Built-In Self-Testing of Micropipelines

O. A. Petlin, S. B. Furber

Department of Computer Science, The University,
Oxford Road, Manchester, M13 9PL, U.K.

Abstract

An asynchronous ARM6 microprocessor (AMULET1), designed at the University of Manchester using a two-phase signalling protocol, and the latest release of the AMULET2e embedded controller, implemented using four-phase signalling, have proved the practical feasibility of the micropipeline design approach. A built-in self-test (BIST) micropipeline design based on an asynchronous BILBO register is presented in this paper. All the stage registers of the micropipeline are implemented using the proposed asynchronous BILBO register which can operate in four modes: normal operation, shift, linear feedback shift register (LFSR) and signature analyser mode. The test procedure described in this paper provides for the detection of all single stuck-at faults in the micropipeline. It is shown that delay faults in the combinational logic blocks of the BIST micropipeline can be tested by using BILBO registers of a doubled size.

1: Introduction

Asynchronous circuits have a potential for low power consumption, promise greater design modularity and exhibit typical case performance rather than worst-case performance [Lav93, Birt95, Brzo95]. The AMULET group at the University of Manchester has been developing asynchronous engineering design methodologies which have already demonstrated the practical feasibility of complex asynchronous VLSI systems. The first asynchronous ARM6 microprocessor (AMULET1) has been designed by the AMULET group and fabricated by GEC Plessey Semiconductors Limited. The AMULET1 microprocessor was implemented using the micropipeline approach based on two-phase signalling [Furb94, Pav94]. The experience gained from the AMULET1 design led to the AMULET2e design, an asynchronous embedded controller with an AMULET2 core. The AMULET2 microprocessor was built using the micropipeline design style with four-phase signalling [Furb97]. However, before the advantages of asynchronous circuits can be exploited commercially, it must be shown that they can be tested effectively for fabrication faults in volume production [Hulg94].

2: Micropipelines

Micropipelines were described by Sutherland in his Turing Award lecture [Suth89]. The micropipeline design approach is based on three fundamental principles:

- the pipelined organization of the computation;
- transition signalling;
- the bundled-data interface.

The data stream moves through the micropipeline stages controlled by signal transitions. Each micropipeline stage can operate as either a sender or receiver. When the data is ready to be transmitted the sender produces a ‘request’ event to the receiver; the receiver accepts the data and sends an ‘acknowledge’ event to the sender (this is a ‘bundled data’ interface). This handshake protocol is repeated every time the next data produced by the sender is ready to be transmitted.

There are two basic signalling protocols: two-phase and four-phase. Micropipelines introduced by Sutherland operate according to the two-phase protocol where both rising and falling transitions are events and have the same meaning. Thus, when the sender is ready to send the data, a rising or falling request event is sent to the receiver which acknowledges the receipt of the data by generating a rising or falling acknowledge event. In four-phase signalling both request and acknowledge signals must be reset before new data can be transmitted.

2.1: Micropipeline structures

Figure 1 illustrates a three-stage micropipeline with processing [Suth89, Birt95]. In the initial state all the event registers of the micropipeline are transparent. A data value is sent to the left event register (*Reg1*) by the environment. Once a request event is generated on control line *Rin* the data is copied into register *Reg1* which then signals events on its *Rout* and *Ain* outputs. In this state event register *Reg1* holds the data stable until it receives an acknowledge signal on its *Aout* input. The request signal generated by register *Reg1* is delayed for enough time to allow the data on the outputs of the following logic block (*Logic1*) to be stable. After receiving a request signal on input *Rin* the second event register (*Reg2*) latches the data, acknowledges it by signalling an event on its *Ain* output and generates a request signal on its *Rout* output for the next event register. As a result, event register *Reg1* is set to the transparent mode where it is ready to accept new data from the environment. The data processing procedure described above is repeated for the rest of the micropipeline stages. The output data produced by the micropipeline can be read by the environment when a request signal is generated on its *Rout* output. Once the output data is latched an acknowledge signal is sent to input *Aout* of the micropipeline. Every micropipeline stage works in parallel and sends data to the neighbouring stage only when the data is ready to be processed.

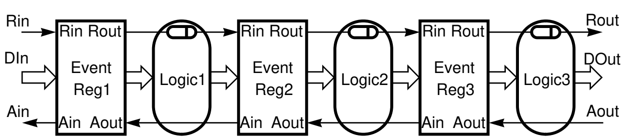

Figure 1: *A micropipeline with processing*

2.2: Micropipeline latch control

There are different ways to control the latching and storing of the data in the micropipeline registers. For example, event-controlled latches described by Sutherland are controlled by a pair of control signals such as "pass" and "capture" [Suth89].

