Design for Testability of Asynchronous VLSI Circuits

A thesis submitted to
the University of Manchester
for the degree of
Doctor of Philosophy
in the Faculty of Science and Engineering

Oleg Alexandrovich Petlin

Department of Computer Science

1996

Contents

Chapter 6 : Design for Random Pattern Testability of Asynchronous Circuits 127

Chapter 7 : Design for Testability of an Asynchronous Adder 177

List of Figures

List of Tables

Abstract

Asynchronous design methodologies are a subject of growing research interest since they appear to offer benefits in low power applications and promise greater design modularity. However, before these advantages can be exploited commercially, it must be shown that asynchronous circuits can be tested effectively in volume production. This thesis presents the results of research into various aspects of the design for testability of asynchronous circuits.

Low power is often achieved by minimising circuit activity. However, testable designs require high transition probabilities. It is shown that design for testability and design for low power are in direct conflict. As a result, the more testable a circuit is, the more power it consumes. The resolution of this conflict can be found in the separation of normal operation and test modes. In test mode the circuit activity is increased, dissipating more power.

Many asynchronous designs use Muller C-elements in a large variety of applications including both control and data paths. Testable CMOS designs for C-elements are presented which provide for the detection of transistor stuck-at and stuck-open faults.

The scan test technique is used to test stuck-at and delay faults in micropipelines. This technique is generalised to the design for testability of either two-phase or four-phase micropipelines. An asynchronous built-in self test (BIST) micropipeline design based on the BILBO technique is presented. The proposed design for the BILBO register allows stuck-at and delay faults to be detected inside the combinational circuits of the micropipeline.

Structural designs for random pattern testability techniques applicable to asynchronous sequential circuits are described. The proposed random test procedure provides for the detection of all single stuck-at faults in the control and data paths of the sequential circuit under test, reducing the overall test complexity to the testing of its combinational network.

Case studies of testable implementations of some high-level asynchronous functions, including an adder and a block sorter, are analysed for their testability, performance and area cost. These designs show that, as expected, there is a trade-off to be made between testability and cost. However, satisfactory testability can be achieved for a circuit designed with a small area overhead for test circuitry and little performance degradation.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or institute of learning.

Copyright and the ownership of intellectual property rights

Copyright in text of this thesis rests with the Author. Copies (by any process) either in full, or of extracts, may be made **only** in accordance with instructions given by the Author and lodged in the John Rylands University Library of Manchester. Details may be obtained from the Librarian. This page must form part of any such copies made. Further copies (by any process) of copies made in accordance with such instructions may not be made without the permission (in writing) of the Author.

The ownership of any intellectual property rights which may be described in this thesis is vested in the University of Manchester, subject to any prior agreements to the contrary, and may not be made available for use by third parties without the written permission of the University, which will prescribe the terms and conditions of any such agreement.

Further information on the conditions under which disclosures and exploitation may take place is available from the Head of Department of the Department of Computer Science.

Acknowledgements

During the years which I have spent in the Department of Computer Science at the University of Manchester I have received a great deal of help from many people, without which the work described in this thesis would not have been achievable.

My supervisor Professor Steve Furber has been a great source of inspiration and moral support. I would like to express my gratitude for his constant interest in my ideas and research results and for correcting and commenting on drafts of this thesis.

The other members of the AMULET research group have created a friendly and comfortable atmosphere in which I have carried out my research. I am grateful to God that I had the opportunity to conduct my research in the AMULET group which for me was a rich source of ideas and vital material for the realization of my research results. I would like to thank all of those people who have spent their valuable time to help me to understand asynchronous VLSI design techniques.

I am especially grateful to Phil Endecott for his help and useful tips with the *FrameMaker* design environment. Many thanks to Craig Farnsworth whose invaluable practical recommendations and consultations with the *Cadence* design environment and the *SIMIC* design verification tools made it possible to implement my ideas in practice.

Some people have helped by reading and commenting on drafts of this thesis as well. I must express my thanks to Phil Endecott and David Gilbert for their useful comments.

The Author

Oleg Petlin obtained an Engineering degree (1989) and a Candidate of Technical Sciences degree (1993) in Computer Science from Kiev Polytechnical Institute (Ukraine). In 1994 he received a Master of Science degree in Computer Science from the University of Manchester.

This thesis is the result of three years of research as a member of the AMULET research group at the University of Manchester. AMULET (Asynchronous Microprocessors Using Low Energy Techniques) comprises several projects looking at different areas where asynchronous logic techniques can be applied.

To my mother

Chapter 1 : Asynchronous VLSI Circuits

This chapter starts by describing the major advantages of using asynchronous VLSI designs and provides an overview of the asynchronous design methodologies and techniques developed so far. Subsequent sections present descriptions of the micropipeline and handshake circuit design methodologies which are the basic asynchronous design styles used in the work described in this thesis. The importance of developing asynchronous design for testability (DFT) techniques is discussed. Finally, the last section contains a thesis overview describing the structure of this thesis and the results published by the author.

1.1 Asynchronous VLSI circuits

Very Large Scale Integration (VLSI) circuits designed using modern Computer-Aided Design (CAD) tools are becoming faster and larger, incorporating millions of smaller transistors on a chip [Rice82, Russ87, Weste93]. VLSI designs can be divided into two major classes: synchronous and asynchronous circuits. Synchronous circuits use global clock signals which are distributed throughout their subcircuits to ensure correct timing and to synchronize their data processing mechanisms. Asynchronous circuits contain no global clocks. Their operation is controlled by locally generated signals [Mead80].

1.1.1 Motivation for using asynchronous circuits

A resurgence of interest in the design of asynchronous circuits has been stimulated by their potential advantages compared to their synchronous counterparts:

- *The absence of the clock skew problem.* The largest problem with clocking in VLSI circuits lies in distributing the clock at the same instant to all clocked elements across the chip. “Clock skew” describes the phenomenon whereby different parts of the VLSI system see the clock at slightly different times due to delay variations in the clock interconnections. Clock distribution schemes which minimise the clock-skew window become more and more costly in modern VLSI designs. This is because modern state of art VLSI technology tends to use smaller transistors in larger chips which increases the importance of physical delays along wires in a chip rather than signal delays through transistors. For instance, the clock driver circuitry in the DEC Alpha microprocessor occupies about 10% of the chip area [Dob93]. In asynchronous circuits, the clock skew problem no longer exists since they do not use synchronization clocks to control their operation.

- *Performance*. The fixed clock period in synchronous circuits is chosen using worst-case performance analysis. As a consequence, synchronous circuits perform at their worst-case rates. In asynchronous circuits, the communication between separate blocks on the chip occurs when the data is ready to be transmitted. As a result, asynchronous designs can exhibit typical case performance rather than worst-case performance.

- *Power consumption*. The power consumption of VLSI circuits is important in portable digital systems since a design objective is to maximize the life of lightweight battery packs. All parts of a synchronous VLSI design are clocked even if they do not produce “useful” results. In asynchronous circuits, only those parts of the circuit which produce meaningful results are involved in the computation process. As a result, the use of asynchronous circuits can lead to lower power consumption [Birt95].

- *Timing and design flexibility.* If a synchronous VLSI circuit is required to work at a higher clock frequency, all parts of the circuit must be improved to operate within the shorter clock period. In an asynchronous circuit, performance can be enhanced by modifying only the most active parts of the design using innovations in VLSI technology. Since asynchronous circuits communicate using signalling protocols rather than clocks the modified components must only obey the requirements of the communication protocol. In principal, greater throughput for synchronous circuits can be achieved only when all VLSI components are realized on a new technology.

- *Adoption to environmental variations*. Changing environmental conditions can significantly vary the logic delays in VLSI circuits. Synchronous circuits are simulated extensively under a wide variation of parameters such as supply voltage and operating temperature to ensure that the chosen clock period guarantees correct operation under all specified conditions. The adaptability of asynchronous VLSI circuits allows them to function correctly under large environmental variations by allowing them to operate more quickly or more slowly accordingly.

The advantages of asynchronous circuits have not yet been fully realised due to the following reasons:

- The design of asynchronous VLSI circuits for specific applications appears to be difficult due to the lack of suitable CAD tools.
- Special techniques for removing hazards in asynchronous circuits often lead to a significant increase in silicon area [Lav93, Brzo95a].

Nevertheless, a growing interest from industry in asynchronous circuits is stimulating research in design methodologies which can be used to develop efficient asynchronous VLSI designs which could compete successfully in a market presently dominated by synchronous circuits.

1.2 Asynchronous VLSI design methodologies

A large number of existing asynchronous design approaches can be classified using the following three main criteria:

- delay models;
- data representation;
- signalling protocols.

1.2.1 Delay models in VLSI circuits

Delay models can be divided into three categories: fixed, bounded and unbounded delay models. In the fixed delay model, the delay is assumed to have a fixed value. According to the bounded delay model the delay may have any value in a given interval. In the unbounded delay model, the delay can have any finite value. Delays in digital circuits are associated with wires and gates. In principle, a circuit model is defined by its function and delay models for its wires and components [Birt95, Brzo95a].

Depending on the delay model assumption asynchronous circuits can be classified into three major groups: delay-insensitive, speed-independent and bounded-delay circuits [Birt95]:

- In *delay-insensitive* circuits all delays in gates and wires are allowed to be arbitrary but finite.
- Gate delays in *speed-independent* circuits are arbitrary and finite but signal transmissions along wires are instantaneous. This assumption allows the use of the isochronic fork [Berk91] where transitions on the forked parts arrive at their destinations at the same time.
- A *bounded-delay* circuit uses the bounded delay model to ensure correct data processing. In this model the delays through the data paths of the circuit are known and bounded whereas the control logic remains delay-insensitive.

1.2.2 Data representation

Data in asynchronous circuits can be represented using either dual-rail or single-rail data encoding.

Both delay-insensitive and speed-independent implementations require dual-rail encoding of data where each data bit is represented by two wires: a “zero” propagation wire and a "one" propagation wire. A standard level-sensitive dual-rail data encoding technique has four states:

- 00 - "initial state; data is not valid";
- 10 - "transmission of a logical zero";
- 01- "transmission of a logical one";
- 11 - "illegal state".

Once the data has been transmitted the wires must be returned to their initial state. Thus, the presence of new data is indicated by a transition on one of the propagation wires. The illegal state is not used in dual-rail data encoding.

The major disadvantage of using the dual-rail data representation compared to single-rail data encoding, where each wire represents one bit of binary information, is that its implementation requires twice as many wires and, as a consequence, leads to larger circuits. Bounded-delay asynchronous circuits allow the use of single-rail data encoding where combinational logic circuits similar to those used in synchronous designs can be used directly. This offers a significant reduction in silicon area.

1.2.3 Signalling protocols

Most asynchronous communications are based on using signalling protocols which define a "handshake" procedure between two computation blocks [Berk93]. A typical handshake protocol for a bundled-data system between a sender and a receiver is shown

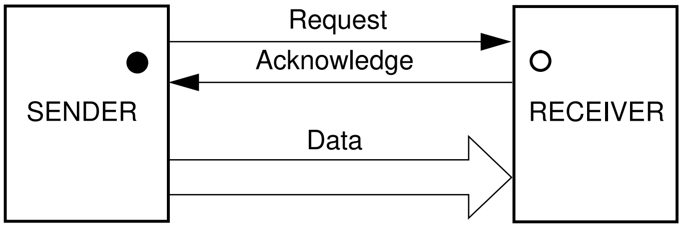


Figure 1.1: *Standard handshake signalling protocol*

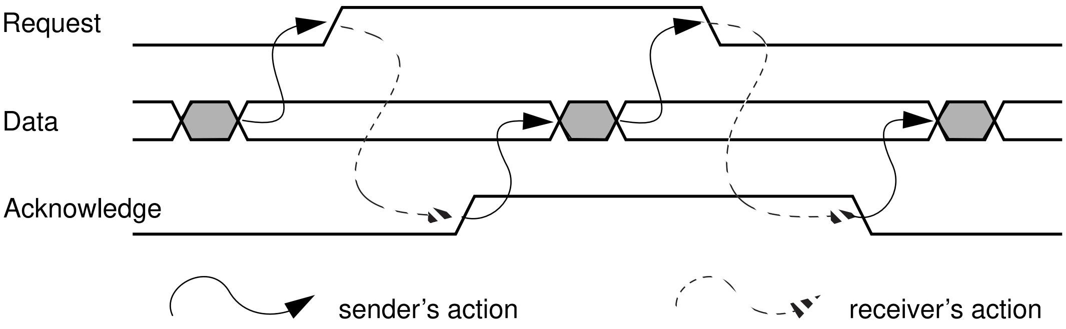


Figure 1.2: *Bundled-data protocol using two-phase transition signalling*

in Figure 1.1. According to this signalling scheme two control signals are required: "request" and "acknowledge". The request signal is used to initiate an action and the acknowledge signal indicates the completion of the requested action. These control signals carry all the necessary timing information to provide for proper data communication. In the handshake protocol, there are always the initiator of the action which generates the request and the passive circuit which waits for a request and then generates the acknowledge. The full and empty circles in Figure 1.1 denote the active and the passive partners in the handshake procedure respectively.

Two transition signalling schemes can be used to implement an asynchronous signalling protocol: two-phase (non-return to zero) and four-phase (return to zero) signalling [Lav93, Birt95]. Figure 1.2 illustrates the two-phase bundled data signalling protocol. There are two active phases in the communication process: the signal transitions (rising or falling) on the request and acknowledge wires. An event on the request or the

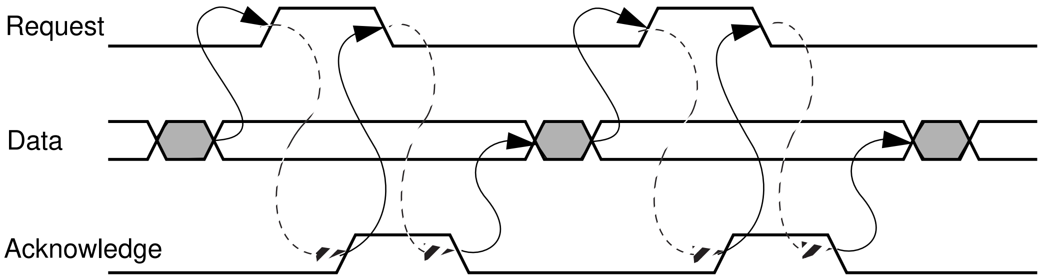


Figure 1.3: *Bundled-data protocol using four-phase transition signalling*

acknowledge control line terminates the active phase of the sender or the receiver respectively. During the receiver's active phase the sender must hold its data unchanged. Once the receiver generates an acknowledge event new data can be produced by the sender. In Figure 1.2 solid and dashed lines represent the sender's and the receiver's actions respectively.

A four-phase transition signalling protocol is shown in Figure 1.3. In this protocol the actions of the receiver and the sender are terminated when both the request and the acknowledge signals are returned to zero.

Note that the dual-rail data representation requires ($2n$+1) wires ($2n$ for the n-bits of data and 1 for the acknowledge) to send an n-bit word of data from the sender to the receiver. A separate request signal is not needed since the presence of new data can be identified by transitions on the n pairs of data wires.

Each signalling protocol has advantages and disadvantages. For example, four-phase signalling requires twice as many signal transitions as two-phase signalling, dissipating more energy. Four-phase control circuits are usually smaller than two-phase circuits but they are more difficult to design [Furb96].

1.2.4 Asynchronous design styles

Delay-insensitive circuits

Molnar, et. al. introduced techniques to implement delay-insensitive circuits which can be either clock-free or locally clocked (Q-modules) [Mol85]. High-level description languages such as *Occam* [Brun89] and a *trace-based* language [Eber87] were used by Brunvand and Ebergen respectively to design module-based delay-insensitive circuits. Specially developed automatic compilation procedures are applied to the high-level design description in order to implement a delay-insensitive circuit.

Quasi delay-insensitive and speed-independent circuits

Martin proposed a methodology for designing so called *quasi delay-insensitive* circuits which differ from speed-independent circuits mainly in the assumption that all forks in speed-independent circuits are isochronic, whereas quasi delay-insensitive circuits allow forks to be either isochronic or delay-insensitive. The design process includes two main steps:

- the developing of the high-level specification using the *communicating sequential processes* (CSP) [Mart90] language;
- the translation of the high-level specification into a circuit implementation.

Philips research laboratories developed the *Tangram* programming language which is similar to CSP. A set of tools providing for the compilation of the Tangram program in a *handshake* circuit has been implemented [Berk88, Berk91].

Several reports proposed a number of design techniques for speed-independent circuits [Chu87, Meng89]. These design approaches are based on the high-level circuit specification in the form of signal transition graph (STG).

Bounded-delay circuits

Bounded-delay circuits use the *fundamental-mode* assumption that the environment must wait for long enough for the output data to stabilize on the circuit outputs. The principles of fundamental-mode design techniques were developed first by Huffman [Huff54] and later extended by Unger [Unger69].

A design approach to building *burst-mode* finite state machines was proposed by Nowick et al. [Nowick91]. According to this approach:

- each state transition can occur under a certain set of input changes (so called an *input burst*) so that no burst from a particular state can be a subset of another burst from the same state;

- any state must be entered with the same set of input values.

The proposed timing mechanism allows the burst-mode finite state machine to be moved to a new state whenever the output associated with the previous state has changed enabling the input signals to be changed.

Ivan Sutherland in his 1988 Turing Award lecture described an elegant approach to building asynchronous pipelines called micropipelines [Suth89]. Micropipelines are asynchronous, event-driven pipelines based on the bundled-data interface. In micropipelines, the data is treated as a bundle, i.e. when the data produced by the sender is stable the sender issues a request event to the receiver; the receiver acknowledges the receipt of the data by sending an acknowledge event (see Figures 1.2 and 1.3). This handshaking mechanism is repeated when further data is produced by the sender.

1.3 Motivation for the chosen design methodologies

As has been shown above there is a great variety of different asynchronous design techniques. In this thesis the micropipeline and handshake circuit design methodologies are considered for the following reasons:

- An asynchronous version of the ARM6 microprocessor (AMULET1) has been designed by the AMULET research group at the Department of Computer Science in the University of Manchester and fabricated by GEC Plessey Semiconductors Limited. AMULET1 was designed using the micropipeline design approach with two-phase signalling which offers a good engineering framework for the design of complex asynchronous VLSI circuits [Furb94, Pav94]. A second generation of asynchronous ARM microprocessor called AMULET2 has recently been designed by the AMULET group. The design of the AMULET2 microprocessor is based on the four-phase micropipeline framework [Furb96].

- A great deal of engineering work has been carried out in the AMULET group to optimise the design flow of asynchronous circuits initially described in the Tangram language [Farn95, EdTR95]. In collaboration with Philips Research Laboratories an effective design environment has been developed and incorporated within the *Cadence* design framework using *SIMIC* design verification tools [Farn95, Sim94]. As a result, during the design process which goes from the Tangram specification to a circuit implementation an engineer may replace Tangram synthesised subcircuits with more efficient hand-developed design solutions.

Therefore, the micropipeline and handshake circuit design styles have been chosen as the basis of the work described in the rest of this thesis because of substantial design experience and tools support existing within the AMULET group.

1.4 Micropipelines

As was mentioned above the micropipeline design approach is based on three fundamental principles:

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

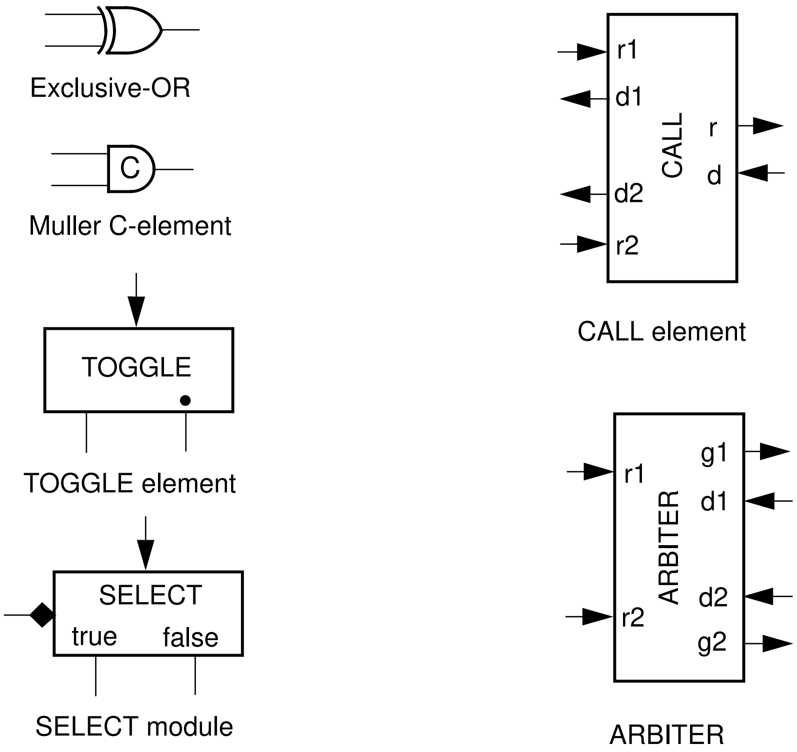
The micropipeline approach was described by Sutherland using event controlled elements which use two-phase transitions [Suth89].

1.4.1 Event controlled logic elements

Figure 1.4 shows an assembly of the most frequently used asynchronous logic modules for two-phase transition events.

The **Exclusive-OR** (**XOR**) element can be used to merge two event streams: if an event is received on either of the inputs of the XOR gate a response event will be produced on its output.

The **Muller C-element** is a memory element which performs a logical AND of input events. The C-element is sometimes called a “rendezvous” circuit for events since it



Exclusive-OR

Muller C-element

TOGGLE element

SELECT module

CALL element

ARBITER

Figure 1.4: *An assembly of basic logic modules for events*

produces a rising or falling event only when rising or falling events have arrived at both of its inputs respectively. Note that events can be generated on the inputs of the C-element at different times. Thus, if the inputs of the C-element are different its output remains unchanged and the C-element holds the previous state.

The **Toggle** element sends a transition alternately to one or other of its outputs when an event appears on its input. The first event is generated on the output marked with a dot.

The **Select** module is a demultiplexer of events. It steers a transition to one of two outputs depending on the logical value on its control input (marked with a ‘diamond’ symbol in Figure 1.4).

The **Call** element serves a function which is similar to a subroutine call in programming. It remembers which one of its two inputs received an event (*r1* or *r2*) and calls the procedure *r*. Once the procedure is finished an event is generated on input *d* and the Call element produces a matching *done* event on the corresponding output (*d1* or *d2*).

The **Arbiter** guarantees that both of its outputs are not active at the same time. The arbitration function is in granting service (*g1* or *g2*) to only one request (*r1* or *r2*) at a time. The other grant is delayed until after an event has taken place on the *done* wire (*d1* or *d2*) corresponding to the earlier grant. The behaviour of arbiter circuits is discussed elsewhere [Chan73].

There are certain restrictions on the behaviour of the environment which generates events to these elements. For instance, if two request signals arrive at the inputs of the XOR gate at nearly the same time the XOR gate produces a spike on its output. This spike can cause the asynchronous circuit to halt or to go into a metastable state. Therefore the environment must not allow this to happen. Similarly, the request events to the Call element must be mutually exclusive, i.e., the next request must be generated for the Call element only when the previous request has been processed. More about the restrictions on the environment behaviour can be found elsewhere [Suth89, Birt95, Brzo95a].

1.4.2 Micropipeline structures

Figure 1.5 illustrates a three-stage micropipeline with processing. 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 reg-

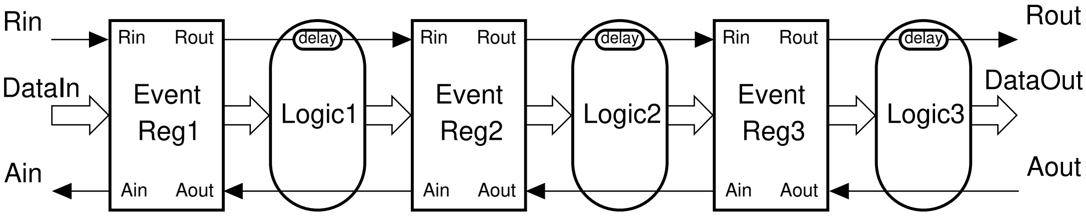


Figure 1.5: *A micropipeline with processing*

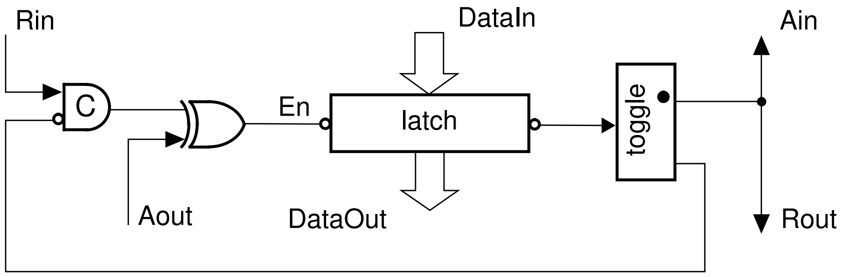


Figure 1.6: *AMULET1 event latch structure*

ister (*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.

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 1.6 illustrates the design of the event latch widely used in the AMULET1 microprocessor [Furb94, Day95, Birt95]. The design of the latch follows the two-phase transition signalling protocol shown in Figure 1.2. 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 (see Figure 1.6).

Initially, the latch is transparent and all the control wires are reset. When a rising event is sent to input *Rin* the output of the C-element goes high and the latch is closed, latching the input data. The toggle element steers the rising event to its ‘dotted’ output. A rising event on input *Aout* makes the latch transparent resetting its *enable* input (*En*). A falling event on the input of the toggle element causes a rising event to be generated on its ‘blank’ output priming the C-element. The operation of the latch is identical when a falling request signal is subsequently sent to its input *Rin*. Different latch structures and their control in two-phase and four-phase micropipelines are discussed elsewhere [Pav94, Day95, Furb96].

1.5 Handshake circuits

As was shown above many research groups are trying to develop effective CAD tools which translate the behavioural specification of an asynchronous circuit into silicon. One such example is the Tangram language developed by Philips Research Laboratories [Berk88, Berk91, Scha93].

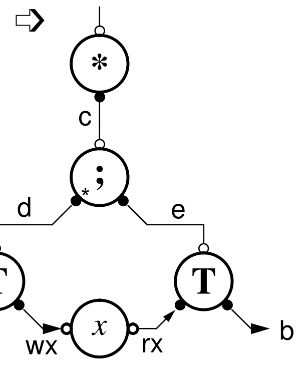
Tangram describes the VLSI circuit as a set of processes which communicate along channels. The Tangram program is translated into an intermediate format called a handshake circuit. Handshake circuits are composed of handshake components and channels on which they communicate [Berk93]. Components communicate with each other along channels using a four-phase signalling protocol. The result of compiling the Tangram program is a silicon layout with particular performance, power consumption and silicon area properties. The transparency of the compilation process makes it possible to go back to the Tangram program level where it is easy to make improvements to the properties of the target VLSI design.

Consider the Tangram program for a single buffer stage shown in Figure 1.7a. This program consists of three parts: an *auxiliaries* part, which is optional, an *externals* part and a *command* part [Scha93].

The *auxiliaries* part includes preliminary definitions of constants, types, etc. This part is separated from the *externals* part by the symbol ‘|’. In our program the *auxiliaries* part declares type *int* of 8 bits.

```
int = type [0..255]
|
(a?int & b!int) .
begin
        x: var int
|
        forever do
                a?x ; b!x
        od
end
```

a)



b)

__Figure 1.7:__ Single buffer stage: a) Tangram program; b) handshake implementation

The *externals* part defines a list of external channels. The program communicates along these channels with the environment. An external channel is described using a name and a direction (the channel description may contain a type definition). The description of an input or output channel consists of the name and the symbol '?' or '!', which are followed by an optional type, respectively. According to the Tangram program (see Figure 1.7a) the buffer communicates with the environment through the input channel *a* and the output channel *b* accepting and sending data of type *int*.

