedback path from the output of a 3-input majority gate to one of its inputs (as in Fig. 3), all faults in the C-element dominate those of the majority gate. Moreover, we can produce a test sequence directly derived from the patterns used to test the underlying majority gate.

Proof: Let T_{mg} be a test, i.e., a set of test patterns, that detects all single, stuck-at faults in the majority gate (MG). T_{mg} will be a set of binary vectors of the form abq', corresponding to the values of the input ports A, B, Q' of MG. As MG is a combinational circuit, the order in which the patterns are applied is not important.

As can be seen in Fig. 3, the addition of the feedback line introduces only two new faults, at points Q and Q2, with Q1 representing the MG output. Assuming that input patterns can be found to drive node Q' to any value (as shown below), any faults at Q2 are equivalent to faults at Q', which are already

covered by T_{mg}. Similarly, any fault at Q is detected with the same patterns that test the equivalent fault at Q1, again already covered by T_{mg}.

The only remaining issue is how to set Q' to a desired value, as it now depends on the C-element output Q, while it is an independent input in MG for which T_{mg} was generated. This is solved by taking advantage of the fact that the specification of all C-elements includes an input combination that sets their output to 1 (or 0) regardless of the previous value of their output.

The new test T for the (sequential) C-element will be composed of patterns of the form ab corresponding to the values on inputs A, B of the circuit. The *sequence* of patterns for testing the (sequential) C-element is now important. T can be generated from T_{mg} so that every pattern in T_{mg} of the form abq' produces the sequence: $q'q'$, ab in T. The first pattern in the sequence generates the required q' value using the input combination that does not depend on the previous output; next the actual pattern is applied. ■

The above method can be easily extended to C-elements with more than 2 inputs, asymmetric C-elements, and C-elements with set/reset inputs (for initialization). It can also be used for other sequential gates with one feedback loop. For example, we have used the above method for SR-latches build from cross-coupled NAND or NOR gates.

The pattern generation process described thus far produces a test T that fully tests the C-element, but contains double the number of patterns of T_{mg}. For example, the two patterns $abq' = 101, 011$ for the majority gate would produce the sequence 11, 10, 11, 01 for the C-element. Instead of setting the output of the C-element to the appropriate value for every pattern in T_{mg}, the patterns of T_{mg} can be ordered in a sequence so that each one re-uses the value of q' generated by the previous pattern. This will reduce the number of patterns for T considerably; in the above example, the final sequence would be 11, 10, 01.

This process of finding a valid sequence of patterns should take into account the following considerations.

- A sequence of two patterns that normally leave the output unchanged could be causing a hazard if, while changing from one to the other, it is possible for the output to change value. For example, in a 2-input C-element with its output previously set to 0, the sequence $AB = 10$, $AB = 01$ can cause a hazard because as the inputs change values they could briefly become $AB = 11$, which would change the output to 1. Thus such sequences must be avoided and sometimes extra patterns have to be inserted to prevent such hazards. Unfortunately, the extra patterns do not exercise any further faults. Such hazards would not be a problem if the C-element inputs changed simultaneously, but in asynchronous systems simultaneity cannot be guaranteed, therefore pattern generation must be conservative and avoid potentially hazardous sequences.
- Sometimes, in order to complete the test sequence for a C-element, a pattern that sets the circuit output (Q in Fig. 3) to a specific value is required, but it is not available because it has already been used. This happens, for example, when the remaining patterns of T_{mg} to be placed in the test sequence all have input $Q' = v$, but the last pattern in the sequence has already set the circuit output to the opposite value of $(\neg v)$. In this case an extra pattern is added which sets the output (Q) to the appropriate value, so that the test sequence can proceed. Unfortunately, this extra pattern does not exercise any further faults.

TABLE I
TEST PATTERNS FOR 2-INPUT C-ELEMENT

a	1	1	0	0	1	0	0	1
b	1	0	0	1	1	1	0	0
q	1	1	0	0	1	1	0	0
					*		*	

The process of generating a valid sequence of patterns has been implemented in a program which takes as input ATPG patterns for the majority gate from a standard ATPG tool. Thus the process of generating test patterns for a C-element can be fully automated.

Table I illustrates the test vectors for a 2-input C-element. The entries marked with "$*$" are the patterns added due to the considerations explained previously. 100% single stuck-at fault test coverage is achieved.