In the initial state all the register latches can be either transparent or in the capture mode depending on the latch transition controlling protocol. Figure 2 illustrates the design of the event latch widely used in the AMULET1 microprocessor [Furb94, Day95, Birt95]. The design of the

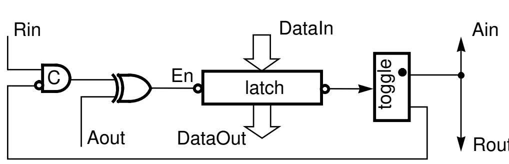

Figure 2: *AMULET1 event latch structure*

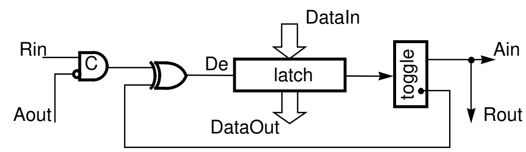

Figure 3: *Two-phase control for a normally closed latch*

latch follows the two-phase transition signalling protocol. The XOR gate together with the toggle element converts the two-phase signalling into a four-phase protocol. This is because the latch is level-sensitive and is transparent when the *En* signal is low and opaque when *En* is high.

The use of 'normally closed' latches is preferable from the power consumption point of view since no transitions in the data paths can occur unless new data has been latched by the stage register [Furb94, Birt95]. Figure 3 shows an implementation of the normally closed latch structure which uses two-phase signalling.

Figure 4 illustrates an n-type single-phase latch which can be used for storing the data in the micropipeline registers. The latch is transparent when the data enable input (*De*) is high and it is opaque when the data enable input is low. When transparent, input data is propagated through the latch to its output. Once the enable signal is reset the data is prevented from flowing to the inverter and the weak data retention circuit. As a result, the data is stored and any signal changes on the data input will have no effect on the stored data. Single-phase latch designs are investigated elsewhere [Yuan89].

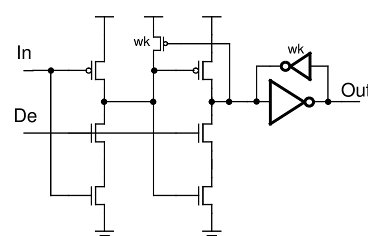

Figure 4: *Single-phase static latch*

3: Testing micropipelines

3.1: Fault model

A number of works have already addressed the testing of micropipelines for single stuck-at faults [Pag92, Khoc94, Pet95a, Pet95b]. The stuck-at fault model is widely used to describe the fault behaviour of the circuit under test. According to this fault model a circuit line is stuck-at one or zero if it is disconnected from any other circuit's wires and connected to the power supply or ground respectively.

3.2: Faults in micropipelines

Stuck-at faults in micropipelines can be divided into three classes [Pag92]:

- faults in the latch control part;
- faults in the combinational logic blocks;
- faults in the latches.

Faults in the latch control

Stuck-at output faults in the latch control circuits (see Figures 2 and 3) cause the faulty micropipeline to halt. The micropipeline moves through at most one step and then halts in the presence of a stuck-at input fault in its control part. Thus, such stuck-at faults can be identified easily during normal operation mode.

Faults in the processing logic

It was assumed that all the latches of the micropipeline are transparent initially [Pag92]. This allows the processing logic to be treated as a single combinational circuit. To detect any of the single stuck-at faults in such a circuit test vectors can be obtained using any known test generation technique [MClus86, Russ89].

Therefore, the test procedure for the micropipeline consists of two major steps:

1) the micropipeline is emptied, i.e. all the latches are transparent;

2) the test vectors are applied to the inputs of the micropipeline and the responses of the combinational logic blocks are compared with good responses.

This test algorithm has been adapted to the detection of stuck-at faults in the processing logic of a micropipeline with normally closed latches [Pet95a, Pet95b].

Faults in the latches

Single stuck-at faults. Any stuck-at fault on the input or output of a latch is equivalent to a corresponding fault in the combinational logic.

Single stuck-at-capture faults. A single stuck-at-capture fault in a latch causes a register bit of the latch to remain permanently in the capture position. For example, a stuck-at-0 fault on the *De* input of the latch shown in Figure 3 sets the faulty latch in capture mode permanently. As an effect of this fault, the faulty bit can be detected as a constant logic one or zero. When all the latches of the micropipeline are transparent this fault is equivalent to an appropriate stuck-at fault on a line of the combinational logic. Thus, stuck-at-capture faults can easily be detected using standard tests for stuck-at faults in combinational networks.

Single stuck-at-pass faults. These faults set a register bit of a latch in pass mode permanently. A stuck-at-1 fault on the *De* input of the latch illustrated in Figure 3 makes the faulty latch transparent permanently. It has been shown that a two pattern test is required to detect this kind of fault [Pet95a, Pet95b].