The *command* part of a Tangram program describes the behaviour of the program. In the *command* part of the Tangram program shown in Figure 1.7a a variable *x* of type *int* is defined after keyword ***begin*** which opens the scope (keyword ***end*** closes the scope). Note that new names with their types, which are used within the scope, must be defined only before the symbol '|'. The behaviour of the program must be described after the symbol '|'. In the Tangram program of the buffer an input value is received along channel *a* and stored in variable *x* (*a?x*). During the next step (the sequence of steps is separated by the symbol ';') the value of variable *x* is transmitted via output channel *b* (*b!x*). This procedure is repeated forever which is defined by the command ***forever do ... od***.

The Tangram program shown in Figure 1.7a is translated into the handshake circuit illustrated in Figure 1.7b. The circuit contains several channels which connect the active port ($^\bullet$) of one handshake component to the passive port ($^\circ$) of another. The communication process is started by a handshake on the active end of the channel and completed when an acknowledge is received from the passive end. The transmission of a new signal along a channel can start only after the receiver has confirmed the receipt of the previous transmission.

The environment activates the buffer along the activation channel (marked with ⇨) of the component called *repeater*. The repeater performs the repetition operation (command ***forever do***) along channel *c*. A sequencer process is triggered by a request signal on the passive port of the *sequencer* component which performs the operator “;”. As a result, the sequencer performs a handshake on channel *d* first (it is marked with *) and then a handshake along channel *e*. The output channels of the sequencer activate the corresponding *transferer* components marked with *T*. The left transferer is triggered first along channel *d*, fetching the data from input *a* and passing it to storage element *x*. Once the left transferer completes the handshake along channel *d* the right transferer is activated by a request signal generated on channel *e*. As a result, the data is fetched from storage element *x* and transmitted to output *b*.

Figure 1.8 illustrates implementations of the repeater (see Figure 1.8a), the sequencer (see Figure 1.8b), the transferer (see Figure 1.8c) and the storage element (see Figure 1.8d) [Birt95, EdTR95, FarnTR96].

The inputs and outputs of the repeater are low in the initial state (see Figure 1.8a). When $a_r\uparrow$ a rising event is generated on the request output b_r, i.e., $b_r\uparrow$. A rising acknowledge event on input b_a of the repeater sets its output b_r to zero. When $b_a\downarrow$ a new rising request is produced on output b_r of the repeater. Since the request on channel *a* is never acknowledged output a_a is grounded.

The order of events on the wires of the sequencer can be described as follows:

$$(a_r\uparrow \quad b_r\uparrow\, b_a\uparrow\, x\downarrow\, b_r\downarrow\, b_a\downarrow \quad c_r\uparrow\, c_a\uparrow\, a_a\uparrow\, a_r\downarrow\, x\uparrow\, c_r\downarrow\, c_a\downarrow \quad a_a\downarrow)^*. \tag{1.1}$$

Symbol “*” denotes that the sequence of events in parenthesis is repeated an infinite number of times.

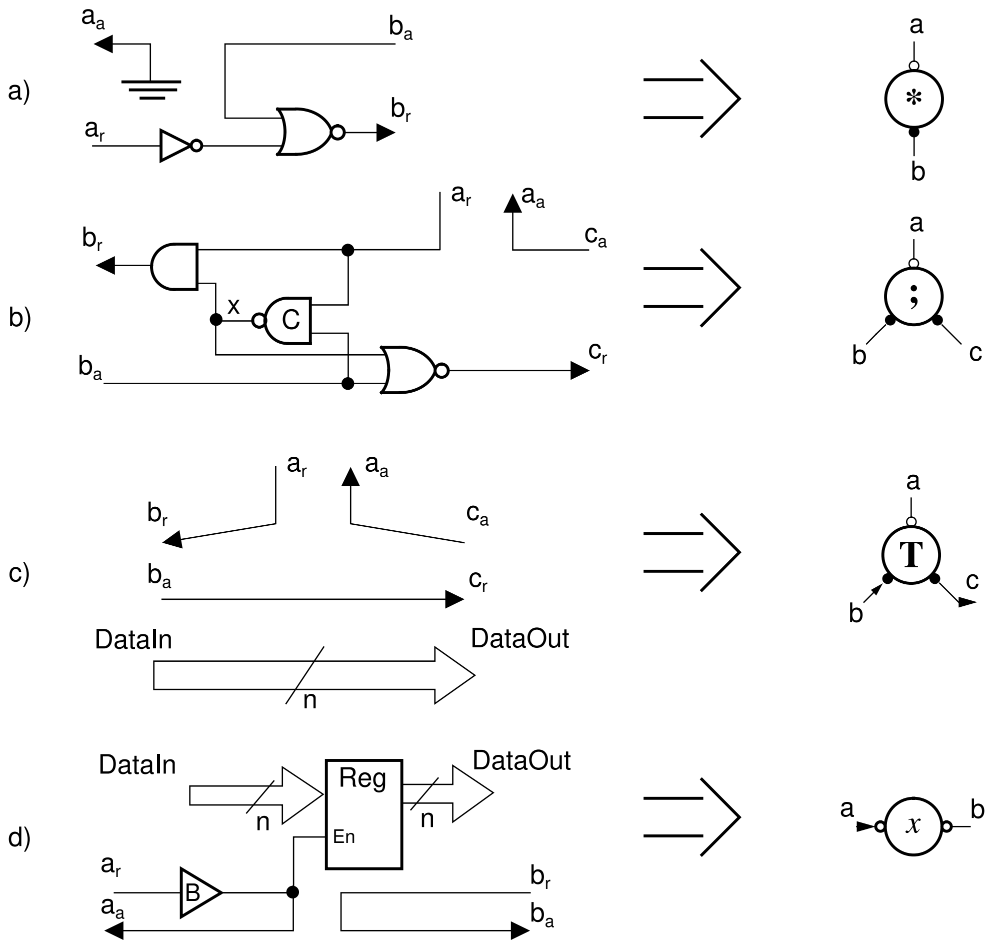


Figure 1.8: *Handshake components a) repeater; b) sequencer; c) transferer; d) storage element*

The transferer is activated along channel *a* producing a request for data on channel *b*. When the data is ready to be transmitted channel *c* is activated. The completion of the data transmission is indicated by an acknowledge signal along channel *a*.

The storage element shown in Figure 1.8d is implemented using an *n*-bit register built from level-sensitive latches. When the data is stable on the *DataIn* inputs the a_r signal goes high making the register latches transparent. As a result, the data is stored in the register. Note that the a_r signal is buffered to ensure the required drive strength for the enable signal. After that a rising event is produced on output a_a, i.e., $a_a\uparrow$. Once $a_r\downarrow$ the latches of the register are closed and then $a_a\downarrow$. If the data needs to be read a rising request signal is produced on input b_r causing a rising event on output b_a. Once the data is transmitted input b_r is returned to zero setting output b_a to low. Designs of other handshake components can be found elsewhere [Scha93, Birt95, FarnTR96].

1.6 Design for testability of asynchronous circuits

Although asynchronous VLSI circuits demonstrate potential advantages which allow them to be used in designs with low power consumption, high component modularity and average case performance, their commercial benefits can only be fully exploited if asynchronous chips can be tested in volume production. The test procedure must guarantee a high fault coverage for a given class of faults in the circuit under test within a given time interval.

The testing of asynchronous circuits for fabrication faults is more complex than that of synchronous circuits. The major factors that complicate the testing of asynchronous circuits are [Hulg94]:

- The presence of a large number of state holding elements. This makes the generation of tests hard or even impossible.
- The difficulty of detecting hazards and races.
- The absence of a global synchronization clock. This decreases the level of controllability of the states of the asynchronous circuit.
- Logic redundancy, which is introduced into asynchronous circuits to ensure their hazard free behaviour, sacrifices testability. As a result, some stuck-at faults in redundant parts of the asynchronous circuit cannot be detected by logic testing.

As a consequence, the developing of asynchronous DFT techniques which can facilitate the testing of asynchronous VLSI circuits is an important problem which must be solved before their potential can be realized commercially.

1.7 Thesis overview

Current results obtained so far in the field of testing asynchronous circuits are discussed in chapter 2. This includes an analysis of reports devoted to fault modelling in asynchronous VLSI circuits, test generation techniques and design for testability methods for delay-insensitive, speed-independent and bounded-delay circuits.

A relationship between the power consumption and the testability of CMOS VLSI circuits is investigated in chapter 3. The method used to evaluate this relationship is based on elements of information theory. It is shown that design for low power consumption and design for testability are in direct conflict. The resolution of this conflict lies in separating the testing issues from the low power issues by giving the circuit distinct operating and test modes. These results have been submitted to IEEE Transactions on CAD for publication [Pet95d].

Testable designs of static CMOS C-elements which provide for the detection of single line stuck-at and stuck-open faults are given in chapter 4. It is shown that driving the feedback transistors in the proposed testable static C-elements transforms their sequential functions into combinational ones depending on the driving logic value. This simplifies the testing of asynchronous circuits which incorporate a large number of state holding elements. The results presented in chapter 4 have been published as a technical report from the Department of Computer Science, University of Manchester [PeTR95].

A method for designing micropipelines for testability is presented in chapter 5. The test strategy is based on the scan test technique. The proposed test approach provides for the detection of all single stuck-at and delay faults in micropipelines and sequential circuits based on the micropipeline approach. Materials presented in chapter 5 have been published in the proceedings of the 5th Great Lakes Symposium on VLSI, USA, [Pet95b] and the 13th IEEE VLSI Test Symposium, USA, [Pet95c].

Chapter 6 presents two structural approaches to designing asynchronous sequential circuits for random pattern testability. The testable designs of two-phase and four-phase sequential circuits are built using either the micropipeline design style or handshake components. The proposed test procedures for such sequential circuits provide for the separate testing of the combinational logic block and the memory elements. A case study of the AMULET2e memory controller designed for random pattern testability is presented to demonstrate the practicality of the proposed design approaches. Finally, a BIST implementation of a micropipeline is considered. Some of the results described in chapter 6 have been published in IEE Proceedings “Computer and Digital Techniques” [Pet95a].

In chapter 7 different implementations of an asynchronous adder are investigated. It is shown that the choice of single-rail, dual-rail or combined single and dual-rail (hybrid) data encoding techniques in the adder design brings different trade-offs between the testability, performance and area overhead. A case study of an asynchronous comparator demonstrates that a hybrid implementation brings a reasonable compromise between the area overhead, performance degradation and testing costs. These results have been published in the IEE Colloquium on the Design and Test of Asynchronous Systems [Pet96e].

The design for testability of an asynchronous block sorter is presented in chapter 8. Testable structures of the block sorter are implemented using the scan test and BIST design methodologies.

Finally, chapter 9 gives the principle conclusions from the work described in this thesis and suggests some directions for future research.

Chapter 2 : Testing Asynchronous Circuits - Related Work

This chapter provides an overview of fault models used to develop tests for fabrication faults in VLSI circuits and some results reported so far in the field of testing and design for testability of asynchronous VLSI circuits.

2.1 Fault models

Errors in VLSI circuits can be caused by physical faults such as physico-chemical disorders of the technological process (threshold changes, short circuits, open circuits, etc.) or changes in the environment conditions in which the VLSI circuits operate [Abrah86, Russ89]. After fabrication a VLSI circuit must be tested to ensure that it is fault-free. The testing of VLSI circuits for fabrication faults is implemented by applying a set of test vectors to their primary inputs and observing test results on their primary outputs. If the outputs of the circuit under test are different from the specification, the circuit is faulty.

In order to derive tests for the circuit, the fault model and the circuit descriptive model must be chosen. Obviously, the lower the level of circuit representation used in test pattern generation, the more accurate the fault model will be. However, for modern VLSI circuits having millions of transistors on a chip the transistor level description model increases the test generation time drastically. As a result, the right choice of the fault model and the level of the circuit representation can bring a reasonable compromise between the test derivation time and the fault coverage.

2.1.1 Gate-level fault models

The stuck-at fault model. The most widely accepted fault model used to represent different fabrication failures in VLSI designs is the stuck-at fault model [MClus86, Russ89, Weste93]. A stuck-at fault on line *a* connects it to the power supply voltage (V_{dd}) or ground (V_{ss}) permanently. Originally the stuck-at fault model was designed to describe the fault behaviour of the circuit under test at its gate level representation. Figure 2.1 shows a three-input NAND gate. A stuck-at-0 fault on input *A* of the gate (1-SA0) produces a logic one on its output regardless of the values on the other inputs. This fault is equivalent to 4-SA1 fault on output *Y* of the gate. Both faults are detected by applying an 'all ones' test to the inputs. As a result, the fault-free response, which is zero, differs from the fault response, which is one. Table 2.1 contains test vectors for the detection of all stuck-at faults in the three-input NAND gate.

Table 2.1: *Test set for stuck-at faults in the three input NAND gate*

Inputs			Output		Detected Faults
A	B	C	Y(Fault-free)	Y (Faulty)	
1	1	1	0	1	1,2,3-SA0; 4-SA1
0	1	1	1	0	1-SA1, 4-SA0
1	0	1	1	0	2-SA1, 4-SA0
1	1	0	1	0	3-SA1, 4-SA0

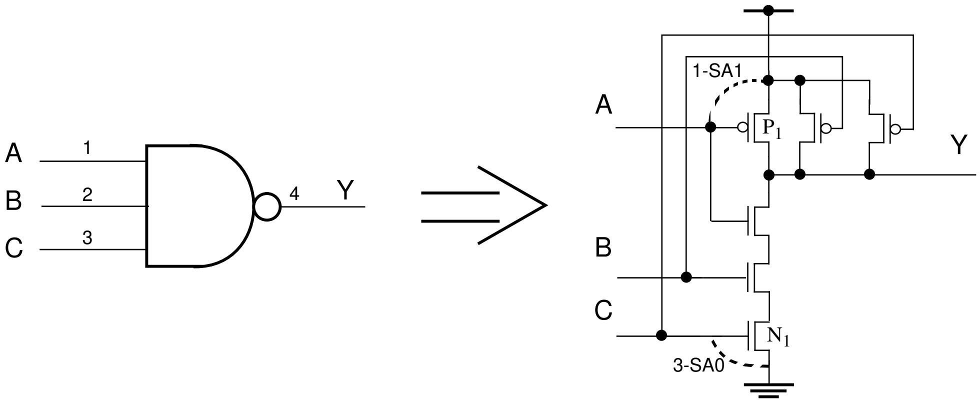


Figure 2.1: *Three-input NAND gate*

For any circuit the total number of all possible faulty circuits with multiple stuck-at faults can be estimated as $3^n - 1$, where n is the total number of signal nets. In practice, only the single stuck-at fault model is considered to avoid an incredibly large number of faulty circuits and to enable fault simulations to be performed in reasonable time.

The bridging fault model. Bridging faults are caused by shorts between signal lines in the circuit. For instance, a short between lines 1 and 2 of the NAND gate shown in Figure 2.1 can be modelled in two ways: lines 1 and 2 are connected together using net 1 or net 2 as an input. These faults can be detected by test (1,0,1) or (0,1,1) respectively. Note that stuck-at faults can be modelled by shorts. For example, 1-SA1 fault is equivalent to a short between the source and the gate of transistor P_1, whereas 0-SA1 fault is equivalent to a short between the source and the gate of transistor N_1 (see Figure 2.1).

The delay fault model. A delay or transition fault alters the signal propagation delay along the faulty line [Lala85, Agra92]. As a result, signals can arrive at the outputs of the circuit before or after the time expected. Testing delay faults in asynchronous circuits is hard due to the absence of a synchronization clock.

2.1.2 Transistor-level fault models

Previous work concerning the accuracy of gate-level fault models has been reported. Experimental results with test chips indicated that 20.8% of all faulty blocks had no gate-level stuck-at faults [Panc92]. Other results have shown that 36% of all faults are of the non-stuck-at variety [ShenM85]. According to these results the application of tests which provide for the detection of all single gate-level stuck-at faults in the chosen chips still “pass” faulty circuits. Fault models described at the transistor level are more accurate and, hence, offer a better coverage of fabrication faults in the circuit under test. Line stuck-at, stuck-open and bridging fault models are used to describe the effects of the majority of fabrication faults in CMOS designs [Chen84, Abrah86, Russ89, Abra90].

The stuck-at fault model. As was mentioned above the stuck-at fault model assumes that a fabrication failure causes the wire to be stuck permanently at a certain logical value.

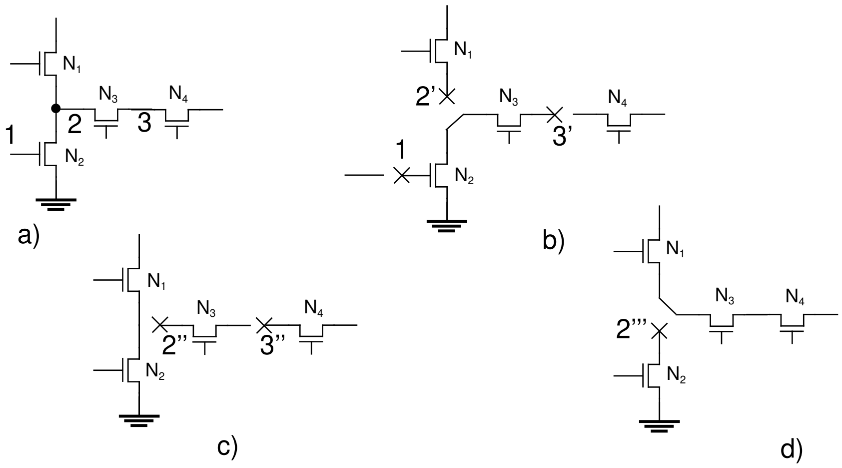


Figure 2.2: *Locations of line stuck-at faults and their interpretation in a fragment of CMOS design*

Consider a fragment of a CMOS design (in Figure 2.2a) with possible locations of line stuck-at faults. For instance, the stuck-at one fault in node 1 (1-SA1) is interpreted as a break on line 1 with the gate of *n*-type transistor N_2 connected permanently to the power supply voltage (in Figure 2.2b). The application of a constant voltage is marked with a cross.

Fault 2-SA can be represented in three ways:

- the disconnection of transistor N_1 from node 2 and setting its source to a logical value (fault 2'-SA in Figure 2.2b);
- the disconnection of transistor N_3 from node 2 and setting its source to a logical value (fault 2"-SA in Figure 2.2c);
- the disconnection of transistor N_2 from node 2 and setting its drain to a logical value (fault 2'''-SA in Figure 2.2d).

Note that fault 2'''-SA is equivalent to fault 1-SA0 when transistor N_2 is permanently off. Thus, fault 2'''-SA can be excluded for the sake of simplicity. Notations 3'-SA or 3"-SA denote a break on the left side of line 3 and setting a permanent logical value on its right side or a break on the right side of line 3 and setting a permanent logical value on its left side respectively (compare Figures 2.2b and 2.2c).

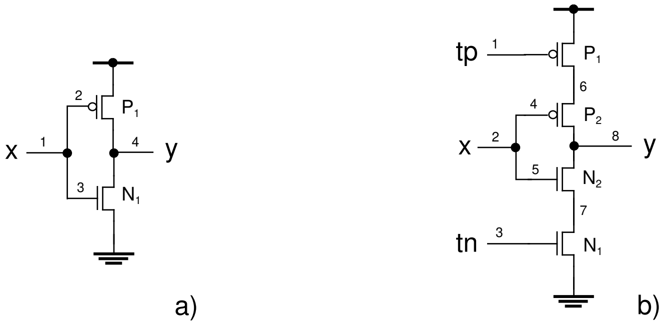


Figure 2.3: *CMOS inverters: a) inverter with two logically untestable stuck-at faults; b) testable inverter*

The basic CMOS inverter shown in Figure 2.3a consists of one of each of the two types of transistor: a p-type and an n-type transistor [Weste93]. When input x is low the n transistor is off and the p transistor is on. Output y is connected to the power supply voltage (V_{dd}) which corresponds to a logical one. If input x is high the n transistor is on and the p transistor is off. Output y is connected to ground (V_{ss}) which is a logical zero. Figure 2.3a shows line stuck-at fault locations in the CMOS inverter. For example, fault 1-SA1 of the inverter sets its output y to a constant logical zero. Input x of the inverter must be set to low to detect this fault, whereupon output y remains low whereas the fault-free response is high.

Consider fault 2-SA0 in the inverter illustrated in Figure 2.3a. This fault sets transistor P_1 permanently on. If input x is high both transistors P_1 and N_1 are on. This leads to an uncertain situation when a logical one or zero can be registered by the test circuitry depending on the strengths of the transistors. As a consequence, the detection of fault 2-SA0 cannot be guaranteed by logic testing. Similar observations can be made for fault 3-SA1.

The stuck-open fault. The stuck-open fault model represents a fault effect caused by a fabrication failure which permanently disconnects the transistor pin from the circuit node. Stuck-open faults can be opens on the gates, sources or drains of transistors. In the presence of a single stuck-open fault (SO) there is no path from the output of the circuit to either V_{dd} or V_{ss} through the faulty transistor. For example, in the presence of fault P_1-SO (Figure 2.3a) output y cannot be set high since there is no connection between

Table 2.2: *Tests for stuck-at and stuck-open faults of the inverter in Figure 2.3b*

Single SA0 faults	Single SA1 faults	Single SO faults	Test sequences			Fault-free output	Faulty output
			x	tn	tp	y	y
2	8		1	1	0	0	1
8	2		0	1	0	1	0
	1,4	P_1,P_2	1	1	0	0	0
			0	1	0	1	0’
3,5		N_1,N_2	0	1	0	1	1
			1	1	0	0	1’
4			1	1	1	0	0
			1	0	0	0’	1
	5		0	0	0	1	1
			0	1	1	1’	0
1	6		1	1	1	0	0
			0	1	1	0’	1
7	3		0	0	0	1	1
			1	0	0	1’	0
6			1	1	0	0	0
			0	1	0	1	0_w
	7		0	1	0	1	1
			1	1	0	0	1_w

V_{dd} and node *y*. This fault can be identified by a set of two test patterns <T_1=1,T_2=0> applied sequentially to input *x*. As a result, the output of the faulty inverter remains low whereas the output of the fault-free inverter is high. Fault N_1-SO is detectable by a test set <T_1=0,T_2=1>.

Figure 2.3b shows a CMOS inverter which is testable for all single stuck-at and stuck-open faults. Two additional transistors (P_1 and N_1) controlled by two separate inputs are inserted into the inverter shown in Figure 2.3a. Table 2.2 contains tests for line stuck-at faults and transistor stuck-open faults in the inverter. Note that faults 4-SA0 and 5-SA1 are detectable by logic testing. For instance, a test sequence <T_1=111,T_2=100> applied to the inputs of the inverter detects fault 4-SA0. The effect of fault 6-SA0 or 7-SA1 is a ‘weak zero’ (0_w) or a ‘weak one’ (1_w) output signal respectively. These voltage levels are very close to the corresponding logical 1 and 0 voltage levels since output *y* was previously set to the same logical values. Faults 1-SA1 or 3-SA0 result in a ‘floating zero’

(0') or 'floating one' (1') output signal respectively. The output capacitance of the inverter can be considered as a dynamic memory element which keeps its precharged value for a certain time. It is assumed that the time between the application of two test vectors is small enough not to allow a floating output voltage level to reach the CMOS threshold level [Wad78, Red86, Weste93]. Hereafter we will treat weak and floating logical values as normal ones. A stuck-at fault on the gate of a CMOS transistor keeps the transistor on or off permanently depending of the type of fault. Thus, transistor stuck-open faults in CMOS designs can be represented by their correspondent gate stuck-at faults. For instance, fault 5-SA0 on the gate of transistor N_2 is equivalent to fault N_2-SO (see Figure 2.3b). As a result, testing for stuck-at faults of the inverter illustrated in Figure 2.3b guarantees the detection of all its stuck-open faults.

The bridging fault model. It has been shown that in CMOS technology a bridging fault at the transistor level can convert a combinational circuit to a sequential one [Lala85]. This creates extra problems for detecting such faults. Some bridging faults at the transistor level representation of the circuit under test have no logic realizations at the gate level. Testing for such faults requires the circuit structure to be tested which is not easy [Russ89].

2.2 Testing delay-insensitive and speed-independent circuits

It has been observed that stuck-at faults in delay-insensitive circuits, where each transition is confirmed by another, cause the whole circuit to halt; this is called the self-diagnostic property of delay insensitive circuits [Dav90]. A stuck-at fault on a line is equivalent to an infinite signal propagation delay along this line. As a result, a transition that is supposed to occur does not happen because of a stuck-at fault; this is called an inhibited transition [Haz92]. A fault that causes an inhibited transition eventually makes the delay-insensitive circuit deadlock which is easy to detect. A request is issued to the circuit under test whereafter an acknowledge signal is assumed to arrive within a bounded time. If the acknowledge signal does not arrive within the specified time, the circuit is considered to be faulty.

For instance, according to the four-phase protocol (see Figure 1.3) the environment generates the following inputs:

$$Req\uparrow; \quad [Ack]; \quad Req\downarrow; \quad [\neg Ack]. \tag{2.1}$$

The circuit responds with:

$$[Req]; \quad Ack\uparrow; \quad [\neg Req]; \quad Ack\downarrow, \tag{2.2}$$

where a *handshake expansion* [*exp*] denotes the waiting for the Boolean expression (*exp*) to become true [Mart89].

As a result, in the presence of a stuck-at fault on any of the control lines (*Req* or *Ack*) either the environment or the faulty circuit will wait forever.

It has been observed that speed-independent circuits are self-checking in the presence of output stuck-at faults [Beerel92]. Some stuck-at input faults in speed-independent circuits can cause premature firing. Premature firing is a transition which happens too early according to the fault-free circuit specification. The detection of such faults requires a special testability analysis to be carried out [Haz92]. Figure 2.4 illustrates an implementation of a D-element which sequences two four-phase handshakes [Huz92, Hulg94]. The functioning of the D-element can be described by the following specification:

$$*[[l_i]; \; u\uparrow; \; [u]; \; l_o\uparrow \; [\neg l_i]; \; r_o\uparrow; \; [r_i]; \; u\downarrow; \; [\neg u]; \; r_o\downarrow; \; [\neg r_i]; \; l_o\downarrow] \tag{2.3}$$

The D-element shown in Figure 2.4 is a speed-independent circuit with two isochronic forks with inputs l_i and r_i respectively. It is easy to show that all output stuck-at faults in the D-element cause the circuit to halt. Consider an input stuck-at-0 fault on net r_2.

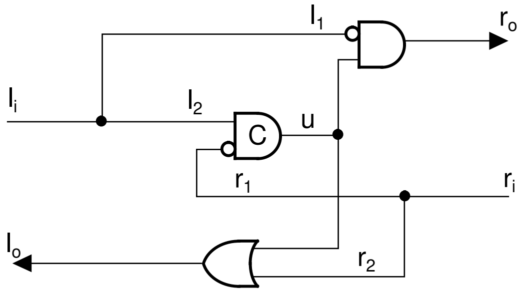


Figure 2.4: *D-element*

According to specification (2.3) output l_o goes low after r_i is reset. In the presence of fault r_2-SA0 output l_o is prematurely set to zero by a falling event on net u which must happen after resetting input r_i. Fault l_1-SA0 causes a premature firing on output r_o. Note that the D-element demonstrates the self-checking property for any other input stuck-at faults.

A successful proposal to adapt known test generation algorithms for testing delay-insensitive circuits using the stuck-at fault model has been reported [Haz92]. A technique to test delay-insensitive circuits was synthesized from a high-level specification. It used the high-level synthesis method for delay-insensitive circuits that was developed by Martin [Mart89]. It has been shown that it is possible to derive conditions for inhibited and premature transitions in the presence of a stuck-at fault [Haz92]. The goal of the proposed test generation approach is to find a sequence of input events which puts the faulty circuit in one of the states where it generates a premature transition or halts. Clearly, it is enough to find a test which causes the circuit to halt in order to detect a fault. It is more difficult to derive a test for faults which cause only premature firings.