B. Universal Application of Patterns

Intuitively, it might seem that different gate implementations of a specific C-element would require different sets of test patterns. On the contrary, while working on pattern generation for C-elements, we observed that the same set of patterns was sufficient for all different implementations (gate mappings) of C-elements based on the 3-input majority gate.

Theorem 2: All nonredundant, standard-cell based implementations of the 3-input majority gate, $q = ab + ac + bc$, can be tested for single and multiple stuck-at faults using the same set of six test patterns.

Proof: The proof is based on the following three steps.

- In the presence of any stuck-at fault in the majority circuit, the resulting logic function (called fault function) is a positive function [6] : it is defined as a sum of products of uncomplemented variables, or the constants 0, 1.
- By enumerating all the possible fault functions, we show that six patterns are both necessary and sufficient to detect all single and multiple stuck-at faults.
- The fault functions that are detectable *only* by these six patterns can be caused by at least two, unrelated, single stuck-at faults on the fanout branches of the circuit inputs. Such faults can occur in sum of products (e.g., 2-level NAND), product of sums (2-level NOR) implementations and also in complex gate implementations.

1) Suppose that, in the presence of a fault, the resulting logic function of the majority gate is not a positive function. Assuming that input a is complemented in the fault function, for some value combination of the other inputs the output will be $\neg a$. Similarly for the other inputs b, c. Since all inputs go through an even number (0 or 2) of inverted gates (for both NAND and NOR implementations, thus any other implementation), this is not possible. Thus all the resulting fault functions are positive.

TABLE II
POTENTIAL FAULT FUNCTIONS OF 3-INPUT MAJORITY GATE

fault function	NAND impl	NOR impl.	detecting patterns
0	...	...	p4, p5, p6
abc	...	...	p4, p5, p6
ab	...	...	p4, p5
bc	...	...	p5, p6
ac	...	...	p4, p6
ab+ac	b3/c3 sa0	b1/c2 sa0	p4
ab+bc	a2/c2 sa0	a1/c3 sa0	p5
ac+bc	a1/b1 sa0	a2/b3 sa0	p6
a	...	...	p3, p4
b	...	...	p2, p5
c	...	...	p1, p6
a+bc	b1/c2 sa1	b3/c3 sa1	p3
b+ac	a1/c3 sa1	a2/c2 sa1	p2
c+ab	a2/b3 sa1	a1/b1 sa1	p1
a+b	...	...	p2, p3
a+c	...	...	p1, p3
b+c	...	...	p1, p2
a+b+c	...	...	p1, p2, p3
1	...	...	p1, p2, p3

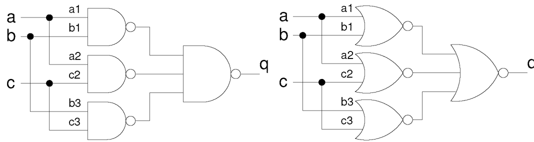

Fig. 4. NAND and NOR majority gate implementations.

2) Table II shows all the possible fault functions for a 3-input majority gate, the faults in which they correspond for two circuit implementations (shown in Fig. 4 and the patterns (shown in Table III) that can be used to detect them. It can be seen that six of the functions are each detectable only by a single, specific pattern, all of which are distinct. Moreover, all the other fault functions can be detected by two or three of these patterns. Thus, assuming that these functions can be realized in the presence of faults, the set of six patterns shown in Table III is both necessary and sufficient to detect all faults.
3) What remains to be proved is that at least the six aforementioned fault-functions of Table II can occur in any implementation. Fig. 4 shows the majority gate implementation as a sum of products and as a product of sums. The second and third columns of Table II show the faults at the input fan-outs that can result in the fault functions of interest. Other internal faults can produce the same fault functions; we are interested in the input faults because if these are detected, all internal faults are also detected for circuits without internal fanout branches [15, Th. 4.1], which is the case for the majority circuit.

Note from the table that two, different stuck-at faults in the circuits can produce each fault function. This means that, even in a complex gate implementation, such as the one in Fig. 3, where one of the three inputs has no fanout branches, all of the test patterns are still required. ■

The above theorem can be used to create a test sequence for C-elements built with standard-cells. The following test sequence of length 8 can be used to test *any* 2-input C-element: $11(x),\ 10(1), 00(1), 01(0), 11(0), 01(1), 00(1), 10(0)$based on the following sequence of the six patterns of the corresponding majority gate: -, p5, p1, p2, p6, p4, p1, p3. The values in brackets represent the third input of the majority gate, i.e., the output of the previous pattern. Note that the first pattern is required to set the output to a known state.