3.3: Scan testing of micropipelines

An elegant scan test approach has been proposed by Khoche and Brunvand [Khoc94]. The micropipeline can act in two modes: normal operation and scan test mode. The micropipeline performs to its specification in normal operation mode. In test mode, all the latches are configured into one shift register where each latch works as an ordinary master-slave flip-flop. The stage registers of the micropipeline are clocked through the control lines where the *Aout* input is used as a clock input. The C-elements pass their negated inputs onto the outputs forming a clocking line for the scan path. As a result, the test patterns are loaded from the scan-in input into all the latches of the micropipeline. Afterwards the micropipeline is returned to normal operation mode in which only one request signal is generated. To observe the contents of the register latches the micropipeline is set to scan test mode. The contents of all the latches are shifted out to the scan-out output. The proposed test technique allows the detection of all the stuck-at faults and bundling constraint violations in micropipelines [Khoc94]. The proposed scan test interface uses clocks produced by a clock generator which is not always available in asynchronous VLSI designs.

A method to design and test micropipelines and sequential circuits based on the micropipeline design style has been reported [Pet95a, Pet95b]. The test approach is implemented using specially designed scan latches manipulated by the scan test control logic. The proposed scan test procedure allows single stuck-at and delay faults in the combinational logic blocks to be detected.

However, the results reported so far do not address the built-in self-testing (BIST) of micropipeline structures. In this paper, an asynchronous BIST micropipeline design based on the built-in logic block observation (BILBO) approach is considered.

4: Asynchronous BILBO register design

In BIST VLSI designs the test patterns are generated by a circuit included on the chip and the response analysis is also fulfilled by on-chip circuitry. There are two possible realizations of self-testing: *InSitu* and *ExSitu* self-testing [Russ89]. *InSitu* self-test structures use system registers to

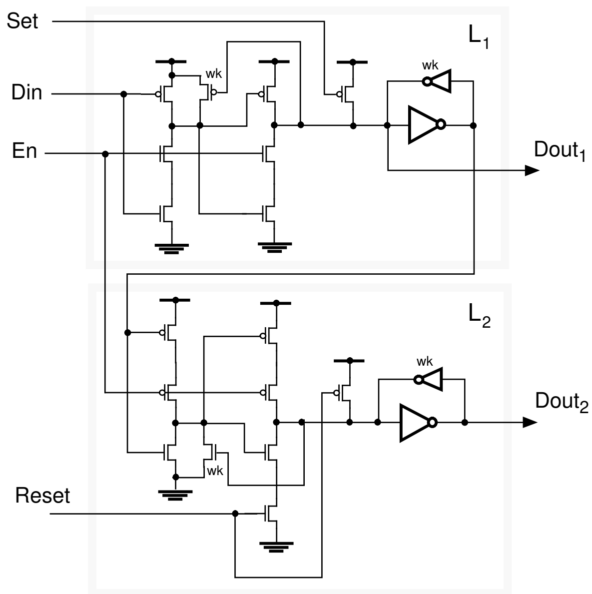

Figure 5: *CMOS implementation of the BILBO register latch*

generate and compact test data whereas the *ExSitu* design uses registers external to the system function to generate tests and analyse the responses of the circuit.

A classical example of *InSitu* self-test is the BILBO technique [MClus86, Russ89]. This technique is based on the use of a BILBO register which can be reconfigured to act as a pseudo-random pattern generator or as a signature analyser within a VLSI circuit. The BILBO technique uses signature analysis in conjunction with a scan path technique.

The CMOS implementation of one of the latches from which an asynchronous BILBO register may be built is illustrated in Figure 5. The latch design consists of two latches L_1 and L_2 connected together. The implementation of L_1 is similar to the single-phase static latch shown in Figure 4. In addition, latch L_1 has an active low *set* input to set the $Dout_1$ output to high. Latch L_2 is a single-phase static latch which is transparent when its enable input is low otherwise it is opaque. L_2 has an active low *reset* input which holds its output at zero regardless of whether it is transparent or opaque.

Initially latch L_1 is closed and L_2 is open due to a low signal applied to the *En* input. When the data is stable on the *Din* input of L_1 the *En* input is set high, making the latch L_1 transparent and L_2 opaque. The input data stored in the memory of L_1 is passed to the $Dout_1$ output and to the data input of L_2. When the enable input is reset, latch L_1 is closed and L_2 is transparent, storing the data from L_1 in its memory. As a result, the data stored in L_1 is passed to the $Dout_2$ output of the register latch. The procedure of storing

Table 1: *Operation modes of the BILBO register*

C1	C2	Set	Reset	Mode
1	0	1	0	Normal
0	0	1	1	Shift
0	1	0→1	1	LFSR
1	1	0→1	1	SA

data in the register latch is similar to that of a master-slave flip-flop. Note that when *En*=0 the $Dout_1$ output can be set high by an active low signal applied to the *set* input of L_1.