Combinational logic in a synchronous design is a feedback-free network of logic elements which calculates a function of the primary inputs. There are similar feedback-free delay-insensitive circuits which make their computations without buffering the result, although they contain state-holding elements. It was shown that any delay-insensitive circuit in which:

1) there are no feedback lines at the gate level;

2) each production rule for an up-transition (down-transition) has only positive (negative) literals in its guard;

can be reduced to a standard combinational logic circuit to ease the testing procedure. It has been proved that any test which detects all testable faults in this combinational network can also detect all testable faults in the corresponding delay-insensitive circuit.

The standard D-algorithm (see Appendix A) can be extended to obtain test patterns for stuck-at faults in delay-insensitive combinational circuits. Regular forward and backward propagation techniques can be used for such circuits. The major difference with combinational circuits is that there are some state-holding elements in delay-insensitive combinational circuits. It is necessary to take into consideration whether the circuit is in an up-going or a down-going phase for propagating a fault through a state-holding element.

Forward propagation. Let S be a state-holding element. Transform S into S_u by replacing the guard for the down-transition with the negation of the guard for the up-transition. Gate S_u is a combinational gate and is equivalent to S during the up-phase. Thus, the propagation of D and $\overline{D}$ is the same for S as for S_u. For instance, if S is a C-element, then S_u is an AND gate. During a down-phase, S propagates a faulty signal if the output of the gate is 1 after the up-phase. Transform S into S_d by replacing the guard for the up-transition with the negation of the guard for the down-transition. Then S_d is a combinational gate which is equivalent to S during the down-phase, if the output of S is 1 after the up-phase. The propagation of D and $\overline{D}$ is the same for S as for S_d. If S is a C-element then S_d is an OR gate.

Backward propagation. Transform S into S_u for the up-phase and S_d for the down-phase. The backward propagation for these combinational circuits is the same as it was described before. If S is a C-element with output D then all its inputs must be 1 for detection during an up-phase, and at least one input is 1 for detection during a down-phase. If the output is $\overline{D}$ then during an up-phase at least one input is 0; during a down-phase all the inputs are 0.

The design for testability of delay-insensitive circuits has been discussed [Haz92]. It was shown that each fault in a delay-insensitive circuit can be made testable by means of *ad hoc* techniques (see Appendix B). The addition of test points, which can be either control or observation points, alleviates the testability problems greatly. If a premature firing is unstable then a control point is needed; if the premature firing is not propagated to a primary output then an observation point is needed. It was shown how to find a place where a test point must be inserted. For VLSI circuits which are pad-limited, it was proposed to merge the test points together into a queue. A fully testable design for such a test queue was derived.

A test strategy for handshake circuits has been described [Ron93]. This approach is based on a simple test procedure for which handshake components are enhanced with a special test operation. As a result, the test generation time for stuck-at faults is linear in the size of the VLSI circuit. Unfortunately, it does not address the testing issues of the circuit's data paths. A partial scan test technique for asynchronous circuits has been described by Marly Roncken [Ron94]. Taking the digital compact cassette (DCC) error corrector decoder as an example it was shown how an asynchronous partial scan test technique can be adapted for a high-level VLSI programming environment. The DCC error corrector decoder performs in test and normal operation modes. An asymmetric isochronic fork was used to switch the mode of operation of the circuit. It is assumed that there is a particular branch on which transitions are guaranteed to move more quickly along this fork. Unfortunately, the use of the asymmetric isochronic fork does not allow the change of mode to be implemented using delay-insensitive control logic. The reported scan solution was designed for testing the error corrector and a controller which use dual-rail data encoding.

A partial scan test methodology for detecting stuck-at faults in control parts of macromodule based asynchronous circuits has been described [Khoc95]. The proposed test strategy is more effective than the conventional full-scan approach since it requires fewer scan testable memory elements and, also, it demonstrates a high level of fault coverage. In test mode, the scanning of test vectors into the scan path is implemented using a clock whereas the circuit performs asynchronously in normal operation mode. A scan latch selection strategy has been introduced which is based on a structural and testability analysis of the circuit. Once the scan path has been selected the proposed test algorithm allows the detection of faults in the elements of the control part which are not included into the scan test path. The remaining network consists of XOR gates and C-elements. Modifications to the Select block and C-element were introduced in order to fit their

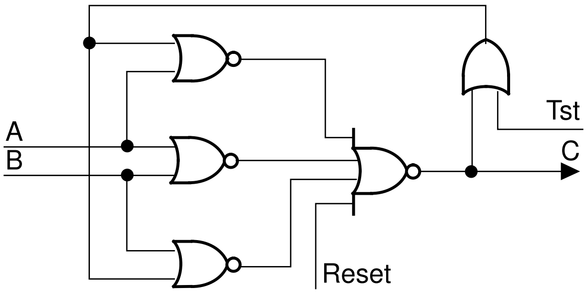


Figure 2.5: *Gate level implementation of the modified C-element*

testable implementations to the partial scan environment. Figure 2.5 illustrates a gate level implementation of the modified C-element which can perform as an AND gate or an OR gate. The C-element operates as an OR gate if an additional test input (*Tst*) is high. The C-element performs the AND function of its inputs by pulsing the global reset signal (*Reset*) high. As a result, a network of modified C-elements can be tested in the same way as a combinational circuit.

2.3 Testing bounded delay circuits

2.3.1 Testing asynchronous sequential circuits

The earliest asynchronous sequential circuits were designed using Huffman finite state machines [Unger69]. Figure 2.6a illustrates a Huffman finite state machine. This machine consists of combinational logic (*CL*), primary inputs (*PI*), primary outputs (*PO*) and feedback state variables. States are stored on feedback loops which may be constructed using delay elements. The combinational logic is fed by the primary inputs and the state inputs (*SI*) which are produced by the feed-back delay elements. After the application of each input vector to the *PI* inputs the state machine moves into a new state changing its state outputs (*SO*) and generating a new vector on its primary outputs. The Huffman model shown in Figure 2.6a can be used to design bounded delay asynchronous circuits. Since we need to make sure that the combinational logic has settled in response to a new input before the present-state entries change, the choice of proper delay elements is important.

An algorithm for generating tests to detect stuck-at faults in asynchronous sequential circuits based on the Huffman model has been reported [Putz71]. This algorithm is based upon an extension of the D-algorithm (see Appendix A). It was assumed that a stuck-at fault *F* modifies only the logical function of *S*. The basic test strategy proposed consists of the following steps:

1) transform the detection procedure of fault *F* in *S* into the detection of a corresponding set F^r of faults in an iterative combinational logic circuit CL^r derived from *S*;

2) extend the D-algorithm to derive a test *T* for F^r in CL^r;

3) simulate the test in *S* to verify whether or not *T* is a test for *F*.

After cutting feedback lines the original circuit *S* becomes acyclic as shown in Figure 2.6b. The acyclic circuit *CL* has primary inputs (*PI*) primary outputs (*PO*) and primary pseudo-inputs (*SI*) and pseudo-outputs (*SO*) which are introduced by the cutting points. If it is necessary to find a test for *F* in *S* of length *r*, that is a sequence of length *r* of primary input patterns which detects *F*, then *S* is modified into a sequence of *r* identical combinational networks CL_i, $1 \le i \le r$, with primary inputs PI_i, pseudo-inputs SI_i,

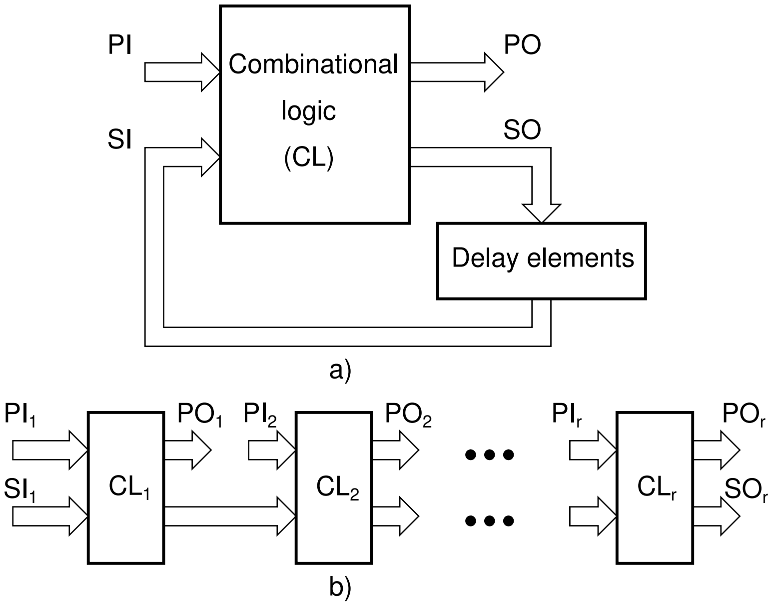


Figure 2.6: *Huffman finite state machine a) and its corresponding iterative combinational circuit b)*

pseudo-outputs SO_i and outputs PO_i. The pseudo-outputs of CL_i are identical to the pseudo-inputs of CL_{i+1} (see Figure 2.6b).

The modified D-algorithm is used to find a test for set F^r faults in CL^r with the following conditions:

1) the derived test cannot be dependent upon any of the pseudo-inputs *SI* of CL^r;

2) an effect of fault *F* must be visible at one of the primary outputs *PO*.

As a result, the test consists of an ordered set of *r* input patterns applied to the primary inputs *PI* of *S*. After that the behaviour of *S* is simulated while applying the derived set of patterns. If during the simulation no races or hazards are registered in *S*, then the test is accepted as a test for fault *F* in *S*.

A set of formal methods has been developed to design testable asynchronous sequential circuits [Suss84, Li88, Keut91]. For example, it was proposed to add one or at most two state variables, one extra input and to use one or more observable outputs in order to make the sequential circuit under test strongly connected and testable through scan-out features [Suss84, Li88]. An asynchronous sequential network is strongly connected if any stable state can be reached from any other state. The scan-out technique is applied directly to the flow table describing the asynchronous sequential circuit to be tested. The test procedure proposed is based on verifying the flow table of the circuit under test. As a result, no fault models are used. The use of this approach is limited by the complexity of the circuit to be tested and becomes impractical for asynchronous VLSI circuits. The test technique does not guarantee both hazard-free operation and hazard-free robust path-delay-fault testability of asynchronous circuits.

Some heuristic techniques and procedures to design asynchronous circuits which are simultaneously hazard-free, robust path-delay-fault testable and hazard-free in operation have been reported [Keut91]. The synthesis of asynchronous sequential circuits is performed using a high level specification, the signal transition graph. Using an appropriate delay model it is possible to design asynchronous circuits which are hazard-free. The test procedure uses scan techniques to apply each pair of test vectors to detect an appropriate path-delay-fault in a robust and hazard-free manner. In such asynchronous circuits every latch can be scanned to increase controllability and observability of its inputs and outputs. It was shown that there is a negligible area or delay penalty required to achieve robust path-delay-fault testability. Nevertheless, the test approach imposes strict limitations on the speed at which the circuit can be tested.

A scan test technique for asynchronous sequential logic circuits synthesized from either a Huffman model or a signal transition graph has been reported [Wey93]. Scan-latches similar to the LSSD polarity hold latches (see Appendix B) are used to design the memory elements of the testable sequential circuit. The proposed scan test procedure provides for the detection of stuck-at faults in the asynchronous sequential circuit, reducing the test generation problem to one of just testing the combinational circuit.

2.3.2 Testing micropipelines

There are a few works devoted to fault modelling and fault testing problems in micropipelines [Pag92, Khoc94]. Stuck-at faults in the control part, combinational logic blocks and latches of the micropipeline have been considered [Pag92].

Faults in the control part

These are faults on the inputs and outputs of the C-elements and the request and acknowledge lines of the micropipeline (see Figures 1.5 and 1.6). As was shown the micropipeline moves through at most one step and then halts in the presence of a stuck-at 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 [Cheng89, 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 micropipeline are compared with good responses.

Faults in the latches

It is assumed that the combinational logic obtained after setting the latches in the transparent mode has no redundant faults.

Single stuck-at faults. Any stuck-at fault on the inputs or outputs of the latch is equivalent to the appropriate fault in the combinational logic. A stuck-at fault on the control lines of the latch (see Figure 1.6) prevents the generation of any events in the latch. This causes the micropipeline to halt. The absence of activity in the micropipeline can easily be identified and, hence, there is no need for test generation for such faults.

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 capture position. For example, a stuck-at-1 fault on the *enable* input of the latch shown in Figure 1.6 sets the faulty latch in capture mode permanently. As an effect of this fault, the faulty bit can be captured 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 be easily 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-0 fault on the enable input of the latch illustrated in Figure 1.6 makes the faulty latch transparent permanently. A two pattern test is required to detect this kind of fault.

Consider a stuck-at-pass fault on a bit of the *k*-th latch of the micropipeline. Let the faulty bit of the latch be connected to line *l* of the complete combinational network (CN) obtained by switching all the latches into the pass mode. The test for the faulty bit consists of two patterns, say P_1 and P_2, which are applied one after another. Pattern P_1 is the test pattern for a stuck-at-*z* fault on line *l* of *CN*, where *z* is a logical value which is equal to 1 or 0. Pattern P_2 is the test vector which forces line *l* to be set to logic value *z*. These test patterns can be obtained easily by means of standard test generation methods for combinational circuits.

The test procedure for detecting a stuck-at-pass fault in the micropipeline is the following:

1. Apply pattern P_1 to the inputs of the micropipeline while all the latches are in the pass mode. Put the *k*-th latch in the capture mode. As a result, line *l* has been set to logic $\bar{z}$. The response is observed at the outputs of the micropipeline.

2. Apply pattern P_2 to the inputs of the micropipeline. Thus, line *l* of CN has been driven to logic *z* since the faulty bit of the latch is connected to line *l* of CN; other lines of the latch are at their logical values corresponding to pattern P_1. This causes at least one output of the micropipeline to be different from the fault-free response.

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 test technique described allows the detection of all the stuck-at faults and bundling constraint violations in micropipelines. However, this scan test technique has been developed only for micropipelines which use a two-phase transition signalling protocol. The proposed scan test interface uses clocks produced by a clock generator which is not always available in asynchronous VLSI designs.

2.4 Summary

The most widely used fault models chosen to describe fault behaviours of asynchronous circuits are stuck-at and delay (transition) faults. The more strict the limitations that are imposed to the delays in the asynchronous circuits, the more thorough the testability analysis that is required.

Testing asynchronous VLSI designs presents new problems which must be addressed before their commercial potential can be realized. The logic redundancy which is involved in the design of asynchronous circuits to ensure their hazard-free behaviour makes their testing difficult or even impossible. Testing for hazards and races in circuits without a synchronization clock is not trivial.

The scan test technique has been adapted well to the testing of asynchronous circuits. However, design for testability problems for asynchronous circuits have not been well addressed because of the difficulties described above.

Chapter 3 : Power Consumption and Testability of CMOS VLSI Circuits

After fabrication, a digital circuit must be tested to ensure that it is fault free. This is not an easy task since the increasing number of logic elements placed on a chip leads to a reduction in the controllability and observability of the internal nodes of the circuits. DFT methods have been developed for digital circuits to facilitate their testing for fabrication faults (see Appendix B). Since DFT methods affect the circuit design, this raises the question: “How do DFT methods affect the power consumption of a circuit?”.

A relationship between the power consumption and the testability of CMOS VLSI circuits is demonstrated in this chapter. The method used to estimate this correlation is based on elements of information theory. It is shown that design for low power consumption and design for testability are in direct conflict. The resolution of this conflict lies in separating the testing issues from the low power issues by giving the circuit distinct operating and test modes.

3.1 Power consumption of CMOS circuits

The rapid development of CMOS technology makes transistors smaller allowing a chip to incorporate ever larger numbers of them [Weste93]. CMOS VLSI circuits are increasingly used in portable environments where power and heat dissipation are vital issues. Examples of such applications are portable calculators, digital watches, mobile computer systems, etc. As a result, the power dissipation of CMOS VLSI circuits is a growing concern for design engineers.

The power dissipated by CMOS circuits can be divided into three categories [Weste93, Deng94]:

- static power consumption due to leakage current (PW_{stat});
- dynamic power dissipation caused by switching transition current (PW_{sw});
- transient short-circuit current (PW_{sc}).

The total power dissipation of a CMOS circuit can therefore be represented by the following sum:

$$PW_{total} = PW_{stat} + PW_{sw} + PW_{sc} . \qquad (3.1)$$

In 'well-designed' data processing circuits the switching power is typically from 70% to 95% of the total power consumption (if the circuit is badly-designed the proportion of static and short-circuit power dissipation increases) [Veen84].

The majority of power estimation tools are oriented towards calculating only the average switching power of CMOS circuits using the following formula [Deng94]:

$$PW_{sw} = f \cdot V_{dd}^2 \cdot \sum_{i=1}^{M} (P_{tr_i} \cdot C_i) , \qquad (3.2)$$

where f is the clock frequency for synchronous circuits or the parameter which estimates the average circuit activity for asynchronous circuits; V_{dd} is the power supply voltage; M is the total number of nodes in the circuit; C_i is the i-th nodal capacitance; P_{tr_i} is the transition probability of the i-th node.

3.2 Information theory and digital circuits

Output node capacitances create memories in static CMOS circuits. The circuit itself can also have state holding elements. Let us consider a circuit with one output. This circuit has only two possible states: zero and one. The circuit changes its states during the application of patterns to its inputs. This process can be represented by the Markov chain shown in Figure 3.1 regardless of whether the circuit is a sequential one or a combinational one. The Markov chain contains two states marked by zero and one. The transition probabilities between the states are placed on the corresponding arcs of the chain where $p_i(j)$ denotes the probability of the transition from state i to state j (i,j=0,1).

The following system of equations describes the behaviour of the Markov chain illustrated in Figure 3.1:

$$\begin{aligned} P_0 &= P_0 \cdot p_0(0) + P_1 \cdot p_1(0) , \\ P_1 &= P_0 \cdot p_0(1) + P_1 \cdot p_1(1) , \\ P_1 + P_0 &= 1 , \end{aligned} \tag{3.3}$$

where P_0 and P_1 are the probabilities of state 0 and state 1, respectively.

Note that for the Markov chain shown in Figure 3.1

$$p_0(1) + p_0(0) = 1 , \tag{3.4}$$

$$p_1(0) + p_1(1) = 1 . \tag{3.5}$$

Solving system (3.3) the probabilities of states 0 and 1 can be found as

$$P_1 = p_0(1) / (p_0(1) + p_1(0)) , \tag{3.6}$$

$$P_0 = p_1(0) / (p_0(1) + p_1(0)) . \tag{3.7}$$

Thus, only two transition probabilities $p_0(1)$ and $p_1(0)$ are required to describe fully the behaviour of the circuit with one output.

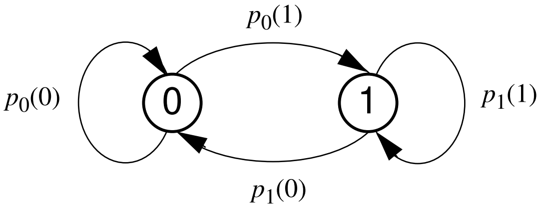


Figure 3.1: *Markov chain representing the mechanism of changing the state of a circuit with one output*

For combinational circuits

$$p_0(1) = p_1(1) = p, \tag{3.8}$$

$$p_1(0) = p_0(0) = q, \tag{3.9}$$

where $p + q = 1$. As a result, $P_1 = p$ and $P_0 = q$.

Figure 3.2 shows six two-input logic blocks and their transition probabilities. It is assumed that the input signal probabilities p_a and p_b are independent. The last three logic blocks shown in Figure 3.2 are Muller C-elements [Brzo95b].

The logic function of the two-input symmetric C-element (the first C-element) is

$$c_t = a \cdot b + a \cdot c_{t-1} + b \cdot c_{t-1}, \tag{3.10}$$

where *a* and *b* are the inputs; c_t is the output of the C-element at time *t*.

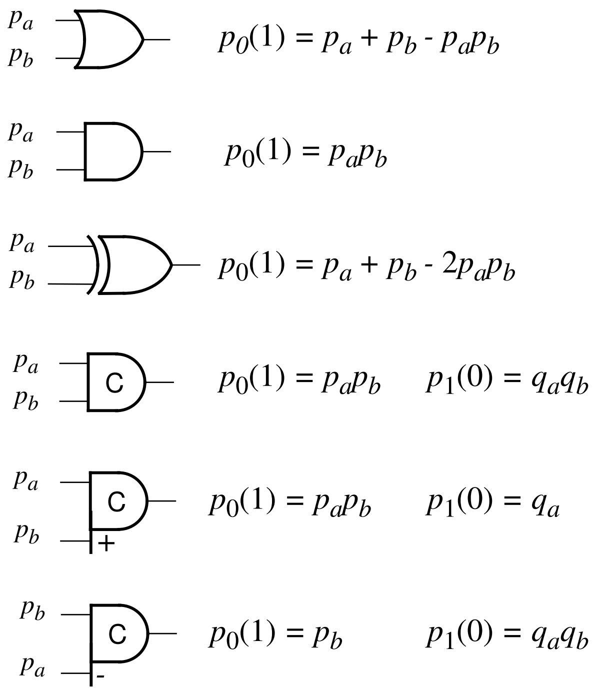


Figure 3.2: *Logic elements and their transition probabilities*

The output of the symmetric C-element is high or low when both inputs are high or low, respectively. The C-element preserves its current state when the inputs are different. In order to calculate the output signal probability of the two-input symmetric C-element equations (3.6) and (3.7) can be used. As a result,

$$P_1 = p_a \cdot p_b / (p_a \cdot p_b + q_a \cdot q_b) \ , \tag{3.11}$$

$$P_0 = q_a \cdot q_b / (p_a \cdot p_b + q_a \cdot q_b) \ . \tag{3.12}$$

The last two C-elements are asymmetric C-elements which perform different functions:

- for the second C-element:

$$c_t = a \cdot b + a \cdot c_{t-1}, \tag{3.13}$$

- for the third C-element:

$$c_t = b + a \cdot c_{t-1} . \tag{3.14}$$

The output of the asymmetric C-element which performs according to function (3.13) is high if both its inputs are high and low if only input *a* is low. It keeps its current state zero when input *a* or *b* is low and preserves state one if input *a* is high. The output of the asymmetric C-element whose behaviour is described by function (3.14) is low if both its inputs are low and high if input *b* is high. It does not change its current state zero if input *b* is low and preserves its state one if input *a* or *b* is high. The state probabilities of the asymmetric C-elements can easily be found using equations (3.6) and (3.7).

Let us estimate the average information content on the output of a circuit. According to information theory, the average information content or entropy (*H*) of a discrete finite state type source is [Shan64, Rosie66]

$$H = \sum_i P_i H_i = -\sum_{i,j} P_j p_j(i) \log_2 p_j(i) \ , \tag{3.15}$$

where P_j is the probability of state *j*; $p_j(i)$ is the probability of the transition from state *j* to state *i*.

Thus, the average output information content of the circuit described by the Markov process shown in Figure 3.1 is calculated as follows:

$$H = P_0 H_0 + P_1 H_1, \tag{3.16}$$

i.e.,

$$\begin{aligned} H = & -P_0 p_0(0) \log_2 p_0(0) - P_0 p_0(1) \log_2 p_0(1) \\ & -P_1 p_1(0) \log_2 p_1(0) - P_1 p_1(1) \log_2 p_1(1) . \end{aligned} \tag{3.17}$$

The average information content on the output of the combinational circuit is equal to

$$H = -p \log_2 p - (1-p) \log_2 (1-p) . \tag{3.18}$$

Let $p_0(1) = x$ and $p_1(0) = y$ then

$$H(x, y) = -\frac{x}{x+y} H(y) - \frac{y}{x+y} H(x) , \tag{3.19}$$

where

$H(y) = y \log_2 y + (1-y) \log_2 (1-y)$ and $H(x) = x \log_2 x + (1-x) \log_2 (1-x)$.

In order to find an extremum of function *H*(*x*,*y*) the following system of two equations must be solved:

$$\begin{aligned} \frac{\partial}{\partial x} H(x, y) &= 0, \\ \frac{\partial}{\partial y} H(x, y) &= 0. \end{aligned} \tag{3.20}$$

System (3.20) can be modified as

$$\begin{aligned} (x+y)(\log_2(1-x) - \log_2 x) + H(y) - H(x) &= 0, \\ (x+y)(\log_2(1-y) - \log_2 y) + H(x) - H(y) &= 0. \end{aligned} \tag{3.21}$$

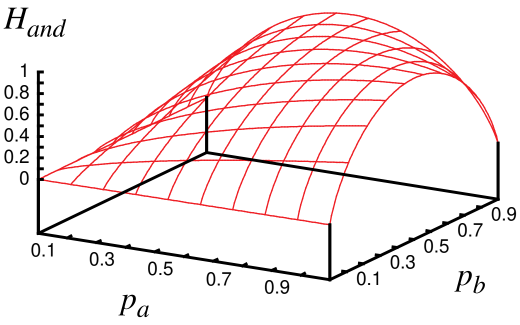


Figure 3.3: *Average ouput information content of the two-input AND gate*

The only solution of system (3.21) is x=y=0.5. It is easy to show that

$$Max(H(x,y)) = H(0.5,0.5) = 1.$$

Thus, the maximum information content can be reached when all the transitions between the various states of the circuit are equiprobable. This result can easily be generalised for any number of circuit states.

Figures 3.3 and 3. 4 illustrate graphically the dependencies between the average output information content (H) and the input signal probabilities p_a and p_b of the two-input AND and XOR gate, respectively. Figures 3.5 and 3.6 show graphs $H_c(p_a,p_b)$ and $H_{ac}(p_a,p_b)$ of the two-input symmetric and asymmetric C-elements which perform according to equations (3.10) and (3.13), respectively. Note that the maximum information content is reached when the output transition probabilities of the logic elements are equal to 0.5. For instance, H_{and}=1 when p_ap_b=0.5. For the XOR gate (see Figure 3.4),

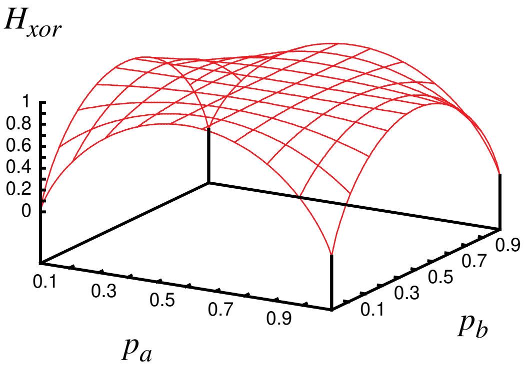


Figure 3.4: *Average ouput information content of the two-input XOR gate*

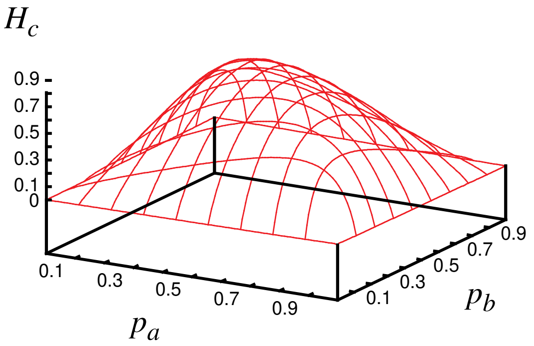


Figure 3.5: *Average ouput information content of the two-input symmetric C-element*

H_{xor}=1 when p_a=0.5 or p_b=0.5. The maximum value of the average output information content of the symmetric C-element (see Figure 3.5) never reaches 1 ($MAX(H_c)$=0.81) even when its probability of state 1 or 0 is 0.5 (when p_a=p_b=0.5). In fact, the transition probabilities of the symmetric C-element can never be equal to 0.5. The average output information content of the asymmetric C-element described by equation (3.13) reaches 1 at point (p_a=0.5; p_b=1) (see Figure 3.6). This is because the asymmetric C-element works as a repeater of the information from its input *a* when *b*=1.