TABLE III
TEST PATTERNS FOR 3-INPUT MAJORITY GATE

	a	b	c
p1	0	0	1
p2	0	1	0
p3	1	0	0
p4	0	1	1
p5	1	0	1
p6	1	1	0

Although the above sequence is longer by one pattern compared to that given by Brzozowski and Raahemifar [6] for their implementation of the C-element: $11, 01, 10, 00, 01, 10, 11$, their sequence has two potential, hazard-generating pairs. A direct conversion to a hazard free sequence would have produced nine patterns, one more than the test sequence proposed here.

It is interesting to note that they also made the observation that the same pattern sequence was able to test many different implementations of the 2-input C-element. This paper finally explains why this happens for the majority style implementations, since we have shown that for all such implementations the same set of six patterns is required regardless of the exact mapping into logic gates.

IV. CHAIN ASYNCHRONOUS INTERCONNECTION FABRIC

Delay insensitive communication links use an encoding that is tolerant of delay variations in the wires, and allows the absence or presence of valid data to be encoded within their signalling activity, and detected explicitly by a "completion detecto." Such links also use a flow control signal (acknowledge) in the reverse direction to regulate when the sender delivers further data.

CHAIN is an architecture for SoC interconnect using delay insensitive signaling, asynchronous flow control, network components (end-point gateways, switches, and pipeline latches) implemented using self-timed techniques and a message passing protocol. CHAIN supports connections between network components using "L" parallel links (each with its own acknowledge signal), where each link has "G" code groups each using an M-of-N encoding [21] where a valid code symbol has M asserted wires from a group of N wires. Variations in $L,\ G,\ M,\ N$ affect the tradeoff between area, performance, power-consumption [22]. At the time of publication, working silicon exists for one of the fastest combinations, $L = 4, G = 1, M = 1, N = 5$ with four-phase (return-to-zero) signalling. All of our examples in this paper use four-phase signalling, which simplifies the control logic.

Fig. 5 shows the building blocks of a CHAIN interconnect, all of which are derived from the delay-insensitive *pipeline latch*, highlighted in grey, which stores and forward the transmitted data.