An asynchronous implementation of a 4-bit BILBO register is illustrated in Figure 6. The Q_i outputs of the L_{i1} latches (i=0,1,2,3) are used as the outputs of the BILBO register. The $Dout_2$ outputs of the register latches (see Figure 5) are connected through the XOR gates to the *Din* inputs of the corresponding latches. The *scan-in* input (*Sin*) is coupled through the multiplexer (MX) and one of the inputs of the XOR gate to the *Din* input of the first register latch. The $Dout_2$ output of the last register latch is used as a scan output from the register. The register latches can be enabled by *scan* and *data enable* clocks applied to the *Sc* input of the register and the *De* input of the OR gate respectively. The *De* signals are generated by the latch control circuit which is similar to the one shown in Figure 3.

The BILBO register shown in Figure 6 can act in four distinct operation modes depending on the control signals *C1* and *C2*: normal, shift, LFSR and signature analyser operation modes. Table 1 contains the values for the control signals and the corresponding operation modes of the BILBO register.

Normal operation mode. In this mode *C1*=1 and *C2*=0. As a result, the *reset* signal is set low holding the outputs of

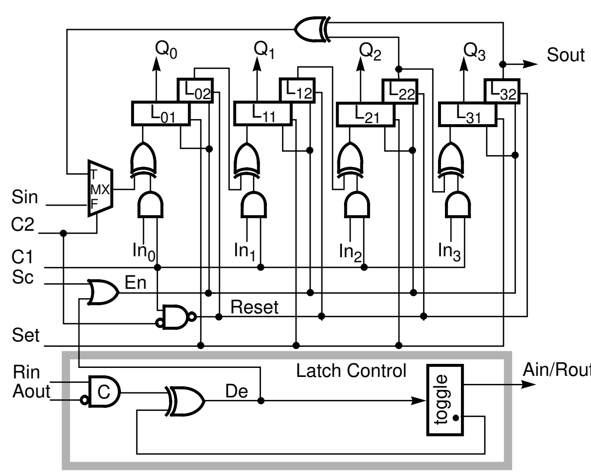

Figure 6: *Asynchronous BILBO register structure*

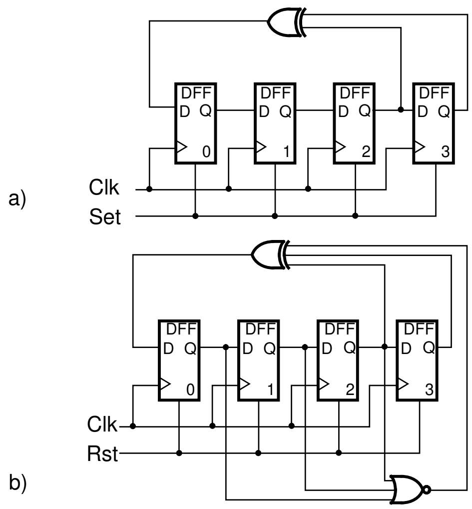

Figure 7: *Four-bit pseudo-random pattern generator*

latches L_{i2} (i=0,1,2,3) at zero permanently. The *set* input is high. The inputs *Sin* and *Sc* are kept low during this operation mode.

When the input data is stable on the *In* inputs of the register the control inputs of the latch control block are activated by the environment. Thus, clock signals are produced on the *De* output by the latch control circuit. The data is passed to the outputs of the L_{i1} latches and latched in their memories.

Shift register mode. Control signals *C1* and *C2* are reset in the shift register operation mode of the BILBO register. In this mode the *set* and *reset* signals are set to high. The register latches act as D-type flip-flops configured in a 4-bit shift register. The shift register is clocked from the *Sc* input. Note the latch control block is not active in this mode, holding its *De* output at zero permanently.

When the scan data is ready on the *Sin* input it is transmitted through the multiplexer to the input of the first flip-flop. A clock signal is applied to the *Sc* input of the register. As a result, the scan data is stored in the first flip-flop and passed to the input of the second flip-flop. The next bits of the scan data are shifted into the register latches in the same manner as a normal shift register. The shift register passes the scan data on its *Sout* output from where the data is read by the environment.