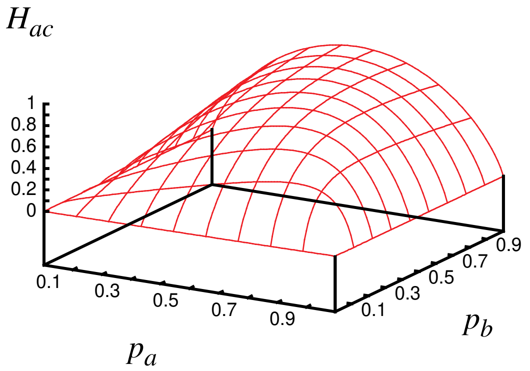


Figure 3.6: *Average ouput information content of the two-input asymmetric C-element*

3.3 Information content and transition probability

Let us consider the following expression:

$$\log_2 v = \frac{\ln v}{\ln 2} \quad . \tag{3.22}$$

It is known that

$$-\ln(1-v) = \sum_{r=1}^{\infty} \frac{v^r}{r},$$

where $0 \le v < 1$ [Bost84].

Hence,

$$-\ln v = \sum_{r=1}^{\infty} \frac{(1-v)^r}{r} = (1-v) \sum_{r=1}^{\infty} \frac{(1-v)^{r-1}}{r}. \tag{3.23}$$

Equation (3.23) can be substituted by the following inequality:

$$-\ln v \ge 1 - v \tag{3.24}$$

since $\sum_{r=1}^{\infty} \frac{(1-v)^{r-1}}{r} \ge 1$ for $0 < v < 1$.

Taking into account equation (3.22) and inequality (3.24), equation (3.17) can easily be transformed into the following inequality:

$$\begin{aligned} H\ln 2 \ge\ & P_0 p_0(0)\,[1-p_0(0)] + P_0 p_0(1)\,[1-p_0(1)] \\ & + P_1 p_1(0)\,[1-p_1(0)] + P_1 p_1(1)\,[1-p_1(1)]. \end{aligned} \tag{3.25}$$

Inequality (3.25) can be simplified bearing in mind the basic relationships between transition probabilities of the Markov process described by equations (3.4) and (3.5). Hence,

$$\begin{aligned} H\ln 2 \ge\ & P_0 p_0(0) p_0(1) + P_0 p_0(1) p_0(0) \\ & + P_1 p_1(0) p_1(1) + P_1 p_1(1) p_1(0) \end{aligned}$$

or

$$H \ge 2\,[P_0 p_0(0) p_0(1) + P_1 p_1(1) p_1(0)] / \ln 2. \tag{3.26}$$

Substituting the probabilities of states 1 and 0 by expressions (3.6) and (3.7), respectively, inequality (3.26) can be written as

$$H \ge \gamma \frac{2 p_0(1) p_1(0)}{p_0(1) + p_1(0)}\ , \tag{3.27}$$

where

$$\gamma = \frac{p_0(0) + p_1(1)}{\ln 2} \text{ and } \gamma \geq 0 .$$

The signal transition probability (P_{tr}) on the output of the circuit is calculated as follows:

$$P_{tr} = P_0 p_0(1) + P_1 p_1(0) . \tag{3.28}$$

The following equation can be derived by substituting state probabilities P_1 and P_0 in equation (3.28) by expressions (3.6) and (3.7), respectively:

$$P_{tr} = \frac{2 p_0(1) p_1(0)}{p_0(1) + p_1(0)} . \tag{3.29}$$

The comparison of expressions (3.27) and (3.29) allows us to conclude that

$$H \geq \gamma P_{tr} . \tag{3.30}$$

For combinational circuits (see equations 3.8 and 3.9) the following expressions can easily be obtained:

$$H \geq \frac{2pq}{\ln 2}, P_{tr} = 2pq, \gamma = (\ln 2)^{-1} . \tag{3.31}$$

Therefore, the average output information content and the output signal transition probability correlate strongly.

Consider the following function:

$$F(x, y) = \gamma(x, y) P_{tr}(x, y) = \frac{2xy(2 - x - y)}{(x + y) \ln 2} , \tag{3.32}$$

where variables *x* and *y* have the same meaning as in equation (3.19).

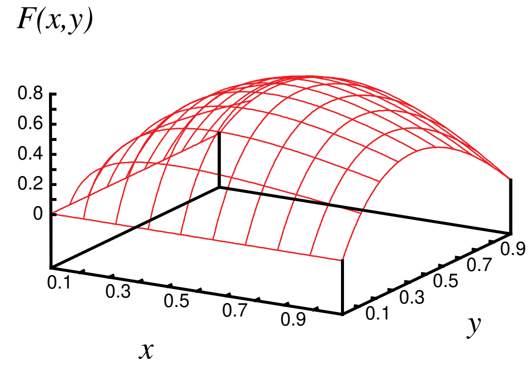


Figure 3.7: *Graph of function F(x,y)*

In order to find an extremum of function $F(x,y)$ the following system of two equations must be solved:

$$\frac{\partial}{\partial x}F(x, y) = 0, \qquad \frac{\partial}{\partial y}F(x, y) = 0. \tag{3.33}$$

After trivial manipulations system (3.33) is modified to

$$2y/(x+y)^2 - 1 = 0, \qquad 2x/(x+y)^2 - 1 = 0. \tag{3.34}$$

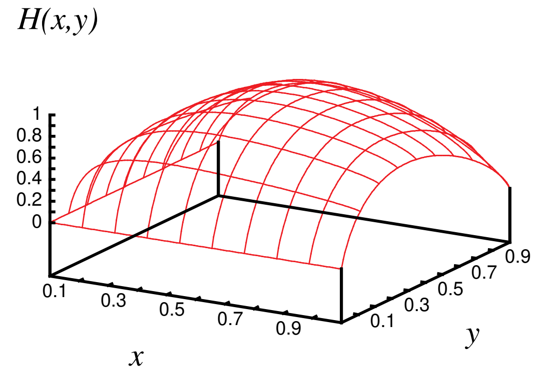


Figure 3.8: *Graph of function H(x,y)*

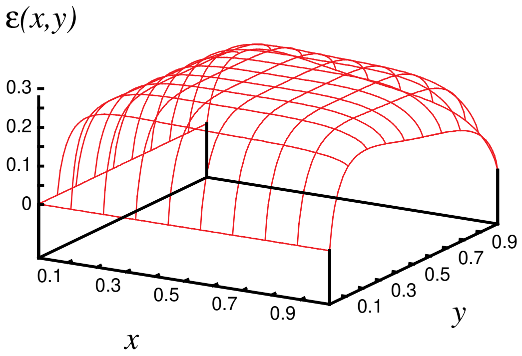


Figure 3.9: *Graph of function* ε*(x,y)*

The solution of system (3.34) is x=y=0.5. It can easily be shown that

$$Max(F(x, y)) = F(0.5, 0.5) = (2\ln 2)^{-1}.$$

Thus, functions $H(x,y)$ and $F(x,y)$ exhibit very similar behaviour. The only difference is that $H(x,y)$ is always greater than or equal to $F(x,y)$ when $0 \le x, y \le 1$. Figures 3.7 and 3.8 illustrate graphs of functions $H(x,y)$ and $F(x,y)$ respectively.

In order to estimate how close functions $H(x,y)$ and $F(x,y)$ are, we investigate the following function:

$$\varepsilon(x, y) = H(x, y) - F(x, y) , \tag{3.35}$$

where $\varepsilon(x,y)$ is the absolute error between function $H(x,y)$ and its approximation $F(x,y)$.

It is trivial to prove that the maximum of function $\varepsilon(x,y)$ is reached at the point when x=y=0.5 ($Max(\varepsilon(x,y))$=0.28). As a result, the maximum absolute error of approximation $F(x,y)$ can never be more than 28%. Figure 3.9 shows a graph of function $\varepsilon(x,y)$.

3.4 Discussion

It has been shown that the testability of a circuit is proportional to its output information content [Agra81]. This means that the more information the nodes of the digital circuit carry to its outputs, the more testable the circuit is, and vice versa. The dynamic power consumption of a CMOS circuit is also proportional to the transition probabilities of its nodes (see equation (3.2)). Hence, the more testable a circuit is, the more dynamic power it dissipates. The converse statement, that the more power consuming the circuit is the more testable it is, can be justified only if the increase in the circuit power dissipation is caused by increased activity in its nodes.

Shen et al. observed that both the random pattern testability and the power dissipation of a combinational logic network are linked to the signal probabilities of its nodes [ShenG92]. They proposed probability modification techniques to restructure the combinational networks to improve both their transition signal probabilities and their power dissipations but these delivered insignificant improvements.

Williams and Angell showed that increasing the transition probabilities in the nodes of a circuit improves its controllability and, therefore, its testability [Will73]. The testability of the circuit can also be improved by inserting test points at some of its nodes, increasing the observability of the circuit. Improving controllability has a direct power cost due to the increased number of transitions, whereas improving observability only marginally increases the power dissipation due to an increased switched capacitance.

Clearly, the power dissipation of a digital circuit is of interest principally when it is in operation in its intended application. Power consumption during test is not usually important. An approach which offers a compromise between testability and power consumption is to design the circuit to work in two distinct operating modes. In normal operation mode, it performs the specified function dissipating minimal or close to minimal switching energy. The circuit is set to test mode to make its testing simple. During the test, the circuit is tested extensively dissipating more energy.

3.5 Summary

It has been shown that the testability of CMOS VLSI circuits correlates with the switching power that they dissipate. The mathematical dependencies presented allow us to conclude that improving the testability features of a CMOS circuit leads to an increase in its switching power dissipation. As a result, design for testability and design for low power dissipation are in direct conflict. This conflict can be resolved by separating the testing issues from low power issues so that the circuit can operate in normal operation and test modes. This is the approach that has been used in the examples considered in the following chapters.

Chapter 4 : Designing C-elements for Testability

4.1 Introduction

C-elements are used widely in asynchronous VLSI designs. As a result, the correct functioning of C-elements is important for the whole asynchronous system in which they are used. Brzozowski and Raahemifar showed that the testing of stuck-at faults in different designs of the C-element is not trivial [Brzo95b]. It has been observed that stuck-at faults of the C-element fall into one of the following categories:

1) faults that are detectable by logic testing since they halt the circuit or change its function;

2) faults that are detectable by delay measurements;

3) faults that may result in an oscillation;

4) faults that may destroy the speed-independence of the circuit;

5) faults that are detectable by measuring the current.

The goal of this chapter is to present different CMOS implementations of static symmetric and asymmetric C-elements which provide for the detection of line stuck-at and transistor stuck-open faults using logic testing. Simulation results for the extracted layouts of all the CMOS C-element designs considered in this chapter can be found in Appendix C.

4.2 Symmetric C-element CMOS designs

Figure 4.1a shows a symbolic representation of the two-input symmetric C-element with inputs *a*, *b* and output *c*. As was mentioned in chapter 1 the symmetric C-element is a state holding element the output of which is high when its inputs are high and low when its inputs are low. Any other input combination does not change the state of the C-element. There are different ways to implement static symmetric C-elements in CMOS technology. Figure 4.1b shows a CMOS C-element which performs according to equation (3.10). For example, when a=1 and b=1 there is a path between V_{ss} and the input of inverter *inv*. As a result, output *c* of the C-element is high and feedback *n*-type transistor N_5 is on. If the inputs of the C-element are different there is always a connection between V_{ss} and the input of inverter *inv*.

Equation (3.10) can be rewritten in the following form:

$$c_t = (a + b) \cdot c_{t-1} + a \cdot b \quad . \tag{4.1}$$

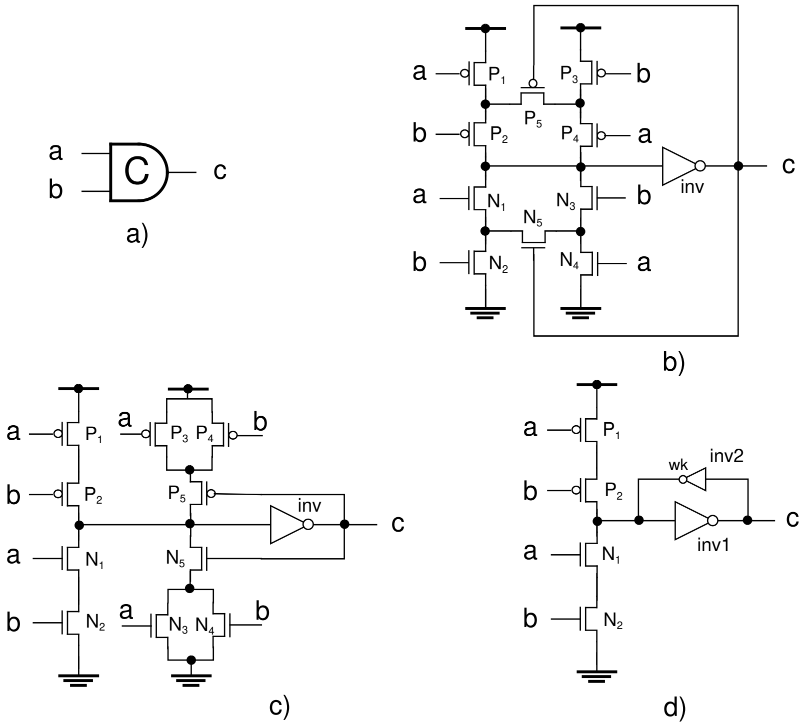


Figure 4.1: *Symmetric C-elements: a) symbol of the two-input C-element; b) and c) static C-elements; d) pseudo-static C-element*

A CMOS implementation of the symmetric C-element which performs according to equation (4.1) is illustrated in Figure 4.1c. This C-element performs in a similar way to the one shown in Figure 4.1b. Both CMOS implementations of the symmetric C-element require 12 transistors.

The symmetric C-element shown in Figure 4.1d is a pseudo-static C-element which performs according to equation (3.10) but in a way similar to that of a dynamic C-element. The only difference is that the weak feedback inverter *inv2* is inserted into the symmetric C-element to create a CMOS memory. If c_{t-1}=0, a_t=1 and b_t=1 then the input of inverter *inv1* is driven to low since the strength of *n*-type input stack (transistors N_1 and N_2) is higher than that of *p*-type stack of weak inverter *inv2*. As a consequence, output *c* of the C-element goes high keeping the input of inverter *inv1* in low. If the input transistor stacks are disabled by different input signals the current state of the C-element is kept unchanged. The implementation of the pseudo-static C-element requires 8 transistors.

For test purposes the inputs and the output of the C-element are assumed to be controllable and observable respectively. It has already been shown in chapter 2 that some single stuck-at faults in the CMOS inverter are not detectable by logic testing. Therefore, such faults are not detectable in the CMOS designs depicted in Figure 4.1. There is a fundamental problem in the testing of static C-elements for stuck-open faults. Stuck-open faults in the feedback transistors of the static symmetric C-element transform it into a dynamic one. For instance, if the output of the weak inverter in the C-element shown in Figure 4.1d is disconnected from the input of inverter *inv1* the faulty C-element still performs according to equation (3.10), but as a dynamic circuit. This kind of fault can be identified only by ‘slow’ testing which ‘waits’ until the output of the faulty C-element is discharged completely. This degrades the test performance. CMOS structures of the symmetric C-element which provide for detection of single line stuck-at and transistor stuck-open faults are considered in the following sections.

4.2.1 Testing for stuck-open faults

The testing of stuck-open faults in the feedback transistors of the static symmetric C-element is implemented by driving them using an extra input and observing test results on the gate output. The structure shown in Figure 4.1b is most suitable as the starting point for the testable implementation. Figure 4.2 illustrates a CMOS design of the symmetric C-element where single transistor stuck-open faults are detectable. The C-element contains an additional weak inverter (transistors P_6 and N_6) the output of which can be overdriven by a logical value applied to pin *m*. The proposed implementation of the symmetric C-element requires 14 transistors.

Tests for the transistor stuck-open faults of the C-element are shown in Table 4.1. If m_i=0 or m_i=1 node *m* is used to drive feedback transistors P_5 and N_5 with a logical zero or one respectively. If m_i=z then node *m* is in a high impedance mode. A hyphen in the column headed m_o means that node *m* does not carry any diagnostic information since it is driven by an external logical value. It can be seen from Table 4.1 that faults P_5-SO and N_5-SO, which transform the static behaviour of the symmetric C-element into a dynamic one, are detectable by sets of two tests with no need for 'slow' testing.

Driving the feedback transistors of the symmetric C-element transforms its sequential function into a combinational one depending on the logical value applied to pin *m*. Table 4.2 contains the operation modes of the symmetric C-element illustrated in Figure 4.2. When m_i=0 or m_i=1 the C-element is transformed into an AND or OR gate respectively.

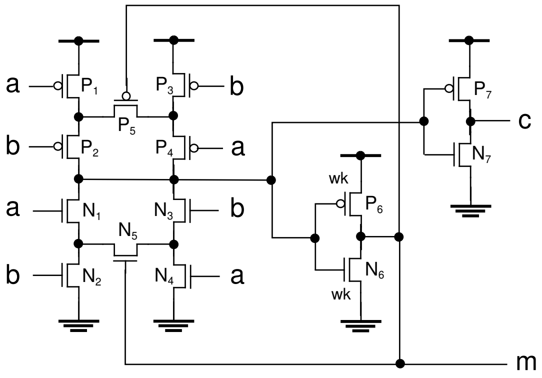


__Figure 4.2:__ Locations of stuck-open faults in the static symmetric C-element

Table 4.1: *Tests for stuck-open faults of the symmetric C-element in Figure 4.2*

Stuck-open faults	Test sequences			Outputs of the fault-free C-element		Outputs of the faulty C-element	
	a	b	m_i	c	m_o	c	m_o
P_2,P_3,P_5,N_7	1	1	0	1	-	1	-
	1	0	0	0	-	1	-
P_1,P_4,P_5,N_7	1	1	0	1	-	1	-
	0	1	0	0	-	1	-
N_2,N_3,N_5,P_7	0	0	1	0	-	0	-
	0	1	1	1	-	0	-
N_1,N_4,N_5,P_7	0	0	1	0	-	0	-
	1	0	1	1	-	0	-
P_6	0	0	z	0	0	0	0
	1	1	z	1	1	1	0
N_6	1	1	z	1	1	1	1
	0	0	z	0	0	0	1

This transformation of the sequential function of the C-element gives a reduction in the number of state holding elements in the asynchronous circuit under test and makes its testing easier. On the other hand, when the circuit (in Figure 4.2) acts as a symmetric C-element output *m* can be used as a test point increasing its observability inside the asynchronous circuit.

Table 4.2: *Operation modes of the C-element in Figure 4.2*

Function	Inputs			Outputs	
	a	b	m_i	c_t	m_o
AND	0	0	0	0	-
	0	1	0	0	-
	1	0	0	0	-
	1	1	0	1	-
OR	0	0	1	0	-
	0	1	1	1	-
	1	0	1	1	-
	1	1	1	1	-
Symmetric Muller C	0	0	z	0	0
	0	1	z	c_{t-1}	c_{t-1}
	1	0	z	c_{t-1}	c_{t-1}
	1	1	z	1	1

The use of a weak inverter in the symmetric C-element illustrated in Figure 4.2 is not efficient in terms of power consumption in test mode. For instance, if during the test pin *m* is kept to one and *a*=*b*=0 then there is a path between V_{dd} and V_{ss} which increases the power dissipation of the C-element. The power consumption can be reduced if the weak inverter is replaced by a tristate inverter which is disabled during the test by an extra control input. It is easy to show that in this case such a C-element remains testable for transistor stuck-open faults.

4.2.2 Testing for stuck-at faults

The symmetric C-element illustrated in Figure 4.2 is not testable for line stuck-at faults since it has inverters which are not fully testable for all single line stuck-at faults (see chapter 2). The implementation of the symmetric C-element shown in Figure 4.3 incorporates inverters which are testable for single stuck-at faults using two additional test inputs *tp* and *tn*. Two extra transistors controlled by inputs *tp* and *tn* are inserted in the feedback paths of the C-element for testability purposes. In normal operation mode, when *tp*=0 and *tn*=1 the circuit performs in the same manner as the one in Figure 4.3.

It is assumed that all the inputs and outputs of the C-element are controllable and observable during the test. Table 4.3 contains tests which are derived to detect its single line stuck-at faults. As was observed previously in chapter 2 the detection of all single

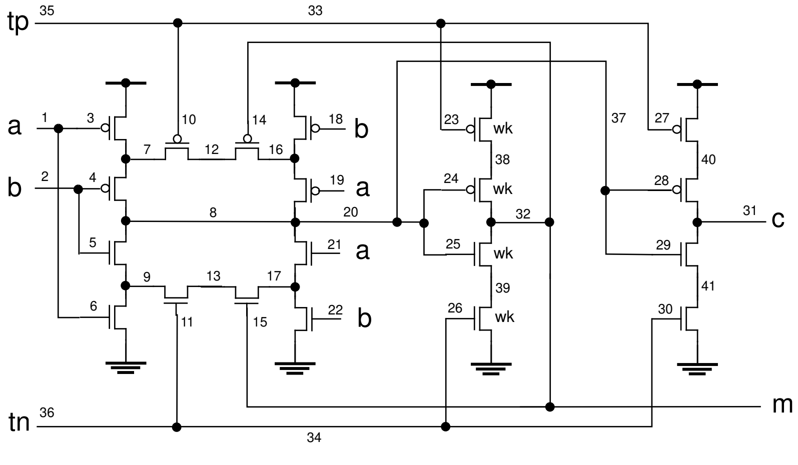


Figure 4.3: *Locations of stuck-at faults in the static C-element*

***Table 4.3:** Tests for stuck-at faults of the C-element in Figure 4.3*

No.	Single stuck-0 faults	Single stuck-1 faults	Test sequences			Outputs of the fault free circuit		Outputs of the faulty circuit	
			a&b	t&tp	m_i	c	m_o	c	m_o
1	31,32,38,40	8,20,37	11	10	z	1	1	0	0
2	8,20,37	31,32,39,41	00	10	z	0	0	1	1
3	7’,12’,16’,30	2,4,7”,10,12”,14,18	11	10	0	1	-	1	-
			10	10	0	0	-	1	-
4	7”,12”,16”	1,3,10,14,12’,16’,19	11	10	0	1	-	1	-
			01	10	0	0	-	1	-
5	2,5,9”,11,13”,15,22	9’,13’,17’,27	00	10	1	0	-	0	-
			01	10	1	1	-	0	-
6	1,6,11,13’,15,17’,21	9”,13”,17”	00	10	1	0	-	0	-
			10	10	1	1	-	0	-
7	25,26,29,30,34,36		11	10	z	1	1	1	1
			00	10	z	0	0	1	1
8		23,24,27,28,33,35	00	10	z	0	0	0	0
			11	10	z	1	1	0	0
9	4,10,18	16”	11	10	0	1	-	1	-
			01	11	0	1	-	0	-
10	3,10,19	7’	11	10	0	1	-	1	-
			10	11	0	1	-	0	-
11	17”	5,11,22	00	10	1	0	-	0	-
			10	00	1	0	-	1	-
12	9’	6,11,21	00	10	1	0	-	0	-
			01	00	1	0	-	1	-
13	14		11	00	1	1	-	1	-
			10	00	1	1	-	1	-
			10	11	0	1	-	0	-
14		15	00	11	0	0	-	0	-
			10	11	0	0	-	0	-
			10	00	1	0	-	1	-
15	23,27,33,35	38,40	00	10	z	0	0	0	0
			11	11	z	0	0	1	1
16	39,41	26,30,34,36	11	10	z	1	1	1	1
			00	00	z	1	1	0	0
17	24,28		00	11	z	0	0	0	0
			00	00	z	0	0	1	1
18		25,29	11	00	z	1	1	1	1
			11	11	z	1	1	0	0

line stuck-at faults in a CMOS design guarantees the detection of all its single stuck-open faults. For instance, fault 18-SA1 is equivalent to keeping the appropriate *p*-type transistor off permanently which means its permanent disconnection from V_{dd}. As a result, the proposed CMOS design is testable for both stuck-at and stuck-open faults.

4.3 Static asymmetric C-elements

Asymmetric C-elements have already been mentioned in chapter 3. Asymmetric C-elements are used widely to improve the performance of the asynchronous control logic in micropipelines [Brzo95b, Farn95, Furb96]. These C-elements are set to a particular state when both signals are one or zero and set to the negated state only by one input. Figures 4.4a and 4.4b demonstrate a symbolic representation and a gate level implementation of the OR-AND type asymmetric C-element. The OR-AND type asymmetric C-element illustrated in Figure 4.4b performs according to the following equation:

$$c_t = a \cdot (b + c_{t-1}) \ . \tag{4.2}$$

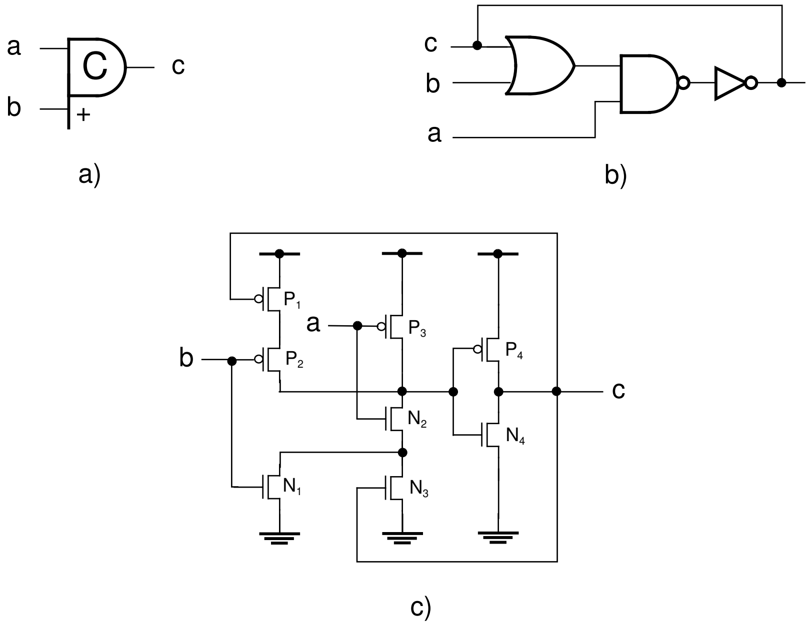


Figure 4.4: *Static OR-AND type asymmetric C-element: a) symbol; b) gate level representation; c) CMOS implementation*

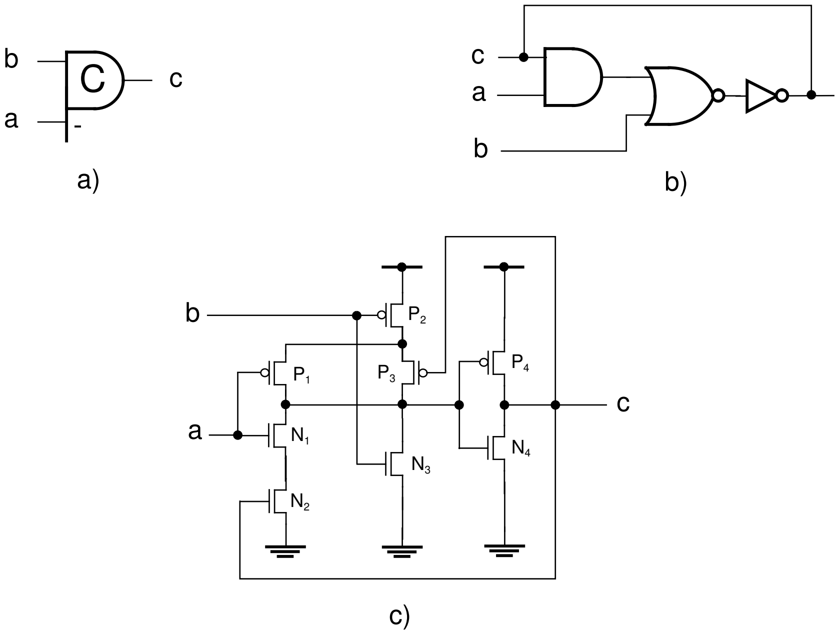


Figure 4.5: *Static AND-OR type asymmetric C-element: a) symbol; b) gate level representation; c) CMOS implementation*

The output of such an asymmetric C-element is set to high when both its inputs are high and set to low if its input *a* is low. It keeps its current state when its input *a* is high and input *b* is low. A static CMOS implementation of the OR-AND type asymmetric C-element is illustrated in Figure 4.4c.