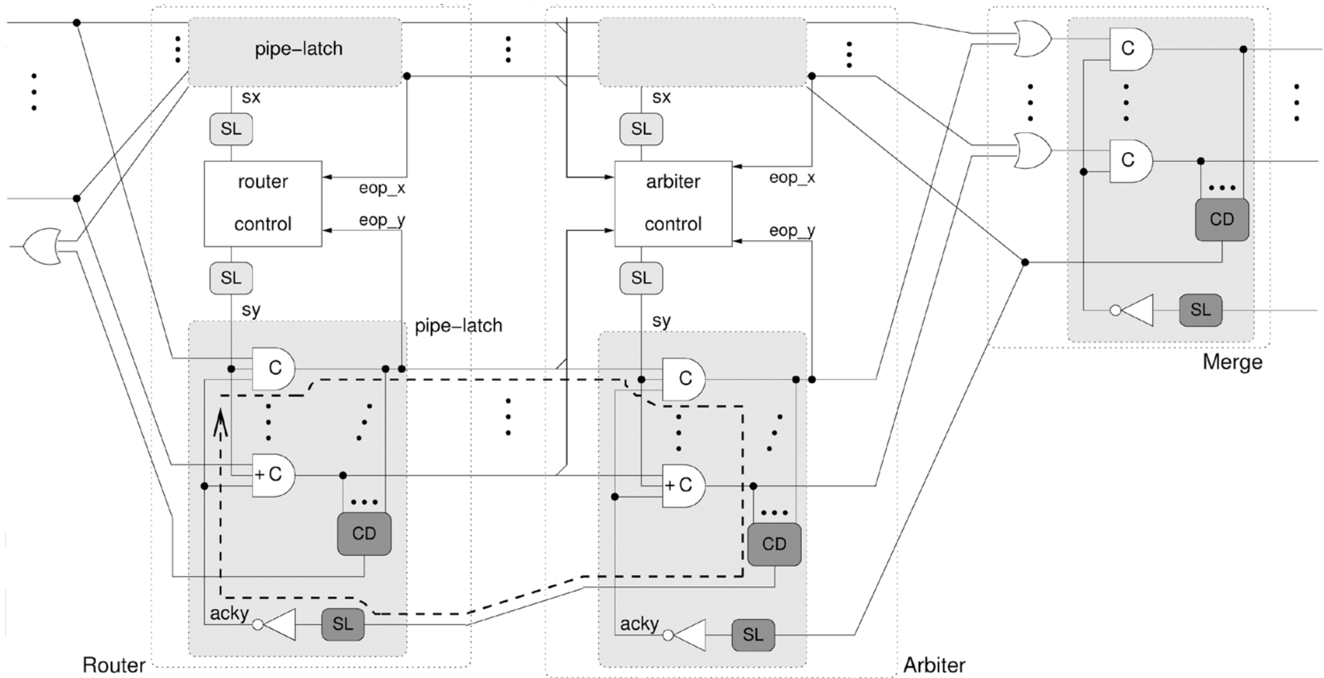

Fig. 5. CHAIN interconnect.

The *router* can divert a packet in one of two possible ways; the direction is determined by the first symbol of the packet which is removed as the packet flows through. The *merge* allows handshakes from either of its two inputs to be passed to the output, but requires that only one input link is active at any time. The *arbiter* ensures that the merge units requirement of mutual exclusivity of activity on its input links is enforced. Together the arbiter and merge act as a self-controlling *multiplexer* for complete handshakes.

Completion detection is performed at the blocks labeled "C" and the completion signal of each stage is used to acknowledge the previous stage. Long connections between these components can be broken into smaller segments using extra pipeline latches. From here on the term pipeline latch will be used for all three variations of pipeline latches found in CHAIN.

Using the terminology introduced in Section II-C, in a CHAIN interconnect there are multiple "local" feedback loops (in C elements) and some "global" feedback loops (between each pipeline latch stage and its downstream pipeline latch). One of the latter, is highlighted using a dotted-loop in Fig. 5. The blocks marked *SL* indicate where scan latches are inserted to break these loops when the circuit is under test.

A. Test Strategy

In test mode, all *route* combinations from transmitters to receivers are set up by scanning appropriate values into the scan-latches of the router and arbiter blocks. For each route, patterns are transmitted from the source ports, while the output ports and the scanned acknowledge signals are observed. It is desirable to test as many routes as possible in parallel—depending on the interconnection topology—so as to minimize the testing time. Since some parts of the interconnect are shared among different end-to-end routes, not all route combinations need to be tested, as long as every pipeline latch is tested at least once. Our custom ATPG program, described in Section VII, produces test patterns taking all these issues into account.

For each route, the interconnection is tested in three phases. First, patterns are applied to expose stuck-at faults in the C-elements of the pipeline latches, including those inside router, arbiter and merge units. The second phase targets faults in the completion detection blocks of the pipeline latches. Finally, the control parts of the routing components are tested.

The first two phases are explained in the following sections in detail. The test patterns used in the last phase were generated using conventional ATPG and then arranged in sequence manually for each of the two types of control blocks; they are then re-used for each instance of router and arbiter blocks. As we do not yet have a general method for arranging the patterns for these circuits in sequence, this last phase is not described here further.

V. Testing C-Element Network

For the CHAIN C-element network, we are using the sequential test patterns produced using the method described in Section III. These sequences achieve 100% fault coverage for all types of C-elements.

Our task is simplified by the regular topology of the circuits. All parallel C-elements inside a pipeline latch are connected in the same way: one input (a), separate for each C-element, is driven from the upstream pipeline latch. The other two (one for merge units) inputs are common to all elements and are driven by scan-latches: b for the acknowledge signal, c for route select, sx or sy. The outputs become the "a-inputs" of the next stage.

In a CHAIN interconnection, three types of C-elements can be found: 2-input C-elements (CM2), 3-input C-elements (CM3), and 3-input asymmetric C-elements (CM21). Input c of CM21, marked with a "$+$," is consulted only for rising output transitions. We aim to use the same patterns for all parallel

TABLE IV
C-ELEMENT 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)	

C-elements in a pipeline latch, testing the same faults in all C-elements simultaneously. Thus, it is essential that the same patterns can be used for the CM3 and CM21 gates, which are simultaneously present in the pipeline latches of routers and arbiters.

The comparison of the test patterns for these two elements showed that all faults in CM21 are covered by the patterns generated for CM3. Table IV, summarizes the test patterns for all three C-elements. The only difference between CM3 and CM21 is at the last line of the table: pattern $\{0, 0, 1\}$ with a previous output of 1 generates a 0 for CM21 (c is the '+ input') while it remains 1 for CM3. This is a minor difference and because it happens at the very last pattern, the only action required was to add an "all-zeros" pattern at the end of the sequence to reset all the C-element outputs to logic zero.