LFSR operation mode. The BILBO register operates as an LFSR when *C1*=0 and *C2*=1. In this mode, the *reset* signal is set high and the *Sc* input is reset. Note that the data from the *In* inputs of the BILBO register is blocked by a zero signal applied to its *C1* input (the outputs of the corresponding AND gates are reset). Thus, the BILBO register (see Figure 6) can perform as the 4-bit LFSR illustrated in Figure 7a. This LFSR is designed using the following primitive polynomial $\varphi(X) = 1 + X^3 + X^4$. The pseudo-random sequences of maximal period $(M = 15)$ are generated on the register outputs.

Initially, the *set* input of the register is reset, setting the Q_i outputs and the outputs of latches L_{i2} (i=0,1,2,3) high. When the register outputs are stable the *set* input is set high. As a result, the 'all ones' state is the initial state of the LFSR. The LFSR generates a new output vector every time a new clock signal is produced by the latch control block on its *De* output. Note that the state of 'all zeros' is illegal for the LFSR shown in Figure 7a since it stays in this state forever. A NOR gate can be added to the design of the LFSR in order to allow it to go through all its possible states (see Figure 7b) [MClus86].

Signature analyser operation mode. When the control signals *C1* and *C2* are set to high the BILBO register can operate as a signature analyser. In this mode, the *reset* signal is set high and the *Sc* input is reset permanently. Note that the data from the *In* inputs of the register is enabled by a high signal applied to the *C1* input.

In the initial state, the outputs of the signature analyser are set high by a low signal applied to the *set* input of the BILBO register. When the outputs of the signature analyser are stable the *set* signal is set high. Once the data is ready on the *In* inputs, the signature analyser is clocked from the *De* output of the latch control block. As a result, the current contents of the register are mixed with the new input data.

The design of the asynchronous BILBO register shown in Figure 6 is similar to the synchronous BILBO design. As a consequence, the testability properties of the asynchronous BILBO register with respect to stuck-at faults are the same as that of the synchronous one. Note that all stuck-at faults in the BILBO register, including faults on either the control lines or its data paths, can be detected when the register is forced to perform in all its operation modes. Stuck-at faults in the latch control circuit (see Figure 6) manifest themselves by causing the circuit to halt [Pet95a].

5: Micropipeline structure with BIST features

Figure 8 shows a two-stage micropipeline structure with BIST features. The stage registers of this micropipeline are built from the register illustrated in Figure 6. The outputs of the micropipeline can be connected to its inputs depending on the Boolean *Bist* which is high in BIST mode and low in normal operation mode. The BIST control unit is activated by a high signal applied to the *Bist* input of the micropipeline allowing the BIST control signals to be generated.

The *RSin*, *ASin*, *RSout* and *ASout* control signals are

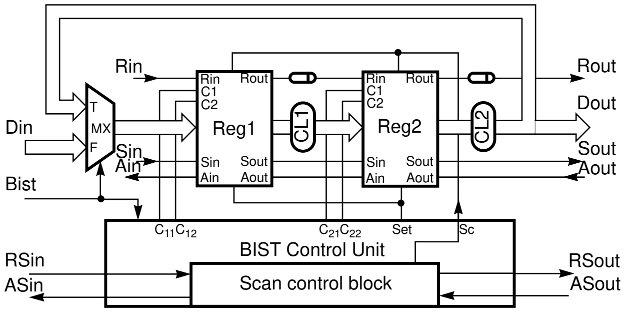

Figure 8: *A two-stage micropipeline with BIST features*

used in an asynchronous interface to shift the data in and out of the stage registers. An implementation of the scan control block is shown in Figure 9. Note that the scan control block is initialised using a global reset signal which is not shown for the sake of simplicity.

Normal operation mode

The micropipeline is set to normal operation mode when *Bist*=0 and *RSin*=*ASout*=0. As a result, the control signals C_{i1} and C_{i2} (i=1,2) of the control unit are set high and low respectively. Its *Set* and *Sc* outputs are also set high and low respectively. In this mode the micropipeline acts in the manner described in section 2.

BIST mode

In BIST mode the Boolean signal *Bist* is set high, enabling the control unit to produce control signals for the stage registers of the micropipeline.

The micropipeline shown in Figure 8 can be tested for stuck-at faults using the following test algorithm:

1. *Testing for stuck-at faults in the register latches involved in the shift operation.* The shift operation is tested by setting the stage registers into the shift register mode and applying an alternating test to the *Sin* input of the micropipeline. Since the stage registers are united in one shift register, the alternating test is passed through all the register latches. Note that the shift operation is controlled by clock signals generated on the *Sc* output of the scan control block.