Figures 4.5a and 4.5b show a symbolic representation and a gate level implementation of the AND-OR type asymmetric C-element. This C-element performs according to equation (3.14). The output of the AND-OR type asymmetric C-element is set to high if its input *b* is high and set to low when both its inputs are low. It preserves its current state when input *a* is high and input *b* is low. A static CMOS implementation of the AND-OR type asymmetric C-element is shown in Figure 4.5c.

These two types of asymmetric C-elements were implemented in CMOS technology on a $1\mu m$, double layer metal CMOS process and simulated using *SPICE* analysis in the *Cadence* CAD environment. Simulation results obtained from their extracted layouts can be found in Appendix C.

Testing for stuck-at and stuck-open faults in asymmetric C-elements is not easy. For instance, such stuck-open faults as N_3-SO (in Figure 4.4c) and P_3-SO (in Figure 4.5c) can be identified only by ‘slow’ testing since they transform the correspondent static asymmetric C-elements into dynamic ones. As was previously mentioned a stuck-at-0 fault on the gate of transistor P_4 and stuck-at-1 fault on the gate of transistor N_4 of the asymmetric C-elements are not detectable by logic testing. Thus, extra design effort is required to make the asymmetric C-elements testable.

4.3.1 Testing for stuck-open faults in asymmetric C-elements

Figures 4.6a and 4.6b illustrate the designs of asymmetric C-elements testable for transistor stuck-open faults. The main approach to the testing of stuck-open faults in asymmetric C-elements is the same: the feedback transistors are driven by an external control

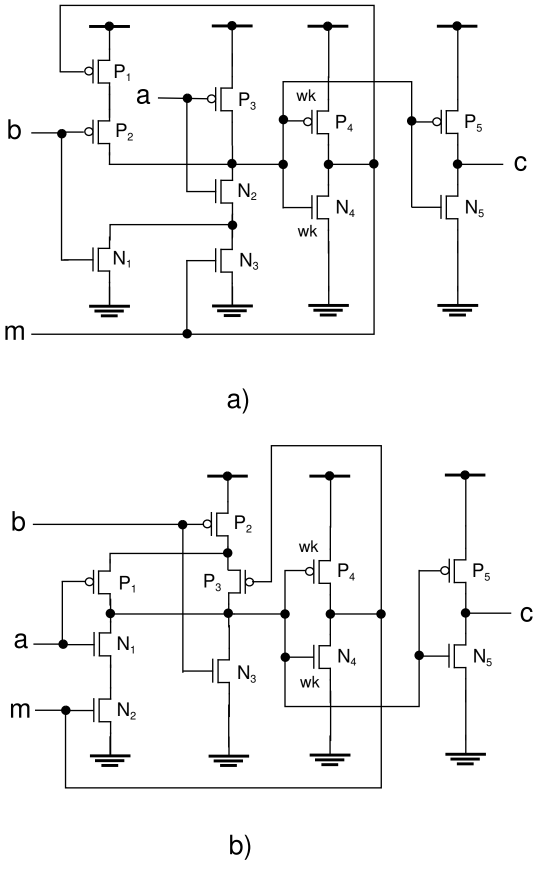


Figure 4.6: *Static asymmetric C-elements testable for stuck-open faults: a) OR-AND type asymmetric C-element; b) AND-OR type asymmetric C-element*

***Table 4.4:** Tests for stuck-open faults of the OR-AND type asymmetric C-element*

Stuck-open faults	Test sequences			Outputs of the fault-free C-element		Outputs of the faulty C-element	
	a	b	m_i	c	m_o	c	m_o
P_1,P_2,N_5	1	1	0	1	-	1	-
	1	0	0	0	-	1	-
N_1,N_2,P_5	0	1	0	0	-	0	-
	1	1	0	1	-	0	-
N_3	0	0	1	0	-	0	-
	1	0	1	1	-	0	-
P_4	0	1	z	0	0	0	0
	1	1	z	1	1	1	0
P_3,N_4	1	1	z	1	1	1	1
	0	1	z	0	0	1	1

***Table 4.5:** Tests for stuck-open faults of the AND-OR type asymmetric C-element*

Stuck-open faults	Test sequences			Outputs of the fault-free C-element		Outputs of the faulty C-element	
	a	b	m_i	c	m_o	c	m_o
P_1,P_2,N_5	1	0	1	1	-	1	-
	0	0	1	0	-	1	-
P_3	1	1	0	1	-	1	-
	1	0	0	0	-	1	-
N_1,N_2,P_5	0	0	1	0	-	0	-
	1	0	1	1	-	0	-
N_3,P_4	0	0	z	0	0	0	0
	0	1	z	1	1	0	0
N_4	0	1	z	1	1	1	1
	0	0	z	0	0	0	1

input *m* making them controllable. For this purpose an additional weak inverter (transistors P_4 and N_4) is inserted into the original designs of asymmetric C-elements. Its output can be overdriven by applying a logical value to its output *m*. If no logical values are applied to output *m* then it can be used as a test point during the test.

Tests for the transistor stuck-open faults of the asymmetric C-elements (in Figure 4.6) are shown in Tables 4.4 and 4.5. The notation used in these tables has the same meanings as the one used in Table 4.1. It follows from Tables 4.4 and 4.5 that all the stuck-

Table 4.6: *Operation modes of asymmetric C-elements shown in Figure 4.6*

Function of OR-AND type C-element	Function of AND-OR type C-element	Inputs			Outputs of OR-AND type C-element		Outputs of AND-OR type C-element	
		a	b	m_i	c_t	m_o	c_t	m_o
AND	B-repeater	0	0	0	0	-	0	-
		0	1	0	0	-	1	-
		1	0	0	0	-	0	-
		1	1	0	1	-	1	-
A-repeater	OR	0	0	1	0	-	0	-
		0	1	1	0	-	1	-
		1	0	1	1	-	1	-
		1	1	1	1	-	1	-
OR-AND type asymmetric C-element	AND-OR type asymmetric C-element	0	0	z	0	0	0	0
		0	1	z	0	0	1	1
		1	0	z	c_{t-1}	c_{t-1}	c_{t-1}	c_{t-1}
		1	1	z	1	1	1	1

open faults of the asymmetric C-elements are detectable by only five tests which include two sequential tests each.

The designs of asymmetric C-elements testable for transistor stuck-open faults were implemented on a $1\mu m$, double layer metal CMOS process and simulated using *SPICE* analysis in the *Cadence* CAD environment. Simulation results obtained from their extracted layouts can be found in Appendix C.

Driving the feedback transistors allows the sequential functions of the asymmetric C-elements to be changed into combinational ones. Table 4.6 shows the operation modes of the C-elements illustrated in Figure 4.6. If output *m* of the OR-AND type asymmetric C-element is driven by a logical zero or one then its function is transformed into an AND gate or a repeater of its input *a* respectively. When output *m* of the AND-OR type asymmetric C-element is overdriven by a logical one or zero its sequential function is transformed into an OR gate or a repeater of its input *b* respectively. These properties of the asymmetric C-elements (in Figure 4.6) can be used to simplify the testing of asynchronous circuits by reducing the number of their state holding elements.

4.3.2 Testing for stuck-at faults in asymmetric C-elements

As was mentioned previously the stuck-open testability of CMOS circuits does not guarantee the detection of all their line stuck-at faults. In order to make the asymmetric C-elements illustrated in Figure 4.6 testable for line stuck-at faults two additional inputs *tp* and *tn* are required. Figure 4.7 shows a CMOS implementation of the static OR-AND type asymmetric C-element testable for stuck-at faults. In normal operation mode inputs *tp* and *tn* are set to zero and one respectively and the circuit exhibits the same behaviour as the one demonstrated in Figure 4.6a.

Table 4.7 contains tests to detect the single line stuck-at faults whose locations are shown in Figure 4.7. It is presumed that all the inputs and outputs of the asymmetric C-element are controllable and observable during the test. The C-element illustrated in Figure 4.7 is testable for its stuck-open faults since it is fully testable for its stuck-at faults.

The asymmetric C-element testable for single line stuck-at faults was implemented on a $1\mu m$, double layer metal CMOS process and simulated using *SPICE* analysis. Simulation results obtained from its extracted layout can be found in Appendix C.

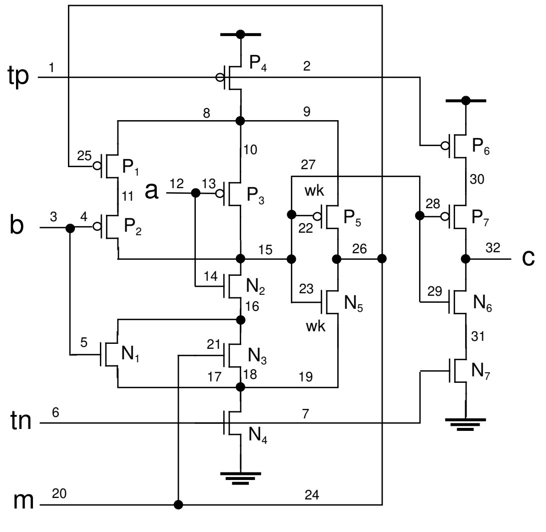


Figure 4.7: *Static OR-AND type asymmetric C-element testable for stuck-at faults*

***Table 4.7:** Tests for stuck-at faults of the asymmetric C-element in Figure 4.7*

No.	Single stuck-0 faults	Single stuck-1 faults	Test sequences			Outputs of the fault-free circuit		Outputs of the faulty circuit	
			a&b	tp&tn	m_i	c	m_o	c	m_o
1	9,20,24,26,30,32	15,27	11	01	z	1	1	0	0
2	15,27	19,20,24,26,31, 32	00	01	z	0	0	1	1
3	7,23,29	1	11	01	z	1	1	1	1
			01	01	z	0	0	1	1
4	6	2,22,28	01	01	z	0	0	0	0
			11	01	z	1	1	0	0
5		3,4,25	11	01	0	1	-	1	-
			10	01	0	0	-	1	-
6		12,13	1x	01	1	1	-	1	-
			0x	01	1	0	-	1	-
7	3,5		01	01	0	0	-	0	-
			11	01	0	1	-	0	-
8	12,14,21		00	01	1	0	-	0	-
			10	01	1	1	-	0	-
9	1		11	11	0	1	-	1	-
			01	11	0	1	-	0	-
10	2		01	01	0	0	-	0	-
			11	11	0	0	-	1	-
11		6	00	01	1	0	-	0	-
			10	00	1	0	-	1	-
12		7	11	01	1	1	-	1	-
			01	00	1	1	-	0	-
13	22,28		00	01	z	0	0	x	x
			00	11	z	0	0	0	0
			00	00	z	0	0	1	1
14		23,29	11	01	z	1	1	x	x
			11	00	z	1	1	1	1
			11	11	z	1	1	0	0
15	25		11	01	1	1	-	1	-
			10	00	1	1	-	1	-
			00	11	1	1	-	0	-
16	4,13		11	01	0	1	-	1	-
			11	00	0	1	-	1	-
			01	11	0	1	-	0	-
17		14	00	01	1	0	-	0	-
			00	11	1	0	-	0	-
			10	00	1	0	-	1	-

***Table 4.7:** Tests for stuck-at faults of the asymmetric C-element in Figure 4.7*

No.	Single stuck-0 faults	Single stuck-1 faults	Test sequences			Outputs of the fault-free circuit		Outputs of the faulty circuit	
			a&b	tp&tn	m_i	c	m_o	c	m_o
18		5,21	00	01	0	0	-	0	-
			10	11	0	0	-	0	-
			11	00	0	0	-	1	-
19		8,10,11	11	01	0	1	-	1	-
			00	11	0	1	-	0	-
20		9,30	01	01	z	0	0	0	0
			11	11	z	0	0	1	1
21	8,11		01	01	0	0	-	0	-
			10	01	0	0	-	1	-
22	10		01	01	1	0	-	1	-
23	19,31		11	01	z	1	1	1	1
			01	00	z	1	1	0	0
24	16,17,18		01	01	1	0	-	0	-
			11	00	1	0	-	1	-
25		17	01	01	z	0	0	0	0
			11	01	z	1	1	0	0
26		16,18	10	01	1	1	-	0	-

4.4 Scan testing of C-elements

Scan testing has already become a standard methodology for testing VLSI circuits [Russ89]. Scan testing presumes that the circuit is set to scan test mode where all its state holding elements are connected together forming a single scan chain (see Appendix B). As a consequence, the states of all memory elements are controllable and observable.

The state holding elements of a scan testable circuit must operate at least in two modes: normal and scan test modes. In normal operation mode, the circuit performs according to its specification. During the scan test, the test patterns are loaded into the state holding elements and the test results are shifted out of the circuit. Figure 4.8 illustrates a CMOS implementation of the pseudo-static symmetric C-element with scan features. It contains two additional control inputs: clock (*Clk*) and scan test (*T*) signals. Inputs *Sin* and *Sout* are used to scan the test pattern in and scan the state bit out of the C-element.

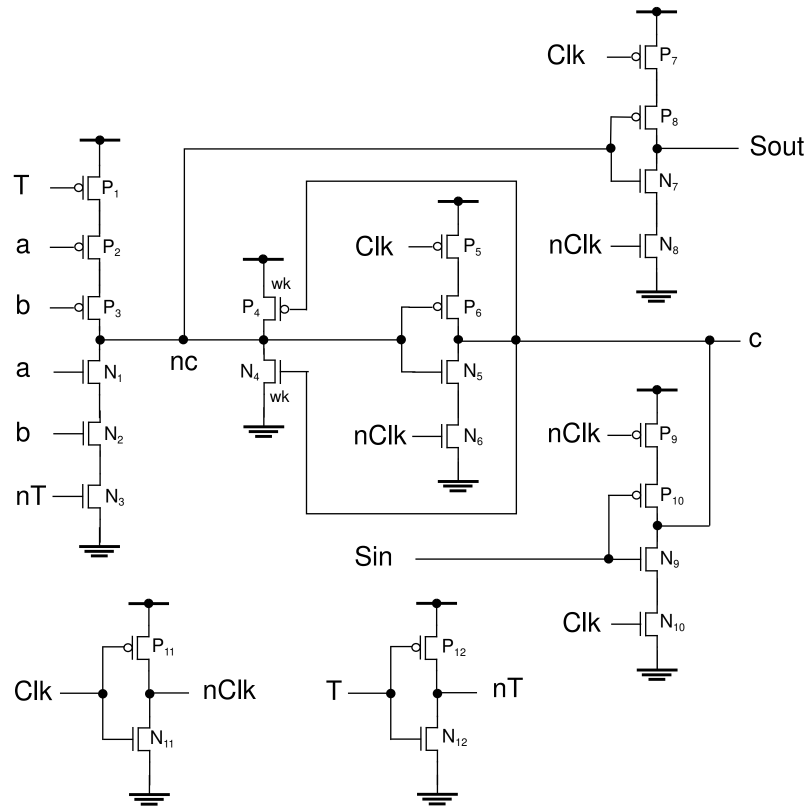


Figure 4.8: *Pseudo-static symmetric C-element with scan features*

Output *Sout* of each scan testable C-element (or any other scan testable memory block) is connected to input Sin of its successor forming the scan chain.

In normal operation mode, when *T*=0 and *Clk*=0 the C-element performs as the pseudo-static C-element depicted in Figure 4.1d. In scan mode, the input transistor stack is disabled by input *T* set to high. Clock signals are generated on input *Clk* to shift the test pattern from input *Sin* into the C-element. When signal *Clk* goes high the output transistor stack of the C-element is disabled and nodes *c* and *nc* are controlled from input *Sin*. Once the clock signal is low the negated bit loaded from input *Sin* is stored in the C-element and is passed to its output *Sout*. Clock signals generated on input *Clk* are used to shift the state bit of the C-element through the scan path to the test circuitry. When *Clk*=1 output *Sout* keeps its current logical value creating a dynamic memory and sup-

Table 4.8: *Tests for stuck-open faults of the C-element in Figure 4.8*

Single stuck-open faults	Test sequences				Outputs of the fault-free C-element		Outputs of the faulty C-element	
	a&b	T	Clk	Sin	c	Sout	c	Sout
P_1-P_3,N_5,N_6	11	0	0	x	1	-	1	-
	00	0	0	x	0	-	1	-
P_5,P_6,P_{12},N_1-N_3	00	0	0	x	0	-	0	-
	11	0	0	x	1	-	0	-
P_7-P_{10},N_4,N_{11}	xx	1	1	1	-	x	-	x
	xx	1	0	1	-	0	-	0
	xx	1	1	0	-	1	-	0
	xx	1	0	0	-	1	-	0
P_4,P_{11},N_7-N_{10}	xx	1	1	0	-	x	-	x
	xx	1	0	0	-	1	-	1
	xx	1	1	1	-	0	-	1
	xx	1	0	1	-	0	-	1
N_{12}	00	0	0	x	0	0	0	0
	11	1	0	x	0	0	1	1

plying input *Sin* of the following memory element. Clock signals must be kept high for enough time to guarantee the proper transmission of logical voltage levels.

An analysis of the symmetric C-element shown in Figure 4.8 reveals that it is testable for single transistor stuck-open faults. Table 4.8 contains tests for detecting the stuck-open faults of the C-element. Symbol ‘x’ denotes a ‘don’t care’ signal. A hyphen means that the appropriate output is not used to observe the test results. Note that the fundamental problem of testing stuck-open faults in the weak feedback inverter of the pseudo-static C-element no longer exists since the weak transistors of the scan testable C-element participate in the scanning of the test data.

Consider an implementation of the scan testable CALL element in order to demonstrate how the C-element shown in Figure 4.8 can be used to build more complex scan testable asynchronous blocks. As was mentioned in chapter 1 the CALL element is an event driven logic block. It remembers which of its inputs received an event first (*R1* or *R2*) and acknowledges the completion of the called procedure by an appropriate event on the matching output (*D1* or *D2*). The CALL element with scan features shown in Figure 4.9 performs using the two-phase signalling protocol where each signal transition denotes an event. All the inputs and outputs are initialized to zero. *T*=0 and *Clk*=0 in normal operation mode. When a request event occurs on input *Ri* it primes C-element *Ci* and passes through the XOR gate producing a request event on its output *R*. Once the required procedure has completed an appropriate acknowledge event is generated on input *D*. As a result, C-element *Ci* changes state and an acknowledge event is passed to output *Di* (i=1,2). The performance of the CALL element is identical for falling request events.

If the CALL element shown in Figure 4.9 is incorporated into an asynchronous VLSI circuit its internal states can be controllable and observable through the scan path. A test bit sent to input *Sin* of the C-element is negated on its outputs *Sout* and *c* (see Figure 4.8). The CALL element is tested by setting test control signal *T* to one. For instance, the clocked sequence 01 must be applied to input *Sin* of the CALL element in order to set its C-elements to one. The CALL element can perform its specified function when

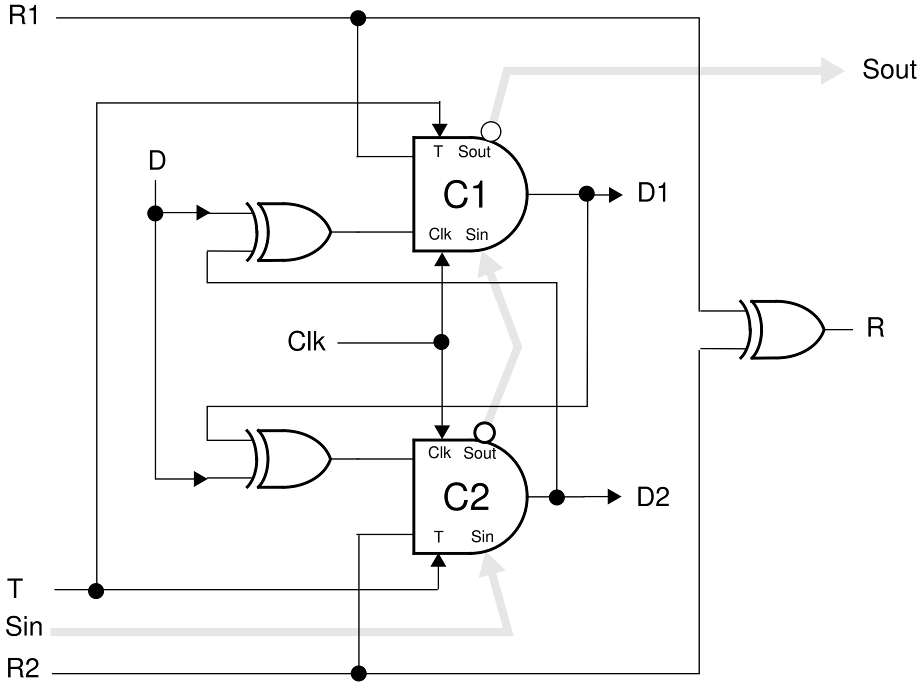


Figure 4.9: *CALL element with scan features*

signals *T* and *Clk* are returned to zero. The state bits of the C-elements are shifted out of the CALL element and compared with known responses when its input *T* is set one and clocks are produced on its input *Clk*.

4.5 Summary

The testable structures of the symmetric and asymmetric C-elements presented in this chapter require different overheads depending on the fabrication faults to be detected. The designs of testable C-elements have been implemented on a $1\mu m$ CMOS process and their extracted layouts have been investigated using SPICE analyses. Table 4.9 contains a summary of cost comparisons of the CMOS C-elements and their testability.

The largest number of transistors is required to implement the scan testable C-element. This is because scanning the data through the C-element can be implemented only in a master-slave manner which requires at least two memory elements.

The implementation of the symmetric C-element shown in Figure 4.2 has a layout overhead of just 17% with one extra control input and guarantees the detection of all its stuck-open faults. The sequential function of such a C-element can be changed into a combinational one (AND or OR) which simplifies the testing of other components incorporated in the asynchronous circuit.

Table 4.9: *Summary of costs of the testable C-elements*

Design	No. of transistors	No. of extra inputs/outputs	Transistor overhead	Layout overhead	Output nodal capacitance $\times 10^{-14} F$	Testability
Figure 4.2	14	1	17%	17%	2.07	SO
Figure 4.3	20	3	67%	45%	3.21	SA&SO
Figure 4.6a	10	1	25%	19%	2.45	SO
Figure 4.6b	10	1	25%	32%	2.55	SO
Figure 4.7	14	3	75%	41%	2.18	SA&SO
Figure 4.8	24	4	200%	115%	11.22	SO

Compared to the design of the symmetric C-element shown in Figure 2.5 the implementation of the C-element illustrated in Figure 4.2:

- requires around half as many transistors (14 versus 26);
- uses just one test input to change the operation mode of the C-element;
- guarantees the detection of all the stuck-open faults in its CMOS design.

The asymmetric testable C-elements illustrated in Figures 4.6a and 4.6b guarantee the detection of all their transistor stuck-open faults and require 19% and 32% layout overheads respectively.

Chapter 5 : Scan Testing of Micropipelines

The scan test has become a standard DFT methodology for testing synchronous digital circuits (see Appendix B). The adoption of well-developed test techniques to the testing of asynchronous circuits is important because it avoids the costs of developing expensive new test equipment. An asynchronous scan test approach to designing testable two-phase and four-phase micropipeline structures is considered in this chapter.

5.1 Micropipeline latch control

The design of a micropipeline with processing was considered in chapter 1. The micropipeline illustrated in Figure 1.5 can operate using either a two-phase or a four-phase signalling protocol. Two-phase and four-phase micropipelines use different latch control circuits to ensure the correct latching mechanism.

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 5.1 shows an implementation of the normally closed latch structure which uses two-phase signalling. In the initial state the outputs of

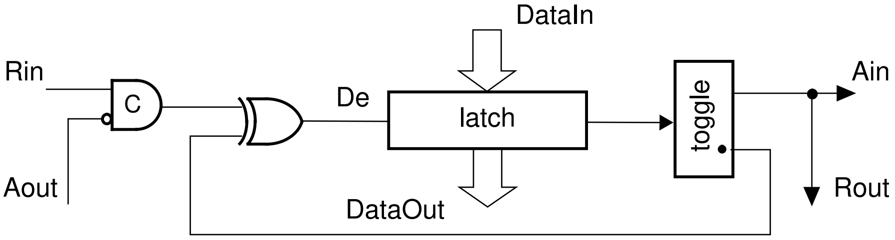


Figure 5.1: *Two-phase control for a normally closed latch*

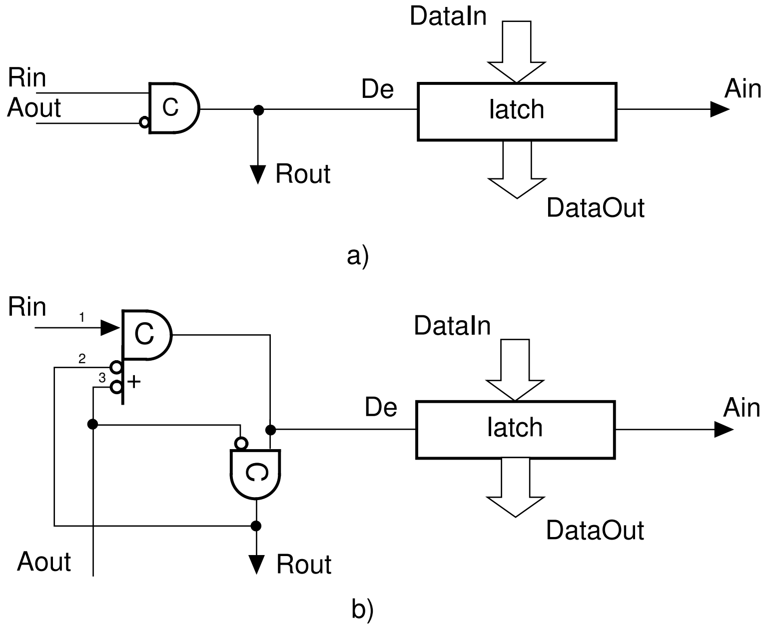


Figure 5.2: *a) Simple and b) semi-decoupled four-phase control for a normally closed latch*

the toggle element and the C-element are reset. When the data is stable on the inputs of the latch a request signal is sent to input *Rin* setting the data enable input (*De*) of the latch to high. As a result, the latch becomes transparent storing the data into its memory. The toggle element steers a rising event to its dotted output causing the data to be latched in the latch memory. The second rising event is steered by the toggle element to its blank output producing a request signal (*Rout*) for the next stage register and an acknowledge signal (*Ain*) for the previous stage of the micropipeline. A rising signal arriving at input *Aout* primes the C-element and the latch is ready to repeat the sequence described above when a falling event is generated on its *Rin* input.