Although this feature was not exploited here, it is interesting to notice that the pattern sequence for CM2 is the same as that of the first part of the CM3 and CM21 (ignoring the third input, c) and the output sequence is also the same.

A. Test Patterns for C-Element Network

Many of the C-element patterns have the useful property that they produce an output value equal to that of their 'independent' input a. These patterns are important because they can be used to exercise all the pipeline latches along a route (from an initiator to a target) simultaneously, since the same values are generated at the a-inputs of all the C-elements in all the latches along the route[1]. This saves a significant amount of test time. In order to take full advantage of this property, we intentionally selected the C-element sequences shown in Table IV, which place as many of these "global patterns" back-to-back as possible.

We call the remaining patterns 'local', as each pipeline latch must be tested separately: the upstream latches in the route generate the appropriate values, while the ones downstream are set to propagate the results to the end of the 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 route should also drive its ports to the same value α. This

[1]Note that pipeline latches are also used in place of repeaters to improve the interconnect throughput.

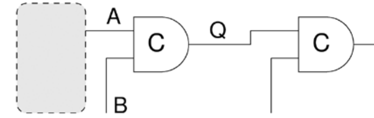

Fig. 6. Fault propagation through C-elements.

will cause each C-element to change its output to α in a ripple fashion.

B. Fault Propagation Through C-Elements

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

In the 2-input C-element of Fig. 6, 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-element previously which could be wrongly detected as a fault.

Thus in order to effectively propagate a fault, the previous value of the C-element output must be taken into consideration. If the previous C-element output Q and the good machine value expected at A are equal, then B should be set to the inverse value; in case of a fault, it will propagate through the C-element, 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 also holds for 3-input C-elements and asymmetric C-elements. When there are multiple C-elements in series, the above method is applied to each element down to a primary output.

VI. Testing Completion-Detection Blocks

The completion detection block for one-hot (1-of-N) encoding is just an $N + 1$-input OR gate, usually built from low fan-in gates. The test patterns applied for the C-elements cover all faults in this OR gate except for the stuck-at-0 faults at its inputs. A series of pattern pairs is required to test them: first, only one of the $N+1$ wires is driven high at any time, followed by a pattern where all wires are at logic zero. As these patterns are "global," all such faults in an end-to-end route are tested with one set of $2(N + 1)$ patterns.

The completion detection circuit for the M-of-N encoding is significantly harder to test since it contains sequential gates. Not many encodings are useful for the specific application, since the number of wires cannot be excessive. In practice CHAIN fabrics with either 3-of-6 or 2-of-7 encoding [22] have been designed at the time of publication; of these, the completion detection circuit for 3-of-6 (see Fig. 7) is more complex, as the encoding is more dense. This circuit is used to illustrate our test methodology here.

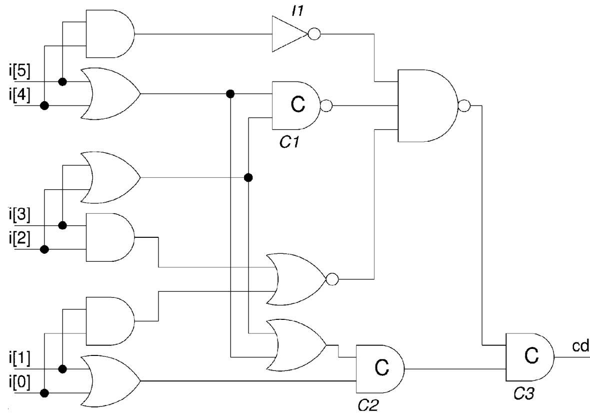

Fig. 7. Completion detection circuit for 3-of-6 encoding.

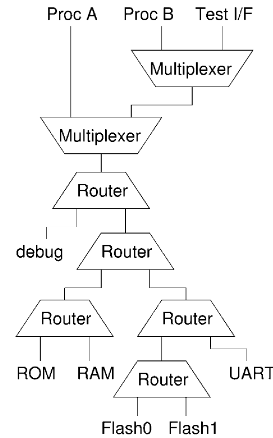

Fig. 8. Example CHAIN interconnect.