2. *Testing for stuck-at faults in the combinational circuits*. The combinational circuit *CL1* is tested when register *Reg1* is set to the LFSR mode and *Reg2* is set to the signature analyser mode. The latches of *Reg1* and *Reg2* are set to high initially (see Figure 6). After the generation of a request signal on the *Rin* input of the micropipeline, the LFSR applies a new random test vector to the inputs of *CL1* and its responses are collected by the signature analyser. The *Rin* input is triggered enough times for the LFSR to generate the required number of random test vectors on the inputs of *CL1*. The number of random tests can be calculated using known expressions reported elsewhere [Savir84, Wag87]. The combinational circuit *CL2* is tested in the way described above with *Reg2* acting as the LFSR and *Reg1* as the signature analyser. The responses from *CL2* are passed through the multiplexer to the inputs of *Reg1*. Note that the signatures produced by the signature analysers are shifted out to the *Sout* output of the micropipeline every time the testing of each combinational circuit has completed.

3. *Testing for stuck-at faults on the Din inputs and Dout outputs*. Stuck-at faults on the *Dout* outputs of the micropipeline can be tested by observing the responses from *CL2* during its testing. Stuck-at faults on the *Din* inputs are tested in normal operation mode (*Bist*=0). In this mode, the 'all zeros' and 'all ones' tests are applied to the *Din* inputs of the micropipeline (two request signals are applied to the *Rin* input). Note that the contents of *Reg1* are shifted out after the application of each test.

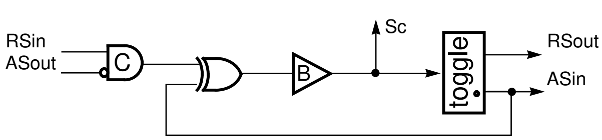

Figure 9: *Scan control block*

The use of the above test algorithm allows most of the stuck-at faults in the micropipeline to be detected. Note that stuck-at-1 faults on the *reset* inputs in the register latches (see Figure 5) cannot be identified in test mode since the *reset* signal is set to low only in normal operation mode. Thus, these faults can be tested in normal operation mode as follows:

1. Set the micropipeline in normal operation mode.
2. Set all the latches of the micropipeline to high (*Set*=1).
3. *Set*=0 and apply a test to the *Din* inputs of the micropipeline.
4. Generate one request signal on the *Rin* input.
5. Shift the contents of the stage registers out to the *Sout* output.

In the presence of a stuck-at-1 fault on the *reset* input in the i-th latch of the j-th (j=1,2) stage register the response bit latched in its (i+1)-st latch will be inverted (see Figure 6).

6: Testing for delay faults

Data is latched in each micropipeline stage register after a certain delay matching the signal propagation delay through the longest path in the corresponding combina-

Table 2: *State sequence of the 4-bit LFSR*

State	Q_0	Q_1	Q_2	Q_3	T_0	T_1
0	0	0	0	0	0	0
1	1	0	0	0	0	0
2	0	1	0	0	1	0
3	0	0	1	0	0	0
4	1	0	0	1	0	1
5	1	1	0	0	1	0
6	0	1	1	0	1	0
7	1	0	1	1	0	1
8	0	1	0	1	1	1
9	1	0	1	0	0	0
10	1	1	0	1	1	1
11	1	1	1	0	1	0
12	1	1	1	1	1	1
13	0	1	1	1	1	1
14	0	0	1	1	0	1
15	0	0	0	1	0	1

tional block. In the presence of a delay fault in the combinational circuit the corresponding path delay is extended. As a result, the output data of the faulty combinational circuit can be latched by the corresponding micropipeline stage register when it is not yet stable, violating the bundled-data constraint. Thus, the latched data may or may not be correct depending on the particular change in the propagation signal delay along the faulty path.

In principle, a pair of test vectors applied sequentially to the inputs of the combinational circuit are required to detect a delay fault [Pram90]. The first test settles the combinational circuit and the second one activates its faulty path, propagating a falling or rising event through it. The circuit's response is latched from the outputs of the combinational circuit after a specified time. If the latched outputs do not match the correct ones then the circuit is faulty. Using the micropipeline structure shown in Figure 8 delay faults in its combinational circuits can be tested.

Let us consider the LFSR shown in Figure 7b. The output sequence of this LFSR has the following property:

> All possible pairs of 2-bit binary vectors can be found sequentially on the odd or even outputs of the LFSR [Pet94].

Table 2 contains the states of the 4-bit LFSR shown in Figure 7b. Columns T_0 and T_1 repeat the outputs Q_1 and Q_3 respectively. After the period of the LFSR all the possible combinations of 2-bit vectors can easily be found in the 2-bit output sequence (columns T_0 and T_1). For instance, the combinations: 11 11, 11 01, 11 10, 11 00 can be found in this sequence.

Lemma. Let the pseudo-random pattern generator be built using the $(k \times n)$-bit LFSR which goes through all possible binary states. Every *k*-th output of the LFSR is used as an output of the generator so that it has exactly *n* outputs. All possible combinations of any *k* *n*-bit binary vectors chosen sequentially can be found in the output sequence of the pseudo-random pattern generator after it has passed through all its states.