Designs of the normally closed latch with simple and ‘semi-decoupled’ (see below) four-phase control are shown in Figure 5.2a and Figure 5.2b respectively [EdTR95]. With the simple latch control the latch can be closed only when *Aout* goes high, i.e., when the next stage of the micropipeline is opened. As a result, the use of the simple latch control circuit is not cost-effective in four-phase micropipelines since at most alternate stages can be occupied at any time [Furb96]. In order to increase the decoupling between the inputs and outputs of the latch to allow the latch to be closed before

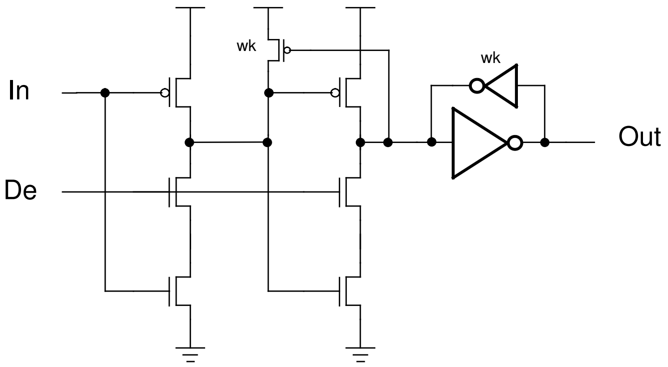


Figure 5.3: *Single-phase static latch*

the output handshake protocol has completed, the semi-decoupled control circuit can be used (see Figure 5.2b). Initially both C-elements are reset. When the input data is stable a rising request signal is generated on the *Rin* input. As a result, the latch is opened and passes the input data to its outputs. A rising acknowledge signal is produced on the *Ain* output. *Rout* goes high preventing the output of the asymmetric C-element from going high. If *Rin* is reset the latch is closed and the input data is latched in its memory. As a consequence, *Ain* goes low and the input data can be changed. The output of the symmetric C-element (*Rout*) is reset when *Aout* is high (which is when the next stage is opened). The next control cycle can be repeated when *Aout* is low enabling the output of the asymmetric C-element to be set to high. The use of the semi-decoupled control circuit shown in Figure 5.2b allows the micropipeline to fill all its stages increasing its performance.

Figure 5.3 shows 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, Weste93].

5.2 Fault model

Stuck-at faults in the control part, combinational logic blocks and latches of the micropipeline shown in Figure 1.5 are considered.

Faults in the latch control

As was shown in chapter 2, stuck-at faults in the control part of the two-phase micropipeline can be detected during its normal operation mode since they cause the micropipeline to halt.

Stuck-at faults in the simple four-phase latch control circuit illustrated in Figure 5.2a cause the micropipeline to halt. For instance, a stuck-at fault on any input of the C-element is equivalent to the corresponding stuck-at fault on its output. As a result, the handshake protocol is violated causing the micropipeline to deadlock. The same fault effect is caused by a stuck-at fault on any other wires in the control circuit.

Most stuck-at faults in the semi-decoupled control circuit shown in Figure 5.2b cause the micropipeline to halt. For example, in the presence of fault 2-SA1 on input 2 of the asymmetric C-element its output cannot be set to high. As a result, the *Ain* output of the control circuit is reset permanently keeping the latch closed. Faults 2-SA0 and 3-SA0 cannot be detected easily since they do not violate the four-phase handshake protocol but cause premature rising events on the output of the asymmetric C-element. Special care must be taken to detect these faults.

Faults in the combinational logic blocks

Faults in the combinational blocks of the micropipeline illustrated in Figure 1.5 can be detected by applying test vectors to their inputs and observing the test results latched in the corresponding stage registers.

Faults in the stage registers

Faults in the stage registers can put the faulty register latch permanently in capture (a stuck-at-0 fault on the data enable input) or pass (a stuck-at-1 fault on the data enable input) mode (see Figures 5.2 and 5.3). Stuck-at faults on the inputs or outputs of the stage register are equivalent to stuck-at faults in the corresponding combinational circuit.

5.3 Scan test design

5.3.1 Scan latch implementation

Figure 5.4 shows a CMOS implementation of a scan latch structure which contains two latches (L_1 and L_2) and a multiplexer.

In normal operation mode (the test control signal *Tst* is low) the tristate inverter of L_2 is off since the shift clock signal *Sc* is held at zero (*nSc*=1). When the data enable signal (*De*) is high the input data (*Din*) passes to the output *Dout* and is latched by L_2 when *De* is low.

In scan mode (*Tst*=0, *nTst*=1) the enable signal *De* is low so that the tristate buffer of L_2 is closed. When the clock signal *Sc* is high (*nSc*=0) the scan data from the scan-in input

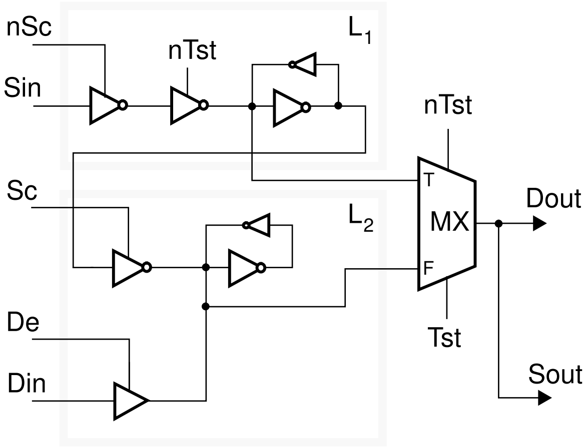


Figure 5.4: *CMOS implementation of the scan latch*

Table 5.1: *Data path delays for the basic and scan latches*

Latch	Path	Delay
Basic	*Din* to *Dout*	2.3ns
Scan	*Din* to *Dout*	4.2ns
	Sin to *Dout*	5.7ns

(*Sin*) is latched by the latch L_1 and passes to the tristate inverter of L_2. While *Sc*=1, L_2 is opened and the shift data is sent to the scan-out output (*Sout*) of the scan latch. When *Sc*=0 (*nSc*=1) the scan data bit is latched by L_2 and the latch L_1 is opened. This procedure is similar to that used for storing the data in a master-slave flip-flop.

In test mode (*Tst*=1, *nTst*=0, *Sc*=0, *nSc*=1) the scan latch performs as in normal operation mode. The only difference is that the response bit from the combinational circuit is stored in L_2 whereas the test bit is held unchanged in L_1 and stimulates the appropriate input of the processing logic of the next stage. Note that, during scan mode when the last test bit is shifted in the scan latch, the Boolean signal *Tst* (*nTst*) must be set to one (zero) before the signal *Sc* (*nSc*) goes down (high) in order to preserve the state of L_1.

The basic and scan versions of the latch structure have been implemented in CMOS technology on a $1\mu m$ double layer metal process and simulated using SPICE (see Appendix C). The basic latch cell used is similar to a single-phase static CMOS latch which requires 11 transistors (see Figure 5.3). 37 transistors were used for the implementation of the scan latch. As a consequence, the transistor-count redundancy of the scan latch is 236% and the area overhead is 258%. This scan latch requires 12% fewer transistors than the one proposed by Khoche and Brunvand [Khoc94]. Table 5.1 shows the simulated delays through data paths of the two latch structures. The simulations have been performed on extracted layouts from the latch designs for worst case conditions: V_{dd}=4.6V, slow-slow process corner, temperature 100^0C.

5.3.2 Scan register design

A two-bit scan register design for a testable micropipeline is shown in Figure 5.5. Compared with the basic stage register it contains five additional wires: test control (*Tst*), scan-in (*Sin*), scan-out (*Sout*) and shift clock (*Sc*).

Normal operation mode (*Tst*=0, *Sc*=0). In the initial state the register latches L_2 are closed and the outputs of the latch control circuit are set to zero (reset control lines are omitted in Figure 5.5). When a request signal is received on the input *Rin* the data enable signal *De* is set to high by the latch control and the latches L_2 are opened. Since the data enable signal can drive a large number of latches (more than two as shown in Figure 5.5) output *De* of the latch control is connected to buffer *B*. A request signal is generated on output *Rout* of the latch control block. The data is transmitted from the inputs *Din* to the outputs *Dout* of the register. Depending on a particular signalling protocol used by the latch control the latches L_2 are closed (*De*=0) when the handshake between one or two neighbouring stage registers has completed.

Scan mode. While *Tst*=0 and *De*=0 the register can be used to scan the data into the latches from its input *Sin*. Simultaneously, the scan data comes to the output *Sout* supplying another scan register. The scan procedure is controlled by clock signals applied to the input *Sc*.

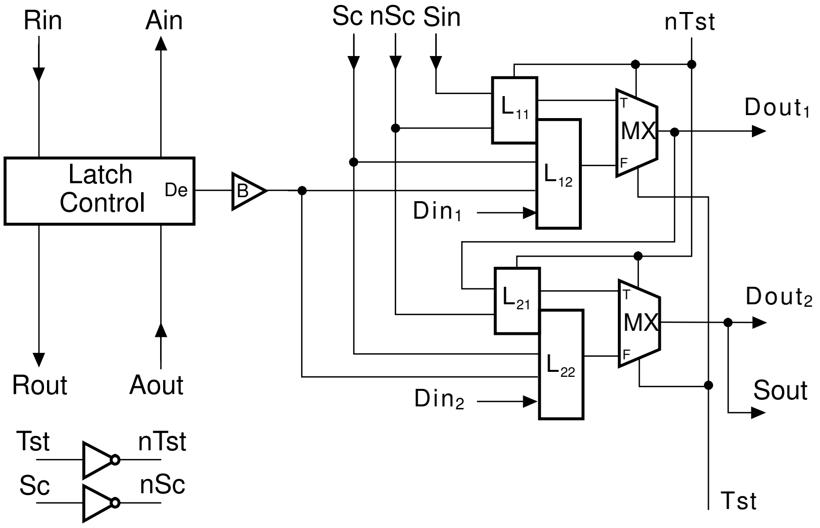


Figure 5.5: *Scan register*

Test mode. During the test (*Tst*=1, *Sc*=0) the test vectors are stored in the first latches L_1. The outputs of these latches are connected through the multiplexers to the outputs of the stage register. After receiving a request signal on the line *Rin* the data is stored into the latches L_2 of the register (see Figure 5.5). The test vectors and the test results are saved in different latches because the data flows through the micropipeline from left to right while the test vectors must be preserved during the test.

5.4 Scan test control

A testable micropipeline design is shown in Figure 5.6. It comprises a micropipeline and the scan test control logic (STCL) unit. The stage registers of this micropipeline are built from scan latches. The scan test control block is used to make an asynchronous test interface for the micropipeline. It also generates shift clock signals *Sc* for a unified shift register. The scan test control can follow either a two-phase or four-phase signalling protocol depending on its structure. The scan test control can either be a central control block or can be incorporated inside some of the micropipeline registers. Similar scan test control units can be used in different parts of the chip to arrange an asynchronous scan test control interface between different asynchronous blocks.

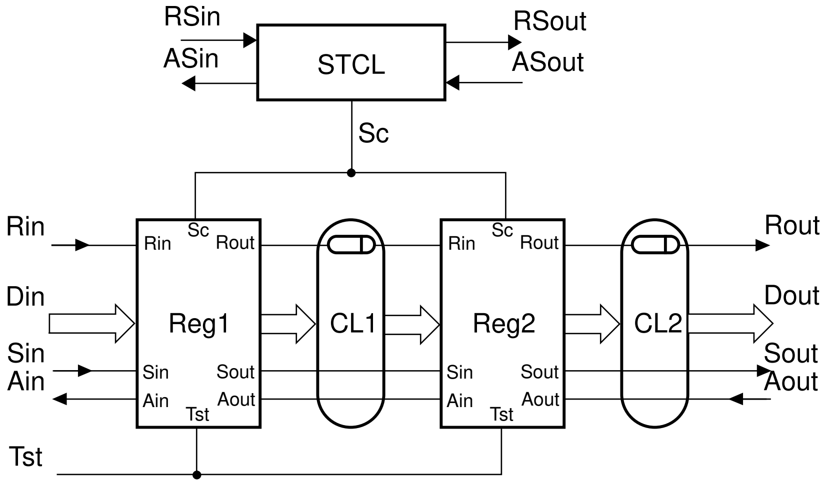


Figure 5.6: *A micropipeline with scan features*

5.4.1 Scan test control for two-phase transition signalling

An example of the scan test control block for two-phase transition signalling is shown in Figure 5.7. The scan test control block generates the control signals in a manner similar to that of the control circuitry of the latch illustrated in Figure 5.1. The presence of the C-element ensures the delay insensitivity of the scan test control block.

Some calculations of the typical delays in the scan test control block have been carried out using SPICE and are shown in Table 5.2.

Table 5.2: *Two-phase scan test control delays*

Path	Delay
RSin to *ASin*	7.0ns
RSin to *RSout*	12.8ns
ASout to *C-element primed*	1.1ns
Cycle time	20.9ns

5.4.2 Scan test control for four-phase signalling

The implementation of the scan test control block for a four-phase communication protocol is simpler than that of the scan test control block for two-phase signalling (see Figure 5.8).

Initially, the C-element is set to zero (*nASin*=*nASout*=1). When a rising shift request signal *RSin* arrives, the C-element changes its state to one. As a result, a falling event is

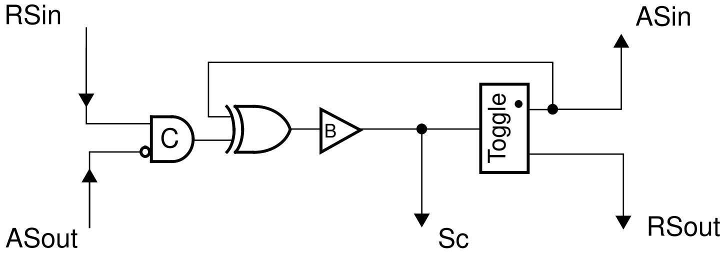


Figure 5.7: *Scan test control logic for a two-phase micropipeline*

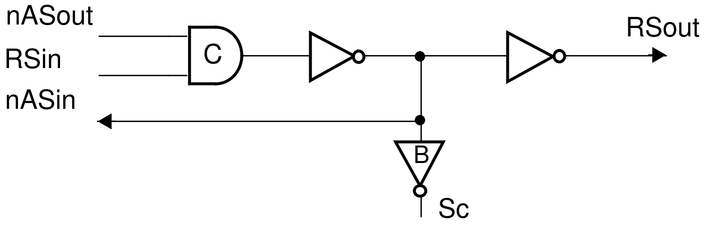


Figure 5.8: *Scan test control logic for a four-phase micropipeline*

produced on the negated acknowledge line *nASin*, the shift clock signal *Sc* goes high and a rising event is generated on the request output *RSout*. If *nASout*=*RSin*=0 the C-element is set to zero. Thus, the signal *Sc* is reset, rising and falling events are produced on the control lines *nASin* and *RSout* respectively.

The delays of the control signals in the scan test control block are presented in Table 5.3. The results show that the use of the four-phase scan test control block improves the performance of the shift operation compared to the two-phase scan test control block.

To reduce the number of external pins for the implementation of the testable micropipeline the pairs of signals such as *Rin* and *RSin*, *Ain* and *ASin*, *Rout* and *RSout*, *Aout* and *ASout* can be combined using multiplexers. These multiplexers are controlled by a Boolean signal switching between scan and normal operation mode. Note that, for two-phase signalling, some of the multiplexers must contain state holding elements.

Table 5.3: *Four-phase scan test control delays*

Path	Delay
RSin↑ to *RSout*↑	6.9ns
RSin↑ to *nASin*↓	2.5ns
nASout↓ to *nASin*↑	2.5ns
nASout↑ to *RSout*↑	6.9ns
Cycle time	18.8ns

5.5 Test strategy

The test strategy for detecting stuck-at faults in micropipelines is very similar to that used in scan testing synchronous circuits. If *Sc*=0 the micropipeline shown in Figure 5.6 can perform in normal operation mode (*Tst*=0) or in test mode (*Tst*=1).

In scan mode (*Tst*=0, *De*=0), the test patterns are loaded into the stage registers which are configured as a unified scan register. The scan path is created by connecting the inputs *Sin* in series to the outputs *Sout* of all the stage registers. Clock signals *Sc* for controlling the shift operation are generated internally by the scan test control.

When the test patterns are loaded into the latches the micropipeline is set to test mode (*Tst*=1). A request signal is produced on the line *Rin* of the micropipeline. The responses from each processing block are stored in the registers.

When *Tst*=0 the contents of the latches are shifted to the output *Sout* of the last stage register. The test results are compared with known good ones. Whilst shifting out the test results to the output *Sout* a new test pattern is loaded from the input *Sin*. The test procedure is repeated. Thus, the complexity of testing the micropipeline is reduced to the testing of its processing logic which comprises mostly combinational circuits.

Testing for faults in the scan test control

The scan test control unit of the testable micropipeline is an additional control block which is not used in normal operation mode. Nevertheless, it must be fault free as it controls the scan path of the micropipeline. A stuck-at fault on any of the lines in the scan test control block prevents the generation of the control signals on its outputs. This is because the scan test control is a fully delay-insensitive asynchronous circuit where every control signal handshakes with others. Such circuits are fully testable for stuck-at faults [Dav90, Haz92].

Testing for faults in the control logic

As was mentioned earlier, stuck-at faults on the control lines of the two-phase micropipeline and the four-phase micropipeline with simple latch control can be detected easily since they cause the micropipeline to halt. This happens because a micropipeline is an event-driven asynchronous circuit [Suth89]. Such stuck-at faults can be identified either in normal operation mode or during the test. The detection of faults in the semi-decoupled latch control circuit of the four-phase micropipeline requires a special test mode. The testability issues of the semi-decoupled latch control circuit are considered later.

Testing for faults in the processing logic

It is assumed that all the processing blocks between the stages of the micropipeline are combinational circuits. The internal inputs of each combinational circuit are controllable and its outputs are observable through the scan path of the micropipeline. Tests for detecting stuck-at faults in all the processing blocks can be derived using well known test generation algorithms such as the D-algorithm, PODEM, FAN and others (see Appendix A).

A test scenario for detecting stuck-at faults in the processing logic of the two-phase micropipeline, which is similar to that of the four-phase micropipeline, can be described by the following sequence of steps:

1. Reset the micropipeline. All the state holding elements (except register latches) are reset.

2. Set the test control signal to zero (*Tst*=0). All the register latches are connected in a unified scan register. When a new test bit on the *Sin* input of the micropipeline is stable a request is generated on the *RSin* input of the scan test control block. Once acknowledged another test bit accompanied by the appropriate request signal is produced on the *Sin* input of the micropipeline until the whole scan path is full. Every time when the scan data is ready the scan test control generates scan clocks on its output *Sc*.

3. Set the micropipeline in test mode (*Tst*=1). Generate one request signal on the *Rin* input. This request propagates through all the registers of the micropipeline. As a consequence, the responses from the processing blocks are saved in the corresponding second latches L_2 in the stage registers. The outputs of the first latches L_1 are held unchanged which allows them to control the inputs of the appropriate combinational processing blocks at known values.

4. *Tst*=0 and repeat *Step 2* simultaneously unloading the latch contents out of the scan register to the *Sout* output and shifting in a new test vector from the *Sin* input of the micropipeline.

5. The test results are compared with the good ones. If the current test is successful repeat *Step 3*. If not, stop the test procedure since the micropipeline is faulty.

6. If, after the required number of tests, no faults have been detected the micropipeline can be tested for stuck-at faults in the register latches.

Testing for faults in the latches

Two types of stuck-at faults are considered for the register latches: stuck-at-capture and stuck-at-pass faults.

Stuck-at-capture (stuck-at-pass) faults of the scan latch (see Figure 5.4) can be caused by stuck-at faults on the control lines of the tristate buffers and inverters which disable (enable) them permanently. Most of these faults can be detected by shifting an alternating 0-1 test through the latches unified in one scan register.

A stuck-at-1 fault on the input *nTst* of the latch L_1 can be identified during test mode when the faulty scan latch and its predecessor are set to different states. In this case the state of the faulty latch L_1 will be changed. Stuck-at-0 and stuck-at-1 faults on the line *De* of latch L_2 are detected by driving the *Din* input with a different logic value to its current state during test mode and scan mode respectively.

Stuck-at faults on the data lines of the scan latch shown in Figure 5.4 are detected during test and scan mode.

Testing delay faults

There is another class of faults in micropipelines which can be detected using the proposed scan test technique. These are delay faults in combinational circuits between the stage registers of the micropipeline. The output data of each combinational logic block is latched after a certain delay when the data has arrived at its inputs. A delay fault in this combinational block will extend path delays. In the presence of such a fault the bundled data interface of the corresponding micropipeline stage will be violated, i.e. the outputs of the combinational logic will be latched before the output signals in the bundle are stable.

The algorithm used to detect delay faults in the processing logic of the micropipeline is similar to that exploited in delay testing of synchronous circuits which has been adapted by Khoche and Brunvand [Khoc94].

Basically, the pair of test vectors (v_1 and v_2) must be applied to the inputs of the combinational circuit to detect its path delay faults. According to this test approach three stage registers (R_{i-1}, R_i and R_{i+1}) are used to detect delay faults in the combinational logic F_i. The tests v_3 and v_1 are stored in the registers R_{i-1} and R_i respectively. The results of the test are saved in the register R_{i+1}. When the test patterns are loaded into the stage registers the combinational circuit is settled (test v_1). The delay fault is tested by applying a request signal to the input *Rin* of the micropipeline set in normal operation mode. This causes the application of the test v_2 to the inputs of the logic F_i ($v_2 = F_{i-1}(v_3)$). The data path of the circuit under test is activated. If there is a delay fault in this path it will cause a delayed response by the combinational circuit whereas the responses are latched after a fixed time determined by the corresponding delay.

5.6 Scan testing of asynchronous sequential circuits

The scan test technique described above can be used to test asynchronous sequential circuits built using the micropipeline design style.

5.6.1 Sequential circuits based on the micropipeline approach

Figure 5.9 illustrates the general structure of a two-phase sequential circuit. This structure contains the combinational logic block (*CLB*) which performs the basic logic operations, and two registers (*Reg1* and *Reg2*) in the feedback loop which store the state of the sequential circuit. The sequential circuit works as a micropipeline. In the initial state, all the latches of *Reg1* are set to their initial states and both the C-elements are set to zero. The input data is generated on the primary inputs (*PI*) of the circuit by the sender which sends a rising request signal (*Rin*) to the sequential circuit. The request signal is delayed by the delay element for long enough for the output data to stabilize on the primary (*PO*) and internal (*SO*) outputs of the combinational circuit. As a result, a rising request signal (*Rout*) is produced for the receiver by the sequential circuit. After receiving an acknowledge signal (*Aout*) and storing a new state in *Reg2* the circuit generates an acknowledge signal (*Ain*) for the sender simultaneously causing the copying of the contents of *Reg2* into *Reg1*. When a new falling request signal is sent by the sender the procedure of processing the data is repeated.

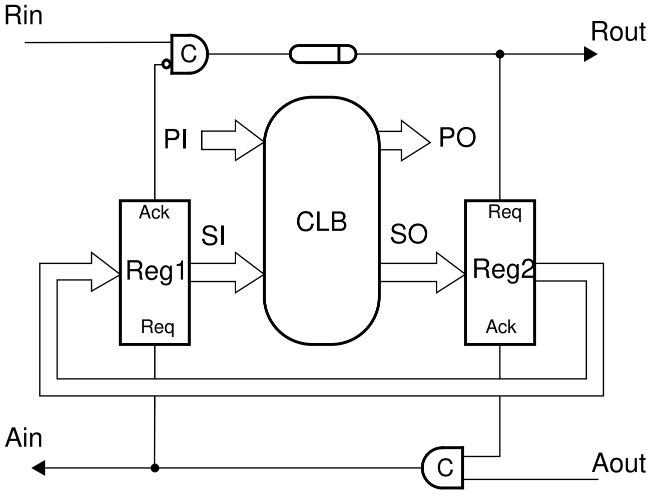


Figure 5.9: *Two-phase sequential circuit*

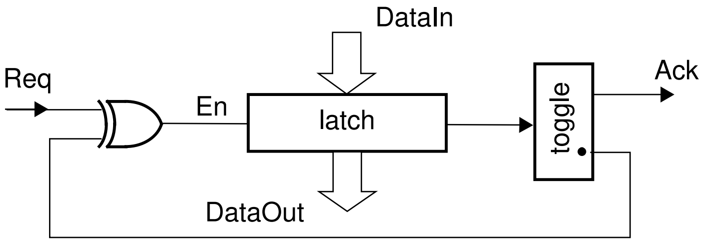


***Figure 5.10:** Latch control of the sequential circuit*

The latch control for the two-phase sequential circuit is shown in Figure 5.10. The functioning of this control circuit is similar to the one illustrated in Figure 5.1. There is no need for the symmetric C-element since the request signal (*Req*) is controlled externally by the C-elements of the micropipeline.

Figure 5.11 illustrates an implementation of the sequential circuit which follows a four-phase signalling protocol. Initially all C-elements are reset. The registers *Reg1* and *Reg2* are closed. The latches of register *Reg1* are set to their initial states. When the input data is stable a rising request signal is produced on input *Rin*. The output of the C-element *C1* is set to high. This rising event is delayed for long enough for the output data to stabilize on the PO and SO outputs of the combinational circuit. As a result, the output of the asymmetric C-element *C2* is set to high opening the latches of *Reg2*. When the latches are transparent register *Reg2* generates a rising acknowledge signal which is passed to the *Rout* output of the circuit. When the output data has been read by the environment a

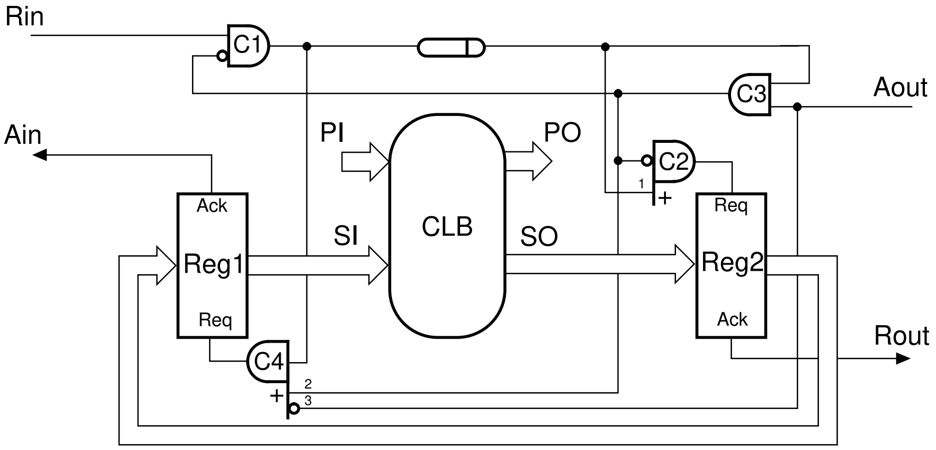


***Figure 5.11:** Four-phase asynchronous sequential circuit with normally closed registers Reg1 and Reg2*

rising event is passed to the *Aout* input of the sequential circuit setting the output of the C-element *C3* to high. As a consequence, the output of *C2* is reset, the latches of *Reg2* are closed and *Rout* goes low. When the acknowledge signal *Aout* is returned to zero the output of *C4* is set to high making the latches of *Reg1* transparent. When the data is stored in *Reg1* the acknowledge signal *Ain* goes high. Afterwards, the *Rin* signal is returned to zero resetting the output of *C1* primed by a high signal from the output of *C3*. The C-element *C4* is reset closing the latches of *Reg1* and resetting the *Ain* output. As a result, the sequential circuit has moved into a new state and it is ready to accept new input data. The output of *C1* can be set to high by a rising request signal arriving at the *Rin* input when the output of *C3* is reset. The structure of registers *Reg1* and *Reg2* is similar to that shown in Figure 1.8d.

5.6.2 Scan test design

The design of the testable two-phase sequential circuit is illustrated in Figure 5.12. It comprises the sequential circuit with the scan test control logic (STCL) block which provides an asynchronous interface in scan mode (see Figures 5.7 and 5.8). The stage registers of such a sequential circuit are built using scan latches shown in Figure 5.4. The structure of the testable four-phase sequential circuit is similar to that shown in Figure 5.12. The only difference between them is in the latch control design.

5.6.3 Scan test scenario

When *Sc*=0, the sequential circuit illustrated in Figure 5.12 can act either in normal operation mode (*Tst*=0) or test mode (*Tst*=1).

If *Tst*=0 and Sc=0 the sequential circuit is set to scan mode to load the test patterns into the latches of the stage registers. The scan path is created by connecting the output *Sout* of *Reg1* to the input *Sin* of *Reg2* (see Figure 5.10). Clocks *Sc* for controlling the shift operation are generated internally by the scan test control.