As both inputs of the last C-element (C3 in Fig. 7) originate from other C-elements, it is hard to generate some of the patterns needed for it. For example, the required pattern sequence $\{11, 10\}$ cannot be applied to the inputs of C3: for C2 to become 0, all primary inputs must be 0, but then the other input of C3 will also become 0. In order to be able to drive the inputs of C3 to the values needed, a scan latch in at least one of C1 or C2 should be added to enable us to set their outputs independently. This would slow down the normal operation of the circuit as the latch would be in the critical path[2]. In order to avoid this we decided to add another input to the circuit, *tstIn*, and convert inverter I1 to a NOR gate. This enables us to set the upper input of C3 to 1 independently of its other input and provide the pattern that we need.

Furthermore, the last C-element (C3 in Fig. 7) restricts the propagation of faults to the single circuit output. In the previous section we showed a way of propagating faults through C-elements which could be applied here, but this would mean that each of the subcircuits leading to the two inputs of C3 would essentially have to be tested in series. Thus, in order to reduce the number of patterns needed, we decided to add observation points at the two inputs of C3. These require the insertion of scan latches, but since we do not use them to inject input values, they are simpler than those shown in Fig. 2: the slave latches controlled by ϕ_3 are removed.

With the above modifications, a sequence of 21 test patterns is needed to test the 3-of-6 completion-detection block. The patterns in the sequence are 'global', i.e., in an end to end route they need to be applied only once for all the pipeline latches in the route. Thus the extra input needed in each block (tstIn) can be derived from a single scan-latch for the whole route, so as to reduce the testing area overhead.

In summary, to test all the pipeline latches in an end to end route, the following patterns are applied: *(a)* 5 global patterns, *(b)* 3 local for each 'plain' or merge pipeline latch, or 7 local for each latch in an arbiter or router, and *(c)* 21 (10 for 1-of-4) global patterns for testing the completion detection blocks. Since many links are shared in an interconnection, the local patterns do not have to be repeated for the parts that are shared by more than one link.

[2]Recall that another scan-latch is already added in the acknowledge path to break the global loop formed between consecutive pipeline latches.

VII. Automatic Pattern Generation for the Interconnect

Having determined the sets of test patterns required for the individual parts of a CHAIN interconnection, it is relatively straightforward to automate the production of a complete sequence of test patterns for a given topology of a CHAIN interconnection. Such a program has been implemented and tested in a number of CHAIN topologies.

It works in a number of phases: first, the maximum number of parallel routes from CHAIN initiators to targets are identified. Since these routes are independent they can be tested concurrently to save time. When no more routes can be tested in parallel, the second phase begins where the remaining, end-to-end routes are tested one by one. In these two phases, after the pipeline latches of an end-to-end route have been tested, the control part of the last untested router in the route is tested. Finally, in the third phase the control blocks of the arbiters are tested. These are currently tested one at a time because they require patterns coming from two initiators at the same time. We are working on algorithm development to test these blocks in parallel, where the interconnect topology allows it.

VIII. Evaluation

The network topology shown in Fig. 8 was used as a "command network" in an experimental smart-card chip [23] ; it was completed before the methodology described here was developed, so there are no DfT circuits in the fabricated chip. The topology is used here to evaluate the area impact of the proposed testing method. The interconnect contains a total of nine ports: the three ports at the top are initiators (masters) and the six at the bottom are targets (slaves). The ports include two processor memory ports (Proc A/B), a test controller port (Test I/F), four memory ports (RAM, ROM, Flash0/1), a UART connection port, and an off-chip debugging port (debug).

We have evaluated two implementations of the interconnect with 1-of-4 and 3-of-6 encoding. They are implemented in the UMC 0.18 μm 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 wire

TABLE V
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

routing to the silicon area is ignored in this evaluation. We believe this is the worst case comparison, as the area overhead of the DfT circuits reported here is not amortised over the total (cells and routing) area of the interconnect.

A total of 35 scan latches are required for the 1-of-4 implementation; including the "boundary" scan-latches at the input and output ports raises this number to 89. This represents a DfT area which is 43% of the total (63% including the boundary latches). In comparison, using the Philips full-scan approach [9] would require 99 scan latches (69% of the total area), or 153, including boundary scan (76% of total).

For 3-of-6 encoding the absolute number of scan latches is higher but similar percentage increases are measured. Table V, summarizes these results and shows the ratio of full-scan vs partial-scan areas (f/p).