Proof. The prove of this Lemma is trivial for *k*=1 since the LFSR is assumed to produce all possible $(k \times n)$-bit binary vectors on its outputs.

Let us prove this Lemma when *k*=2. Hence, the LFSR has 2*n* outputs.

Let the even outputs of the LFSR be the outputs of the generator. Figure 10a shows the mechanism for generating pseudo-random vectors on the even outputs of the LFSR. Let us choose a certain pair of *n*-bit vectors. The first vector from this pair of vectors is read directly from the even outputs of the LFSR after the application of a clock signal at time *t* (see Figure 10a). Since the LFSR acts as a shift register the second vector is shifted from its odd outputs after it is clocked at time *t*+1. As a result, there is a unique 2*n*-bit vector which must be generated by the LFSR in order to produce the required pair of vectors. Since the LFSR can go through all possible 2^{n+k} states the required vector can be derived. This proof can be repeated for any other pairs of *n*-bit vectors.

The mechanism for generating pseudo-random vectors on the odd outputs of the LFSR is illustrated in Figure 10b. Let us fix a certain pair of *n*-bit vectors. The first vector from this pair is produced on the odd outputs of the LFSR

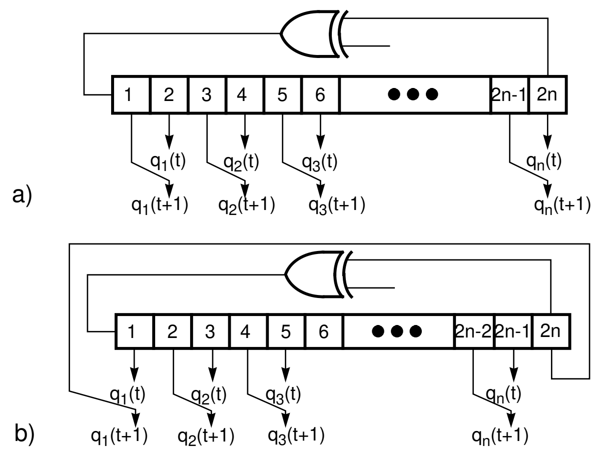

Figure 10: *Mechanisms for generating pseudo-random vectors using a) even and b) odd outputs of the LFSR*

at time *t*. After the application of the next clock the contents of the even flip-flops of the LFSR are shifted into the odd ones (see Figure 10b) producing the second vector. The content of the first output is derived by XORing the outputs of some flip-flops including the last one. Note that the number of inputs of the XOR gate and the flip-flops which feed its inputs depend on the derivation polynomial of the LFSR. Regardless the outputs of those flip-flops which are connected to the inputs of the XOR gate, the content of the first flip-flop can be inverted (when the output of the last flip-flop is a one) or can be unchanged (when the output of the last flip-flop is a zero). As a result, there is a unique state of the LFSR which allows the required pair of vectors to be generated on its odd outputs.

A similar proof can be continued easily for any k more than 2. ❑

Thus, the result of the Lemma can be used for testing delay faults in the combinational circuits of the BIST micropipeline. For this purpose the number of latches in the BILBO registers (see Figure 6) must be doubled by adding one extra D-type flip-flop after each register latch. Note that the outputs of the extra flip-flops are held at zero during normal operation mode. The linear feedback of the BILBO register must be changed according to the new derivation polynomial for the LFSR with the double length. As a result, the BILBO register operating in the LFSR mode generates all possible pairs of binary combinations on the inputs of the corresponding combinational circuit. After the period of such a LFSR all possible paths inside the combinational circuit can be tested which, in turn, guarantees the detection of bundled-data violations in the corresponding stage of the BIST micropipeline.

7: Analysis of the BIST micropipeline structure

The BIST technique for micropipelines presented in this paper has advantages and disadvantages.

Advantages

- The BIST micropipeline structure shown in Figure 6 allows the generation of random tests and the collection of the test results on the chip.
- The combinational circuits of the micropipeline are tested separately at the normal circuit speed, making possible the application of a large number of tests to their inputs.
- The asynchronous BILBO register design described in this paper has the same properties as that of its synchronous counterpart. As a result, the BIST control unit can be implemented in a similar way to the one for the synchronous BIST design.
- The BIST micropipeline design allows the testing of its combinational circuits either for stuck-at or delay faults.

Disadvantages

- The implementation of the micropipeline with BIST features requires the use of BILBO registers. As a consequence, the BIST version of the micropipeline imposes a certain degree of area overhead which depends on the complexity of its combinational part.
- The performance of the BIST micropipeline is degraded in normal operation mode since some extra logic elements are added in its data paths. For instance, the input data arriving at the *In* inputs of the BILBO register (see Figure 6) comes through extra AND and XOR gates before being latched in the L_{i1} latches. These extra delays must be taken into account to ensure the proper bundled data interface.