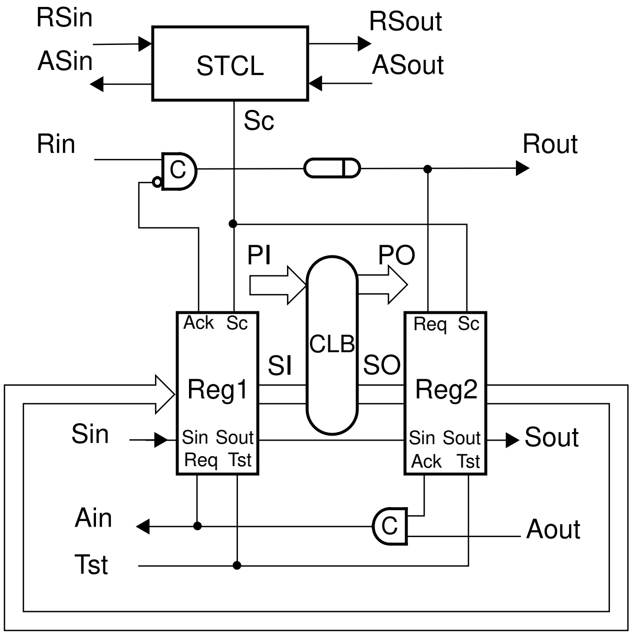


Figure 5.12: Two-phase sequential circuit with scan features

In test mode, a request signal is produced on the line *Rin* of the sequential circuit. The results of testing the combinational logic block are stored in the stage registers.

The contents of the latches are shifted out to the output *Sout* of *Reg2* in scan mode. The test results are compared with known good ones. Whilst unloading the test results a new test pattern is loaded from the input *Sin* of the sequential circuit and the test procedure can be repeated. The complexity of the testing of the sequential circuit is reduced to the testing of its combinational block.

Stuck-at faults on any of the lines in the scan test control block are tested during scan mode. As was mentioned earlier, stuck-at faults on the control lines of the sequential circuit shown in Figure 5.12 cause the sequential circuit to halt. Faults in the latches are detected in the same manner described in section 5.5.

Testing for faults in the combinational logic

The internal inputs (*SI*) of the combinational circuit are controllable and its internal outputs (*SO*) are observable through the scan path. Tests for detecting stuck-at faults in the combinational logic block can be derived using any known test generation algorithms for combinational circuits.

The test algorithm can be described as the following sequence of actions:

1. The test pattern is loaded into *Reg1* during scan mode.
2. In test mode, when the data is stable on the inputs *PI* of the sequential circuit, a request signal is generated on the input *Rin*. The responses from the outputs *SO* of the combinational circuit are stored in *Reg2*.
3. The test results are analysed on the outputs *PO* of the sequential circuit and an acknowledge signal is produced on its input *Aout*.
4. The sequential circuit is set to scan mode to shift the contents of *Reg2* out to the output *Sout* and to load a new test pattern into *Reg1*.

Testing delay faults

The algorithm used to detect delay faults in the combinational logic of the sequential circuit is similar to that described in section 5.5. In principle, the pair of test vectors $<V_1, V_2>$ must be applied to the inputs of the combinational circuit to detect its delay path fault. The pair $<V_1, V_2>$ is made up of $<v_{1p} \ @ \ v_{1s}, v_{2p} \ @ \ v_{2s}>$, where:

- v_{1p} and v_{2p} are the test vectors applied to the inputs *PI* of the combinational circuit;
- v_{1s} and v_{2s} are the state vectors initially loaded into the state registers (*Reg1* and *Reg2* respectively);
- the symbol @ denotes the concatenation of bit vectors.

The test scenario is the following:

1. In test mode, the test pattern v_{1p} is supplied to the inputs *PI* and a request signal is generated on the input *Rin*. The combinational circuit is settled.

2. After receiving an acknowledge event on the input *Aout* the test v_{2_s} is copied into the latches L_2 of *Reg1*. The test vector v_{2_p} is applied to the inputs *PI* of the sequential circuit.

3. The test control signal *Tst* is set to zero and a new request event is generated on the input *Rin*. As a result, the test vector v_{2_s} is applied to the *SI* inputs of the combinational circuit and the required data path is activated. If there is a delay fault in this path it will cause a delayed response by the combinational circuit whereas the responses are latched after a fixed time determined by the corresponding delay.

5.7 Testing faults in four-phase latch control circuits

This section is devoted to the testing of faults in the latch control circuits of the four-phase micropipeline and sequential circuit designs.

5.7.1 Testing for faults in the semi-decoupled control circuit

As was previously mentioned testing for faults 2-SA0 and 3-SA0 in the latch control circuit shown in Figure 5.2 is not trivial since they cause premature firings on the *Ain* output.

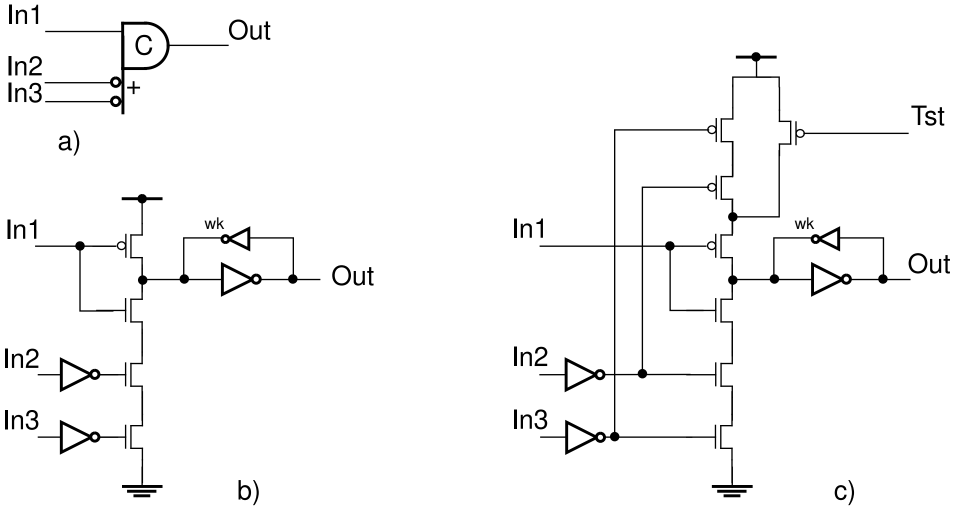


Figure 5.13: *Asymmetric C-element: a) symbol; b) CMOS implementation; c) testable CMOS implementation*

Figures 5.13a and 5.13b show the symbol and a CMOS implementation of the asymmetric C-element used in the design of semi-decoupled latch control. Faults *In2*-SA0 and *In3*-SA0 keep the corresponding n-type transistors opened permanently. These faults can be detected by the pair of tests <011;111> applied sequentially. As a result, the fault-free result is low whereas in the presence of the these faults the *Out* output is set to high. The testing of faults *In2*-SA0 and *In3*-SA0 in the control circuit of the four-phase micropipeline is difficult due to the low controllability of the inputs of the asymmetric C-elements.

A simple solution which makes the inputs of the asymmetric C-element more testable is to convert it into a symmetric C-element. Figure 5.13c illustrates a CMOS design of the circuit which acts as an asymmetric C-element shown in Figure 5.13b and as a 3-input

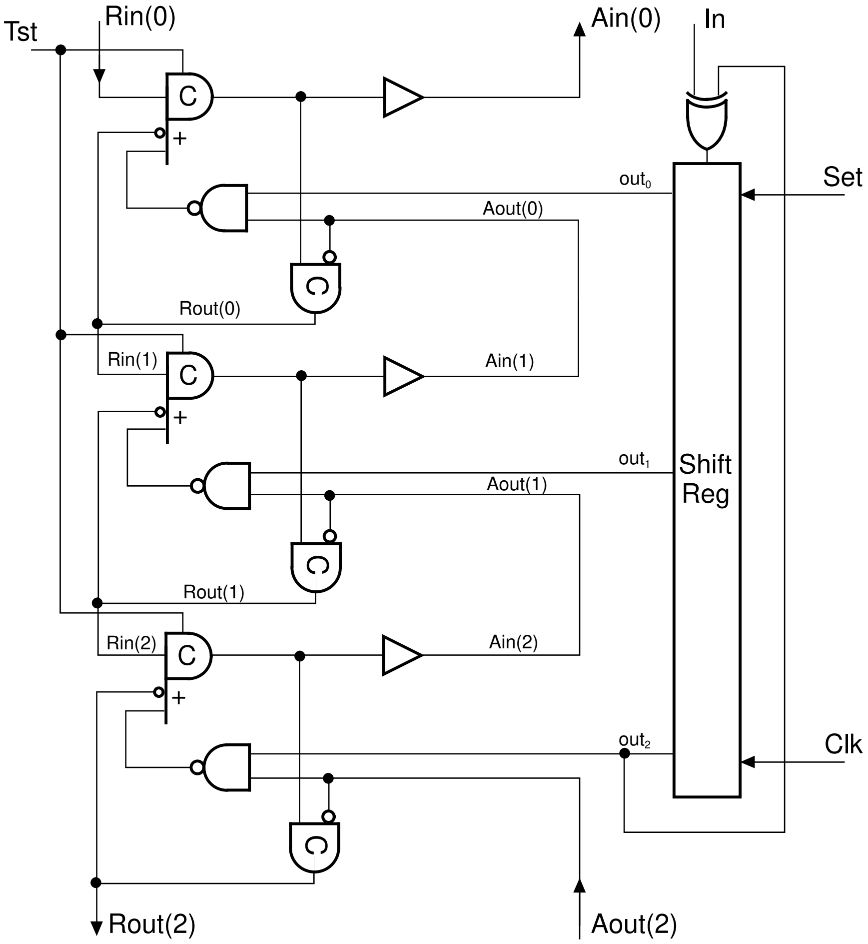


Figure 5.14: *Testable 3-stage four-phase semi-decoupled latch control circuit*

symmetric C-element. When the Boolean input *Tst* is set to high or low the circuit operates as the symmetric or asymmetric C-element respectively. It is easy to show that any stuck-at faults on the inputs of the symmetric C-element is equivalent to the corresponding stuck-at output faults.

Figure 5.14 demonstrates an example of the semi-decoupled control circuit of the three stage four-phase micropipeline. In order to test the control circuit for stuck-at faults all asymmetric C-elements are converted into symmetric ones by setting the Boolean *Tst* to high. The outputs of the 3-bit shift register are set to high. The control circuit is forced by the environment to perform at least one handshake along the input and output channels. As a result, all stuck-at faults on inputs of all the C-elements and all the control lines are detected since they cause the circuit to deadlock. In normal operation mode, the asymmetric C-elements which were converted into symmetric ones are set back to their normal modes by resetting the Boolean *Tst*.

In the circuit shown in Figure 5.14 there is a class of stuck-at faults which must be detected to ensure that the micropipeline performs according to its specification. These are faults on the *Tst* inputs of the testable asymmetric C-elements. Faults *Tst*-SA0 or *Tst*-SA1 set the circuit shown in Figure 5.13c to perform as an asymmetric or symmetric C-elements permanently.

In order to detect faults *Tst*-SA0 and *Tst*-SA1 on the *Tst* inputs of the testable C-elements the following test algorithm can be used:

1. i=0, where i is an index for the micropipeline stage to be tested.
2. Tst=1 for testing faults Tst_i-SA0. Tst=0 for testing faults Tst_i-SA1.
3. Set the i-th output of the shift register to low. Out_j=1 ($j \neq i$). As a result, the output of the i-th NAND gate is set to high.
4. $Rin(0)\uparrow$
5. If $Rout(2)\uparrow$ then $Aout(2)\uparrow$

6. *Rin(0)*↓

7. If *Tst*=1 then in the presence of the Tst_i-SA0 fault the *i*-th testable C-element is reset. Hence, if *Rout(2)*↓ the circuit is faulty. If the *Rout* output is held unchanged (*Rout*=1) the circuit is fault-free.

   If *Tst*=0 then in the presence of the Tst_i-SA1 fault the output of the *i*-th testable C-element is held unchanged. Hence, if *Rout(2)*↓ the circuit is fault-free otherwise it is faulty.

8. If the circuit is faulty, go to step 10, otherwise *i*=*i*+1 and go to step 9.

9. If $i \leq n$, where *n* is the number of micropipeline stages, then go to step 2. Otherwise, the circuit is fault-free and go to step 10.

10. End.

Note that during the test when all testable C-elements act as symmetric C-elements the outputs of the shift register are set to high. The testing of faults on the Tst_i inputs of the testable C-elements requires the corresponding output of the shift register to be reset. This can be implemented by applying a one on the *In* input of the circuit shown in Figure 5.14 after setting the outputs of the register to ones using the *Set* signal. As a result, after the application of one clock signal *Clk* to the shift register a zero is shifted to the first flip-flop of the register. Then the *In* input is reset and (*i*-1) clock signals are applied to the shift register to reset its *i*-th output.

Since the testable C-elements are tested sequentially and the testing of each C-element requires the application of one return-to-zero request signal *Rin* the shift register can be clocked by the *Rin* input. It is easy to show that stuck-at faults on the inputs and outputs of the NAND gates are detected during the test of the control circuit.

5.7.2 Testing for faults in the control circuit of the four-phase sequential circuit

The order of events on the inputs and outputs of the control circuit of the four-phase sequential circuit shown in Figure 5.11 can be written as follows:

$$Rin\uparrow\ Rout\uparrow \quad Aout\uparrow\ Rout\downarrow \quad Aout\downarrow\ Ain\uparrow \quad Rin\downarrow\ Ain\downarrow \tag{5.1}$$

Most of stuck-at faults in the control circuit cause the circuit to deadlock which can be easily identified. For instance, a stuck-at-0 fault on one of the inputs of the C-element *C3* is equivalent to a stuck-at zero fault on its output. In the presence of this fault the *Rout* signal is never reset which in turn causes the whole circuit to halt.

In the control circuit (see Figure 5.11) there are faults which cause premature firings on the outputs of C-elements *C2* and *C4*. These are 1-SA1, 2-SA1 and 3-SA0 faults. Fault 1-SA1 can be detected once after the resetting of all the C-elements. This fault causes the *Rout* signal to go high which must happen only when $Rin\uparrow$ (see (5.1)). In the presence of fault 2-SA1 *Ain* goes high just after a rising event on the *Rin* input. This fault violates the order of events written in (5.1). Fault 3-SA0 causes a premature rising event on the *Ain* output before a handshake is completed along the output channel (signals *Aout* and *Rout* must be returned to zero before *Ain* goes high). Faults 2-SA1 and 3-SA0 can be identified by checking the order of events on the control inputs and outputs in the circuit shown in Figure 5.11.

5.8 A case study of the AMULET2 register destination decoder

5.8.1 Design and implementation

The AMULET2 microprocessor has a circuit called the “register destination decoder”, a high-level implementation of which is shown in Figure 5.15 [Pav94].

The AMULET2 register destination decoder is a four-phase circuit. The decoder accepts a 16-bit binary vector (*In[15:0]*) which contains coded information about the availability of registers in the register bank of the microprocessor. For instance, a one in the *i*-th position of the input vector means that the *i*-th register must be processed. The output from the register destination decoder includes:

1) the four-bit address of the least significant ‘one’ in the input vector (*RD[3:0]*);

2) an active high output (*R15*) which indicates that the output address is ‘15’;

3) an active low output (*nTRM*) which indicates that the output register address contains the address of the most significant ‘one’ in the input vector.

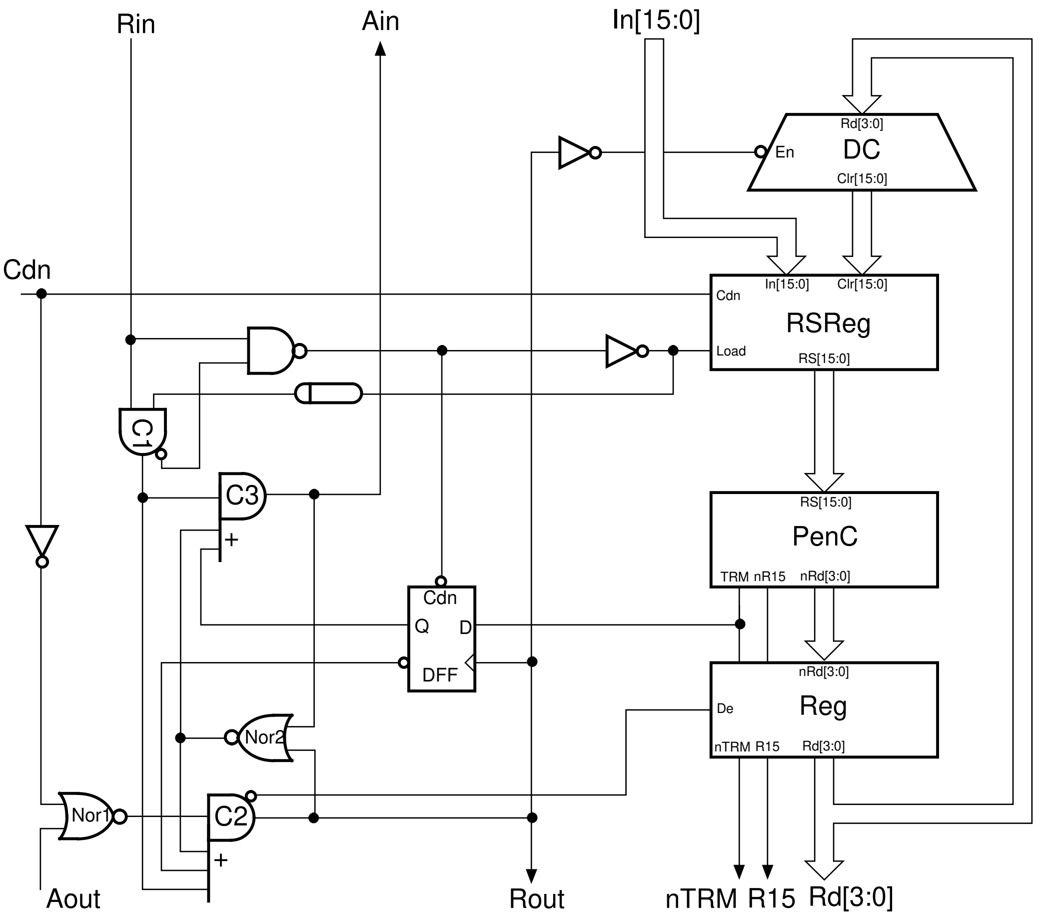


Figure 5.15: *AMULET2 register destination decoder*

Initially, the control inputs and outputs of the decoder are reset and all the C-elements are in zero states. The *clear down* signal (*Cdn*) is set to low to reset the RS latches of register *RSReg* and to reset the output of the asymmetric C-element *C2*.

When the *Cdn* input is high and the input data is stable on the *In* inputs of the decoder the *Rin* signal goes high. As a result, the output of the NAND gate is reset, setting the D-type flip-flop to state zero and loading the input vector into register *RSReg*. A rising signal from the NAND gate is delayed for long enough for the data stored in register *RSReg* to be processed by the priority encoder (*PenC*) and then stored into the latches of the output register (*Reg*). When the output of the symmetric C-element *C1* has been set to high the *load* signal of register *RSReg* is reset, indicating the completion of the loading of the input vector.

Figure 5.16a illustrates a gate level implementation of the RS latch which is used in *RSReg*. The NOR gates are configured to operate as a conventional RS latch with an active low reset input (*Cdn*). The RS flip-flop is set to high when both the *In* and *Load* signals are high. The state of the RS flip-flop can be changed to zero by setting its *Clr* input to high.

When the output of *C2* goes to high:

- a rising event is generated on the *Rout* output of the register destination decoder;
- the latches of register *Reg* are closed (the *De* input is reset);
- the data from the *TRM* output of *PenC* is latched in the master latch of the D flip-flop;

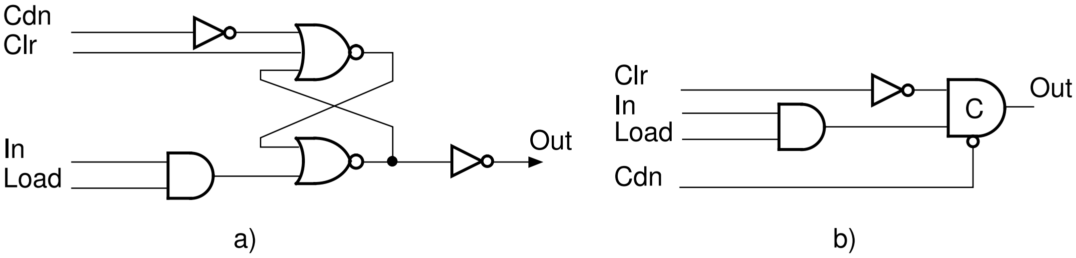


Figure 5.16: *Gate-level implementations of the RS latch using: a) a conventional RS latch; b) a symmetric C-element*

- the address decoder (*DC*) is enabled (if $\overline{En}$=0 the outputs of DC are held at zero).

The outputs of *Reg* can be read by the environment. Concurrently, the address stored in register *Reg* is decoded by *DC* and sent to the inputs of register *RSReg*. Thus, the least significant ‘one’ of the vector stored in *RSReg* is cleared. The modified vector is sent to the priority encoder *PenC*. When a rising event is received at the *Aout* input of the register destination decoder the output of the NOR gate *Nor1* is reset, setting *C2* to state zero, and the address of the least significant ‘one’ is stored in register *Reg* (*De*=1). The *Rout* output is reset and the output of *Nor2* is set to high. The address decoder *DC* is disabled and the master latch of the D flip-flop is opened, storing the data from the *TRM* output of *PenC* in its memory.

When the *Aout* signal returns to zero the output of *C2* is set to high and the procedure described above is repeated until only one RS latch of register *RSReg* is in state one. Then the *TRM* output of *PenC* is set to high. The state of the D flip-flop is changed to one by a rising event generated on the *Rout* output. As a result, when *Aout*=1 the output of *C2* is reset (*Rout*=0) and the output of *Nor2* goes high, setting the output of *C3* to high. Thus, the *Ain* signal goes high. Once the *Rin* signal has returned to zero *C1* goes to state zero, resetting the output of *C3*. When all the control signals have been reset the register destination decoder accepts new data and repeats the sequence of events described above.

5.8.2 Design for testability

The detection of stuck-at faults in the register destination decoder is not a trivial task because of its sequential properties. The scan test technique described in this chapter can be used to reduce the test complexity of the decoder. For instance, the latches of *Reg* (the outputs of which are coupled to the inputs of the address decoder) can be replaced by scan latches (see Figure 5.4). Thus, the inputs of *DC* can be controllable through the scan latches which are connected into one scan register. The sequential function of the RS latches of *RSReg* can be converted into a combinational one using the following circuit modification.

Table 5.4: *State tables for a conventional RS latch and the C-element performed its function*

Inputs		RS latch designed from NOR gates	RS latch designed from NAND gates	Muller C-element
R	S	Q_t	Q_t	Q_t
0	0	Q_{t-1}	-	Q_{t-1}
0	1	1	1	1
1	0	0	0	0
1	1	-	Q_{t-1}	Q_{t-1}

RS latch implementation

Figure 5.16b shows the RS latch of register *RSReg* implemented using a symmetric C-element. The C-element is reset by an active low *Cdn* signal. The operational properties of such an RS latch are equivalent to these of the latch illustrated in Figure 5.16a.

Table 5.4 shows the state tables for a conventional RS latch (designed from both NOR and NAND gates) and the C-element performing the same function. In both RS latch implementations there are illegal input combinations which make the behaviour of the latch unpredictable. The implementation of the RS latch using the symmetric C-element does not have any illegal input combinations. However, a simple analysis of the behaviour of the C-element which operates as an RS latch shows that the C-element imposes certain limitations on its input transitions. Suppose both the *R* and *S* inputs were previously set to high or low. Then the changing of the inputs to low or high at nearly the same time causes opposite signal transitions on the inputs of the C-elements. Thus, the next state of the C-element depends on how fast a particular input transition completes.

According to the specification of the register destination decoder the inputs to the RS latch are not changed at the same time. Hence, the circuit shown in Figure 5.16a can be replaced by the one in Figure 5.16b.

Figure 5.17 illustrates a CMOS implementation of the symmetric C-element with an active low reset signal (*Cdn*). When the *Cdn* signal is high this C-element operates in a

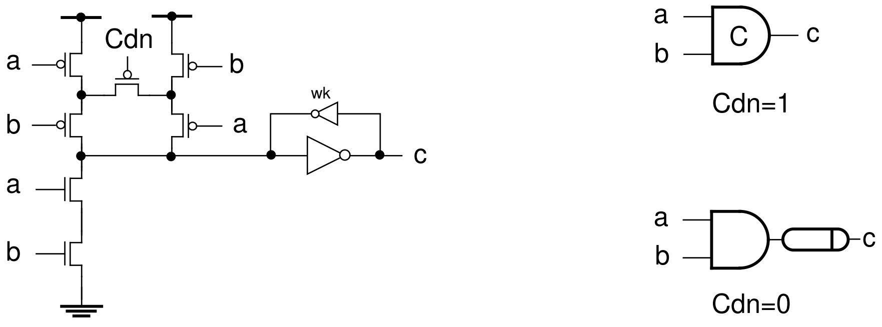


Figure 5.17: *CMOS implementation of the symmetric C-element with an active low reset input*

similar way to the pseudo-static C-element (see Figure 4.1d). When the *Cdn* input is reset the *p* transistor stacks are configured so that if one of the inputs is low the output of the circuit is reset. As a result, the C-element operates as an AND gate with an output delay as shown in Figure 5.17. Note that the design of the C-element shown in Figure 5.17 can be derived easily from the testable C-element implementation illustrated in Figure 4.2.

Thus, the use of the C-element (in Figure 5.17) in the RS latch design allows register *RSReg* to be transformed into a set of AND gates reducing the test complexity of the register destination decoder. In addition, the problem of detecting stuck-at-0 faults in the feedback wires of the RS latch (in Figure 5.16a) which convert its static function into a dynamic one no longer exists.

Testing for stuck-at faults

Figure 5.18 illustrates a scan testable implementation of the AMULET2 register destination decoder. In this design, register *Reg* (see Figure 5.15) is replaced by register *ScanReg* with scan latches the implementation of which is illustrated in Figure 5.5 (the latch control block is not shown). The latches with outputs *nTRM* and *R15* are built from the single-phase static latch shown in Figure 5.3 since their outputs do not stimulate the inputs of *DC*.

Single stuck-at faults in the testable structure of the register destination decoder can be divided into two groups:

1. Stuck-at faults in the data paths.

2. Stuck-at faults in the control circuit.

The decoder can operate in normal operation (*Tst*=0) and test mode (*Tst*=1). In normal operation mode the circuit shown in Figure 5.18 operates in the same way as was described in section 5.8.1. In test mode the register destination decoder is tested for most stuck-at faults in its data and control paths. The testing for faults in normal operation mode is necessary to guarantee that extra logic elements (such as *Nor3*, *And2*, *Or1* and *Or2*) incorporated into the original design of the register destination decoder are fault-free.