It is clear that the proposed partial-scan method provides very substantial area savings compared to the full-scan approach: the circuit area with full-scan is over 1.6 times larger. However, our approach requires 3 latches per scan element, with a cell area of 211 μm^2, rather than 2 used for standard LSSD (151 μm^2). As the results show, even with the increased area per scan latch, the total area overhead imposed by full-scan LSSD is much higher. Moreover, since for both scan latches the critical path from the input to the output is through a multiplexer and two latches, the delay overhead—in normal operating mode—of both methods is identical.

Our automated pattern generator produced a sequence of patterns which achieves a total fault coverage of over 99.5%. This coverage includes faults in the boundary scan latches and the scan chain. In fact, the only faults that are undetected are in the control blocks of the arbiters, where a metastability filter, implemented with standard cells [24], requires the use of OR gates with all inputs tied together, which cannot be tested for stuck-at-0 faults at their inputs individually.

IX. CONCLUSIONS

Although asynchronous circuits are likely to become more widespread in the future as part of GALS systems, they have serious testability issues that need to be addressed. In a GALS system the interconnect is the component that is most likely to be asynchronous. This paper presents a partial-scan methodology and automatic test pattern generation for the CHAIN asynchronous interconnect.

Scan-latches are inserted to break GFLs only, thus reducing the testing area overhead by at least 60% compared to asynchronous full-scan methods, while there is no noticeable difference in the delay overhead caused by the two methods. Test pattern generation is fully automated; we developed a program which reads the topology of the interconnect and the order of the scan chain and outputs the test pattern sequence. We report stuck-at fault coverage of over 99.5% for a CHAIN interconnect with three "initiators" and six "targets."

One of the major contributions of the paper, which enabled the above high fault coverage, is the development of a new method for generating test sequences for C-elements and other circuits with one feedback loop. We show that, at least for a 2-input C-element, the produced test sequence is the shortest possible and that it is applicable to a wide range of possible implementations.

Although the presented techniques can currently be applied only to CHAIN interconnects, we believe that elements of this work can be the basis for testing a larger class of asynchronous circuits. Thus in future work we will investigate general ways of producing test patterns for sequential circuits with multiple "local" feedback loops expanding on the technique shown here for C-elements.

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, Sep./Oct. 2002.

[2] H. Hulgaard, S. M. Burns, and G. Borriello, "Testing asynchronous circuits: A survey," *Integration, the VLSI J.*, vol. 19, no. 3, pp. 111–131, Nov. 1995.

[3] P. A. Beerel and T. H.-Y. Meng, "Semi-modularity and testability of speed-independent circuits," *Integration, the VLSI J.*, vol. 13, no. 3, pp. 301–322, Sep. 1992.

[4] V. I. Varshavsky, M. A. Kishinevsky, V. B. Marakhovsky, V. Peschansky, L. Y. Rosenblum, A. R. Taubin, and B. S. Tzirlin, Eds., *Self-Timed Control of Concurrent Processes: The Design of Aperiodic Logical Circuits in Computers and Discrete Systems*: Kluwer Academic, 1990.

[5] I. David, R. Ginosar, and M. Yoeli, "Self-timed is self-diagnostic," Dep. Elect. Eng., Technion, Haifa, Israel, Tech. Rep. EE PUB no. 758, 1990.

[6] J. A. Brzozowski and K. Raahemifar, "Testing C-elements is not elementary," in *Proc. Asynchronous Design Methodologies*, May 1995, pp. 150–159.

[7] A. Khoche and E. Brunvand, "A partial scan methodology for testing self-timed circuits," in *Proc. IEEE VLSI Test Symp.*, 1995, pp. 283–289.

[8] M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Saldanha, and A. Taubin, "Partial-scan delay fault testing of asynchronous circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 17, no. 11, pp. 1184–1199, Nov. 1998.

[9] K. v. Berkel, A. Peeters, and F. t. Beest, "Adding synchronous and LSSD modes to asynchronous circuits," in *Proc. Int. Symp. on Asynchronous Circuits and Systems*, Apr. 2002, pp. 146–155.

[10] M. Roncken, E. Aarts, and W. Verhaegh, "Optimal scan for pipelined testing: An asynchronous foundation," in *Proc. Int. Test Conf.*, Oct. 1996, pp. 215–224.

[11] O. A. Petlin and S. B. Furber, "Scan testing of asynchronous sequential circuits," in *Proc. Great Lakes Symp. on VLSI*, Mar. 1995, pp. 224–229.