8: Conclusions

A micropipeline design with BIST features has been introduced in this paper. The proposed BIST micropipeline design is based on the well-known BILBO register of which asynchronous implementation has been discussed. The use of the asynchronous BILBO register allows stuck-at faults inside both the combinational logic blocks and stage registers of the micropipeline to be detected. It has been shown that doubling the size of the BILBO register provides for the detection of both stuck-at and delay faults in the combinational circuits of the micropipeline. Finally, the proposed micropipeline with BIST features can be used in asynchronous VLSI circuits where the pseudo-random pattern generator and the signature analyser are placed on the chip.

Acknowledgements

The authors would like to express their gratitude to John Brzozowski for his useful comments and discussions on the topic of this paper.

References

[Birt95] G. Birtwistle, A. Davis (Eds), "Asynchronous digital circuit design", Springer, 1995.

[Brzo95] J. A. Brzozowski, C-J. H. Seger, "Asynchronous circuits", Springer-Verlag New York, Inc., 1995.

[Furb94] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, J. V. Woods, “AMULET1: A micropipelined ARM”, Proc. IEEE Computer Conf., March 1994.

[Furb97] S. B. Furber, J. D. Garside, S. Temple, J. Liu, P. Day, N. C. Paver, “AMULET2e: An asynchronous embedded controller”, Int. Symposium on Advanced Research in Asynchronous Circuits and Systems (Async97), Netherlands, April 7-10, 1997.

[Day95] P. Day and J. V. Woods, “Investigation into micropipeline latch design styles”, IEEE Trans. VLSI Systems, vol. 3, no. 2, June 1995, pp. 264-272.

[Hulg94] H. Hulgaard, S. M. Burns, G. Borriello, “Testing asynchronous circuits: A survey”, TR-FR-35, Department of Computer Science, University of Washington, Seattle, WA, USA, 1994.

[Khoc94] A. Khoche, E. Brunvand, “Testing micropipelines”, Proc. Int. Symposium on Advanced Research in Asynchronous Circuits and Systems (Async94), Utah, Nov. 1994, pp. 239-246.

[Lav93] L. Lavagno, A. Sangiovanni-Vincentelli, “Algorithms for synthesis and testing of asynchronous circuits”, Kluwer Academic Publishers, 1993.

[MClus86] E. J. McCluskey, “Logic design principles: with emphasis on testable semicustom circuits”, Prentice/Hall International Inc., 1986.

[Pag92] S. Pagey, G. Venkatesh, S. Sherlekar, “Issues in fault modelling and testing of micropipelines”, First Asian Test Symposium, Hiroshima, Japan, Nov. 1992.

[Pav94] N. C. Paver, “The design and implementation of an asynchronous microprocessor”, PhD Thesis, University of Manchester, Manchester, UK, June 1994.

[Pet94] O. A. Petlin, “Random testing of asynchronous VLSI circuits”, MSc Thesis, University of Manchester, 1994.

[Pet95a] O. A. Petlin, S. B. Furber, “Scan testing of asynchronous sequential circuits”, Proc. 5th Great Lakes Symposium on VLSI, New York, March 1995, pp. 224-229.

[Pet95b] O. A. Petlin, S. B. Furber, “Scan testing of micropipelines”, Proc. 13th IEEE VLSI Test Symposium, Princeton, New Jersey, USA, May 1995, pp. 296-301.

[Pram90] A. Pramanick, S. Reddy, “On the design of path delay fault testable combinational circuits”, Proc. 20th Fault Tolerant Computing Symp., June 1990, pp. 374-381.

[Russ89] G. Russell, I. L. Sayers, “Advanced simulation and test methodologies for VLSI design”, Van Nostrand Reinhold (International), 1989.

[Savir84] J. Savir, P. H. Bardell, “On random pattern test length”, IEEE Trans. on Computers, vol. C-33(6), June, 1984, pp. 467-474.

[Suth89] I. E. Sutherland, “Micropipelines”, Communications of the ACM, Vol. 32, no. 6, June 1989, pp. 720-738.

[Wag87] K. D. Wagner, C. K. Chin, E. J. McCluskey, “Pseudorandom testing”, IEEE Trans. on Computers, C-36(3), March, 1987, pp. 332-343.

[Yuan89] J. Yuan, C. Svensson, “High-speed CMOS circuit technique”, IEEE Journal on Solid-State Circuits, vol. 24, no. 1, Feb., 1989, pp. 62-70.