[12] F. t. Beest, A. Peeters, M. Verra, K. van Berkel, and H. Kerkhoff, "Automatic scan insertion and test generation for asynchronous circuits," in *Proc. Int. Test Conf.*, Oct. 2002, pp. 804–813.

[13] F. t. Beest, A. Peeters, K. van Berkel, and H. Kerkhoff, "Synchronous full-scan for asynchronous handshake circuits," in *IEEE Eur. Test Workshop*, May 2002, pp. 381–387.

[14] E. B. Eichelberger and T. W. Williams, “A logic design structure for LSI testing,” in *Proc. 14th Design Automation Conf.*, Jun. 1977, pp. 462–468.
[15] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing & Testable Design*. New York: IEEE, 1993.
[16] M. Roncken, “Partial scan test for asynchronous circuits illustrated on a DCC error corrector,” in *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, Nov. 1994, pp. 247–256.
[17] K. van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij, “The VLSI-programming language tangram and its translation into handshake circuits,” in *Proc. Eur. Conf. on Design Automation*, 1991, pp. 384–389.
[18] A. Efthymiou, C. Sotiriou, and D. Edwards, “Automatic scan insertion and pattern generation for asynchronous circuits,” in *Proc. Design Automation and Test in Europe Conf.*, vol. I, Feb. 2004, pp. 672–672.
[19] A. Efthymiou, W. J. Bainbridge, and D. Edwards, “Adding testability to an asynchronous interconnect for GAL’s SoCs,” in *Proc. Asian Test Symp.*, Nov. 2004, pp. 20–23.
[20] D. E. Muller and W. S. Bartky, “A theory of asynchronous circuits,” in *Proc. Int. Symp. on the Theory of Switching*, Apr. 1959, pp. 204–243.
[21] T. Verhoeff, “Delay-insensitive codes—An overview,” *Distributed Computing*, vol. 3, no. 1, pp. 1–8, 1988.
[22] W. J. Bainbridge, W. B. Toms, D. A. Edwards, and S. B. Furber, “Delay-insensitive, point-to-point interconnect using M-of-N codes,” in *Proc. Int. Symp. on Asynchronous Circuits and Systems*, May 2003, pp. 132–140.
[23] L. A. Plana, P. A. Riocreux, W. J. Bainbridge, A. Bardsley, J. D. Garside, and S. Temple, “SPA—A synthesisable amulet core for smartcard applications,” in *Proc. Int. Symp. on Asynchronous Circuits and Systems*, Apr. 2002, pp. 201–210.
[24] K. van Berkel, F. Huberts, and A. Peeters, “Stretching quasi delay insensitivity by means of extended isochronic forks,” in *Proc. Asynchronous Design Methodologies*, May 1995, pp. 99–106.

Aristides Efthymiou (S’93–A’95–M’02) received the B.Sc. and M.Sc. degrees in computer science from the University of Crete, Greece, where he implemented a shared-memory buffer for a high-speed network switch, and the Ph.D. degree in computer science from the University of Manchester, Manchester, U.K., where he worked on power-adaptive, asynchronous processor microarchitecture.

He is a Lecturer in the School of Informatics, the University of Edinburgh, Edinburgh, U.K. His research focus is on the design of robust, high-speed, low-power circuits and microarchitectures, with an emphasis on self-timed circuits and systems.

John Bainbridge (S’98–A’99–M’03) received the M.Eng. degree in electronic systems engineering from Aston University, U.K., and the Ph.D. degree in computer science from the University of Manchester, U.K. His Ph.D. dissertation, “Asynchronous system-on-chip interconnect” won the British Computer Society Distinguished Dissertation award and led onto further exploration of globally asynchronous locally synchronous GALS methods and self-timed Network-on-Chip technology.

He is currently with Silistix Ltd., Manshester, U.K., a venture funded start-up continuing the development and exploitation of this technology.

Douglas Edwards received the B.Sc. degree in physics and electronic engineering from the University of Manchester, Manchester, U.K., and the M.Sc. and Ph.D. degrees studying the properties of Zinc Sulphide Silicon Heterojunctions.

After a period with Ferranti as a process engineer improving the yield of CDI integrated circuits, he joined the School of Computer Science researching high density memory technology, high speed optical networks and hardware accelerators for printed circuit design. He is currently a Senior Lecturer in the School of Computer Science, the University of Manchester, Manchester, U.K. His current research interests are in computer-aided design (CAD) for the synthesis of asynchronous circuits and has led the team which has developed the Balsa synthesis system.