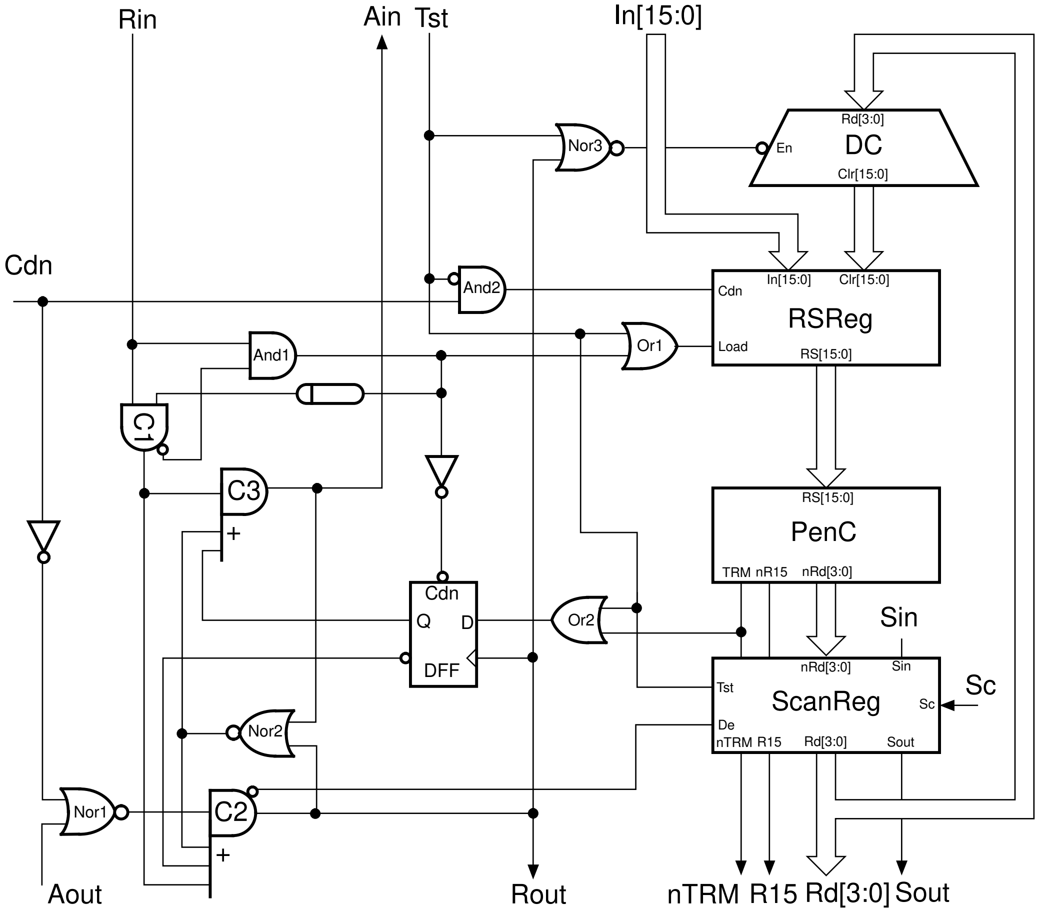


Figure 5.18: *AMULET2 register destination decoder with scan features*

Testing the register destination decoder in normal operation mode.

1. The circuit is reset and the Boolean *Tst* is set to low.

2. A test of all ones is applied to the *In* inputs of the circuit. As a result, the circuit performs 16 handshakes along its output channel clearing 'ones' in the vector stored in register *RSReg*. The outputs from register *ScanReg* are read by the environment.

3. Once *R15* is high and the *Rout* signal has set to high (the latches of *ScanReg* are closed) the outputs of *ScanReg* can be read by shifting its contents out to the *Sout* output. Concurrently, a new test vector is shifted into register *ScanReg*.

4. When the test has been shifted into register *ScanReg* the circuit is set to test mode by setting the Boolean *Tst* to high. Thus, the test stored in the latches of *ScanReg* is applied to the inputs of *DC*. The *Rd* outputs of *ScanReg* remain unchanged regardless of the *nRd* inputs and the *De* input (see Figure 5.4).

5. When the control inputs and outputs of the register destination decoder have been returned to zero it is tested in test mode.

Testing the register destination decoder in test mode.

1. In test mode address decoder *DC* is enabled (the output of *Nor3* is low), register *RSReg* is set to *load* mode (the *load* input is set to high) and its C-elements operate as two-input AND gates (the output of *And2* is reset). Note that the data input of the D flip-flop is set to high permanently which allows only one handshake to be completed on both the input and output channels.

2. When a test vector has been applied to the *In* inputs of the register destination decoder *Rin* goes high, resetting the D flip-flop and setting the output of *C2* to high. As a result, *Rout* goes high, the state of the D flip-flop is changed to one and the test responses from the combinational circuit combined by *DC*, *AND* gates of *RSReg* and *PenC* are latched in *ScanReg*.

3. The Boolean *Tst* is reset and the contents of the scan latches of *ScanReg* are shifted out while a new test vector is shifted in. Note that the contents of *ScanReg* can be read directly from its outputs *nTRM*, *R15* and *Rd[3:0]*.

4. When the test has been shifted in *ScanReg* the Boolean *Tst* is set to high and a handshake along the output channel of the circuit can be completed.

5. Once all the control signals of the register destination decoder have been reset a new test can be applied to its *In* inputs.

A set of 47 test patterns for the united combinational circuit has been obtained with the help of automatic test generation program tools developed at Virginia Polytechnic Institute [LeeTR93].

A fault simulation analysis carried out using the *SIMIC* fault simulator revealed that all single stuck-at faults in the data paths and most stuck-at faults in the control circuit can be detected during the testing of the register destination decoder in normal and test modes.

Stuck-at-1 faults on the asymmetric inputs of the C-elements *C2* and *C3* cause premature rising events on their outputs. These faults can be detected by converting the asymmetric C-elements into symmetric ones using the technique described in section 5.7. An extra control input (*TCel*) is required to switch the operation modes of the testable C-elements. If *TCel*=0 the C-elements operate as asymmetric ones, otherwise they operate as symmetric C-elements.

The test algorithm for detecting stuck-at-1 faults on the inputs of *C2* can be described as follows:

1. Reset the register destination decoder. *Tst*=0.

2. *TCel*=1. *C2* and *C3* operate as symmetric C-elements.

3. *Rin*↑. As a result, the output of *C2* is set to high and *Rout* goes high. The state of the D flip-flop is changed to one since the *TRM* output of *PenC* is set to high by applying the appropriate test on the *In* inputs.
4. *Rin*↓ and *Aout*↑. The output of *C2* is reset if there are no stuck-at-1 faults on its inputs and remains high in the presence of these faults.

Stuck-at-1 faults on the asymmetric inputs of *C3* can be tested using the following test algorithm:

1. Reset the register destination decoder. *Tst*=0.
2. *TCel* is set to low. Thus, C-elements *C2* and *C3* operate as the asymmetric C-elements.
3. *Rin*↑. The state of *C2* is changed to one. The output of the D flip-flop is set to high since the *TRM* output of *PenC* is set to high by applying the appropriate test on the *In* inputs.
4. *Aout*↑. The state of *C2* is changed to zero and *Ain*↑.
5. *TCel*=1. Hence, *C2* and *C3* operate as the symmetric C-elements.
6. *Rin*↓
7. *Rin*↑. The state of the D flip-flop is set to zero, resetting the output of *C3*. This is possible because a rising signal from the output of *And1* is delayed for long enough for the D flip-flop and *C3* to change their states. In the presence of stuck-at-1 faults on the inputs of *C3* the *Ain* output remains high, otherwise the *Ain* output is reset.

Note that the operation modes of the C-elements *C2* and *C3* can be tested easily since if they act as symmetric C-elements the register destination decoder halts in normal operation mode. If the C-elements act as asymmetric C-elements during the test this can be identified by a changed order of events on the control lines of the decoder.

5.8.3 Cost comparisons

The AMULET2 register destination decoder and its testable version have been implemented in CMOS technology on a $1\mu m$ double layer metal process. Table 5.5 shows cost comparisons of the AMULET2 register destination decoder without and with testability features. The performance and dynamic power dissipation have been estimated for both designs in their normal operation modes with the help of *SIMIC* design verification tools. The performance has been measured by applying a set of 16-bit vectors each of them containing 15 ‘zeros’ and a ‘one’. As a result, the decoder processes each input vector by completing one handshake along its input and output channels.

Table 5.5 shows that the testable register destination decoder demonstrates 8% performance degradation and 11% area overhead compared to the original design without testability. The power dissipation of the testable design has increased slightly.

Table 5.5: *Cost comparisons for the AMULET2 register destination decoder designs*

Design	Performance in normal operation mode, ns/test	PD[a], %	Area, $\times 10^{-2}\ mm^2$	AO[b], %	Power consumption in normal operation mode, nJ/test
Without testability	25.5	n/a	14.6	n/a	33.8
With scan features	27.7	8%	16.2	11%	35.1

a. PD is the performance degradation;
b. AO is the area overhead.

5.9 Summary

The scan test technique presented in this chapter supports testing for stuck-at and delay faults in two-phase and four-phase micropipelines and asynchronous sequential circuits based on the micropipeline design style. The internal inputs and outputs of the processing logic blocks are fully controllable and observable through the scan path. The test patterns are scanned in and the test results are shifted out from the stage registers, unified into one shift register. The scan path of the testable micropipeline is controlled by the scan test control block. Two implementations of the scan test control blocks which follow two different communication protocols have been presented. The universal structures of the scan test control blocks allow them to be adapted for arranging either a global asynchronous shifting of the test data between different parts of the chip or a local scan path within a particular block.

Testing for stuck-at faults in the control part of the two-phase micropipeline is easier than for the four-phase micropipeline. The use of asymmetric C-elements in four-phase micropipelines creates the potential danger of premature firing, the detection of which is not trivial. Stuck-at faults which cause premature firings in the control part of the four-phase micropipeline can be detected either by converting asymmetric C-elements into symmetric ones or by checking the order of events on the control wires.

The proposed testable micropipeline structure greatly simplifies the testing of micropipelines by reducing the test complexity to that of the processing logic. The overall overhead can be estimated only for a particular case since it depends on the complexity of the processing logic and the chosen signalling protocol.

A case study of the AMULET2 register destination decoder has demonstrated the practical feasibility of the scan test technique presented in this chapter. It has been shown how a symmetric C-element can be used to perform the function of an RS latch. The proposed implementation of the C-element with a reset input allows it to be transformed into an AND gate, making the testing of the register destination decoder easier. The scan testable design of the register destination decoder exhibits low area overhead and performance degradation.

Chapter 6 : Design for Random Pattern Testability of Asynchronous Circuits

Although DFT methods give test engineers a great opportunity to simplify the testing of VLSI designs, test generation and fault simulation costs are still large and are rising with the increasing complexity of VLSI devices. As a result, random (or, more correctly, pseudo-random) testing (see Appendix A) becomes a viable alternative for testing VLSI circuits for at least two reasons:

- Algorithmic test generation methods for modern VLSI circuits are becoming too expensive in terms of computational time. The use of pseudo-random pattern generators (PRPG) for the testing of VLSI circuits does not require any special properties from the circuit under test except that it does not have illegal input combinations.
- It is possible to build VLSI structures with BIST features (see Appendix B).

In this chapter two-phase and four-phase designs of sequential circuits for random pattern testability and a micropipeline structure with BIST features are considered.

6.1 Asynchronous pseudo-random pattern generator and signature analyser designs

Clearly, both the pseudo-random pattern generator, which is used to generate stimulus for the circuit under test, and the signature analyser, which collects the test results, must operate asynchronously following the signalling protocol of the test object. The designs of the asynchronous pseudo-random generator and signature analyser can be imple-

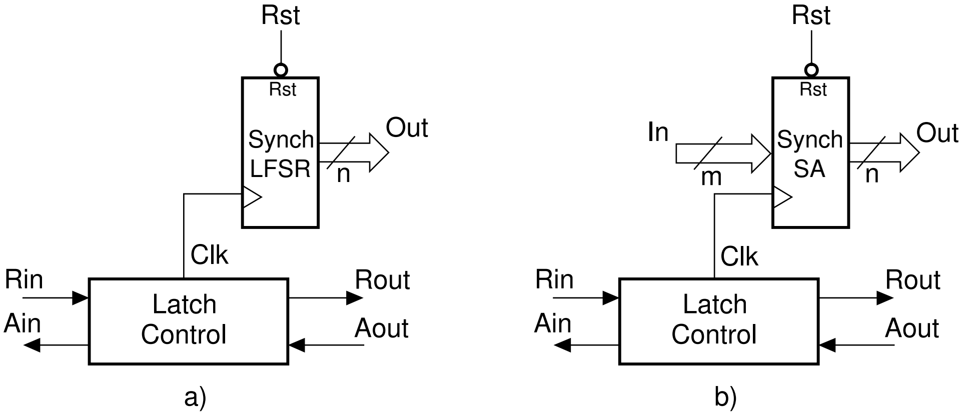


Figure 6.1: *Asynchronous a) pseudo-random pattern generator; b) signature analyser based on a synchronous LFSR*

mented using the corresponding synchronous LFSR and signature analyser described in Appendix A. Figures 6.1a and 6.1b illustrate the design of an asynchronous LFSR and signature analyser respectively. Both the synchronous LFSR in Figure 6.1a and the synchronous signature analyser in Figure 6.1b are clocked by the latch control circuits which can follow either two-phase or four-phase signalling. The two-phase latch control circuit can be implemented using the circuit shown in Figure 5.1. An example of the four-phase latch control circuit is shown Figure 6.2. The *Rin* signal is buffered to guarantee the required drive strength of the clock signal (*Clk*). The *Rout* and *Aout* signals or the *Rin* and *Ain* signals in the latch control circuits of the asynchronous signature analyser or LFSR can be used by the environment to calculate the number of test vectors applied to the circuit under test respectively.

Note that the synchronous LFSR and a signature analyser are built using shift registers with master-slave flip-flops. This allows two different data latching schemes to be used in four-phase signalling. According to the first data latching scheme the input data must be stable before *Rin* goes high until *Aout* is set to high. In the second data latching

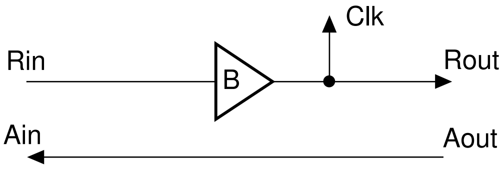


Figure 6.2: *Four-phase latch control*

scheme the input data must be stable until a handshake is completed along the input (*Rin* and *Ain* control wires) and output channels (*Rout* and *Aout* control wires), i.e., all the control signals are returned to zero. In contrast the two-phase latch control circuit allows the input data to be changed on the inputs of the signature analyser when a rising or falling event is generated on the *Rout* output (the clock signal *Clk* has already returned to zero).

Consider the circuit shown in Figure 6.2. When *Rin* goes high setting the clock to high the data is latched in the master latches of the register whereas the slave latches are transparent. As a result, a new vector is produced on the outputs of the shift register. When the *Rin* signal is returned to zero the master latches of the register are transparent whereas the slave latches are opaque. As a consequence, the outputs of the shift register remain unchanged. Since the LFSR (see Figure 6.1a) does not have inputs it can be used in both data latching schemes. Clearly, it is important for the signature analyser (see Figure 6.1b) to have its input data stable only before *Rin* goes high. Thus, the input data can be changed on the inputs of the signature analyser either after *Aout* is set to high (the first data latching scheme) or after the completion of a handshake along its control channels (the second data latching scheme).

An asynchronous pseudo-random pattern generator and signature analyser can be implemented as handshake circuits. Since the generator and signature analyser themselves are built using the LFSR their handshake implementations are similar to the design of the handshake storage element illustrated in Figure 1.8d. The pseudo-random generator can be built either as an autonomous block (see Figure 6.3a) or request-driven handshake circuit (see Figure 6.3b). The outputs of the signature analyser depend on its inputs and the initial seed of its LFSR. As a result, the handshake implementation of the signature analyser must be built as a request-driven block (see Figure 6.3c). Note that initialization signals for the LFSRs of the generator and signature analyser are not shown in Figure 6.3.

The autonomous generator shown in Figure 6.3a starts to perform after the activation of its activation channel. When the sequencer has completed a handshake along its left

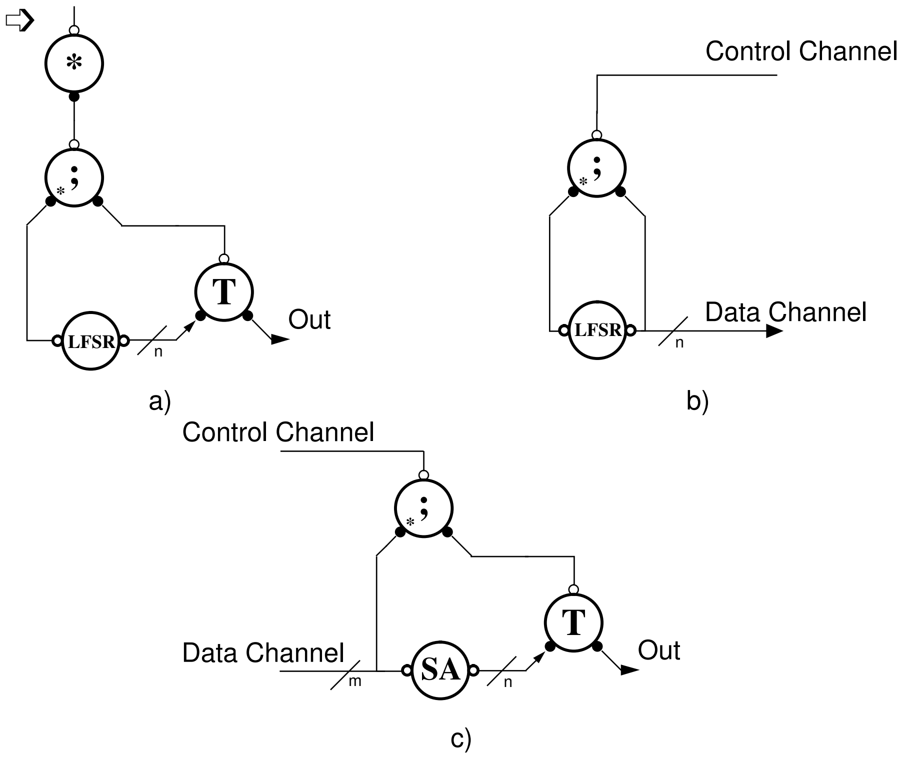
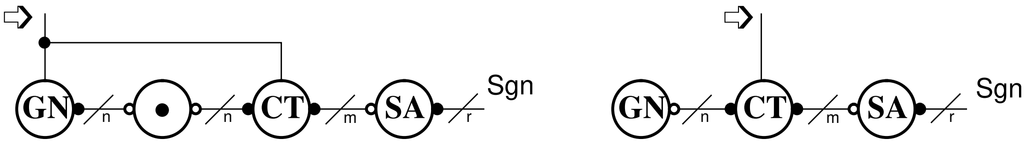


Figure 6.3: *Handshake implementations of the a) autonomous; b) request-driven pseudo-random pattern generators; and c) the signature analyser*

channel a new vector is ready on the outputs of the LFSR. As a result, the transferer is triggered by a request along the right channel of the sequencer fetching the data from the outputs of the LFSR and generating a request for the test object. The circuit under test accepts the test vector by completing a handshake along the output channel of the generator. Afterwards the generator produces a new pseudo-random vector on the outputs of the LFSR and activates its output channel. The request-driven generator illustrated in Figure 6.3b produces a new test vector only when its control channel is activated by the circuit under test. Compared to the request-driven generator the use of the autonomous pseudo-random generator is more effective in terms of performance since it acts in parallel with the circuit under test. The signature analyser shown in Figure 6.3c is a request-driven handshake circuit which is activated when the input data is stable and its control channel is triggered by the circuit under test. The transferer is used to transmit intermediate signatures from the outputs of the four-phase signature analyser (SA) to the environment and count the number of tests applied to the inputs of the test object. Note that



a) b)

Figure 6.4: *Random test interface using a) the autonomous generator; b) the request-driven generator*

the signature analyser uses the second data latching mechanism where the input data is changed on the inputs of the four-phase signature analyser after completing a handshake along its left passive channel.

Figure 6.4 illustrates the random test interface using the autonomous (see Figure 6.4a) and the request-driven pseudo-random pattern generator (see Figure 6.4b). The random testing procedure is started by setting the global start signal (⇨) to high. When the autonomous generator is used the generator (GN) and the circuit under test (CT) operate in parallel (see Figure 6.4) and their operation is synchronized using a handshake element called *passivator* [FarnTR96].

An implementation of the passivator is shown in Figure 6.5. When channel *a* is ready to pass the data (the request signal a_r is high) and channel *b* is ready to accept the input data (the request b_r is set to high) the state of the symmetric element is changed to one setting the acknowledge signals a_a and b_a to high. The transmission of the data is completed by resetting the request signals along channels *a* and *b*. Note that both ports of the passivator are passive.

The use of the request-driven generator (see Figure 6.4b) does not require the passivator since the operation of the generator and signature analyser is controlled by the circuit under test. Note that the request-driven generator has one passive port. The number of random test vectors applied to the inputs of the test object is equal to the number of

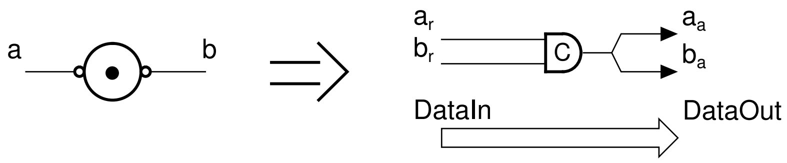


Figure 6.5: *An implementation of the passivator*

Table 6.1: *State sequence of the 2-bit PRPG*

State	Q_0	Q_1	Q_2	Q_3	T_0	T_1	State	Q_0	Q_1	Q_2	Q_3	T_0	T_1
0	0	0	0	0	0	0	9	1	0	1	0	0	0
1	1	0	0	0	0	0	10	1	1	0	1	1	1
2	0	1	0	0	1	0	11	1	1	1	0	1	0
3	0	0	1	0	0	0	12	1	1	1	1	1	1
4	1	0	0	1	0	1	13	0	1	1	1	1	1
5	1	1	0	0	1	0	14	0	0	1	1	0	1
6	0	1	1	0	1	0	15	0	0	0	1	0	1
7	1	0	1	1	0	1	16	0	0	0	0	0	0
8	0	1	0	1	1	1	17	1	0	0	0	0	0

requests generated on the output channel (Sgn) of the signature analyser. After the required number of handshakes along channel *Sgn* the signature produced on the output of the signature analyser is compared with the good one. If they are equal, the circuit is fault-free, otherwise it is faulty.

As was mentioned in Appendix A an LFSR can be designed to generate all possible binary vectors on its outputs. Figure A.2b illustrates an implementation of the 4-bit synchronous LFSR which goes through all 16 possible states including the ‘all zeros’ state. The output sequence of this LFSR has the following unique property:

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

Table 6.1 contains the states of the 4-bit LFSR shown in Figure A.2b. 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 be found easily in the 2-bit output sequence compound by 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 6.1. 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

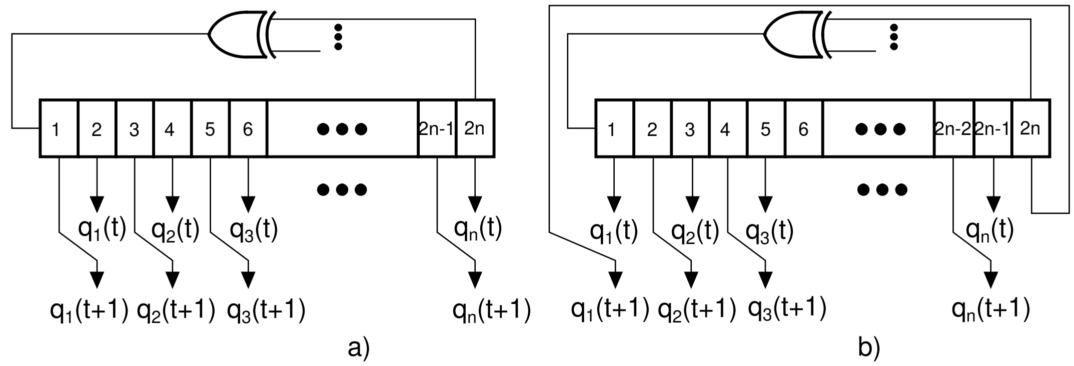


Figure 6.6: *Mechanisms for generating pseudo-random vectors using a) even and b) odd outputs of the LFSR*

sequence of the pseudo-random pattern generator after it has passed through all its states.

[1]*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 6.6a 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 6.6). 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 6.6b. 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 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

---

[1] The idea of this proof was suggested by Prof. John Brzozowski in private discussions on this topic.

ones (see Figure 6.6b) 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. ❑

This property of the LFSR will be used to design of a micropipeline with BIST features (see section 6.7).

6.2 Sequential circuit designs

The two-phase and four-phase designs of a asynchronous sequential circuit were discussed in chapter 5. The use of normally closed registers *Reg1* and *Reg2* is preferable for scan testing since the contents of the state registers can be observed and controlled along a scan path. If the test data is not shifted in and out of the state registers then *Reg2* can be transparent initially. As a result, the sequential circuit latch control is faster than the control of the circuit with normally closed state registers. There is no need to open the latches of *Reg2* every time the data is ready on the outputs of the combinational circuit. The data from the *SO* outputs of the combinational circuit is already stored in *Reg2* when a rising event is generated on the *Rout* output. Note that the latch control of registers *Reg1* and *Reg2* for the two-phase sequential circuit shown in Figure 5.9 assumes that both the registers are closed initially.

Figure 6.7 shows the design of a four-phase sequential circuit with the normally closed *Reg1* and the normally transparent *Reg2*. Initially, all C-elements are reset. The latches of *Reg1* are set to their initial states. When a rising signal arrives at the *Rin* input the output of the C-element *C1* goes high. This rising signal is delayed for long enough for the

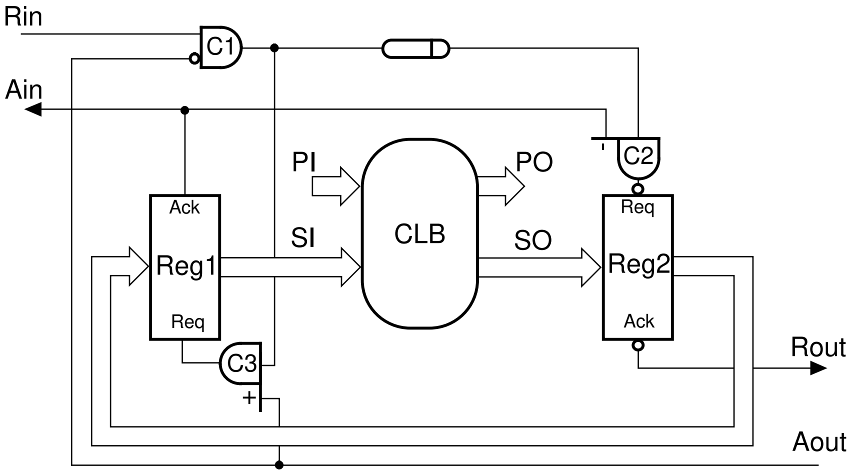


Figure 6.7: *Four-phase sequential circuit with the normally closed Reg1 and the normally transparent Reg2*

data on the *PO* outputs to stabilise and the data from the *SO* outputs to be stored in *Reg2*. As a result, the output of the asymmetric C-element *C2* is set to high latching the data from the *SO* outputs of the combinational circuit in *Reg2*. Once the data is latched in *Reg2* a rising event is produced on the *Rout* output. When the output data has been read by the environment a rising event is produced on the *Aout* input. The state of the asymmetric C-element *C3* is changed to one. Thus, the latches of *Reg1* become transparent storing the input data in their memory. When the data has been stored in register *Reg1* the *Ain* signal goes high.

Note that when *Rout* goes high the data from the *PO* outputs must be latched by the environment. This is because a rising event on the *Aout* input causes the latches of *Reg1* to be transparent changing the *SI* inputs of the combinational circuit. Thus, the *PO* outputs of the combinational circuit can be changed before the *Rout* and *Ain* signals are reset.

Once the *Rin* signal has returned to zero the output of *C1* is reset. As a result, the output of *C3* is reset closing the latches of *Reg1*. When the data has latched in *Reg1* the *Ain* signal is returned to zero enabling the output of *C2* to be reset. If both inputs of *C2* are reset its state is changed to zero making the latches of *Reg2* transparent. As a result, *Rout* goes low. When the *Aout* signal has returned to zero the sequential circuit is ready to accept new data and the sequence of events described above is repeated.

6.3 Parallel random testing of sequential circuits

6.3.1 Probabilistic properties of an XOR gate

It has already been reported that a 2-input XOR gate can be used to improve the effectiveness of the random testing of synchronous VLSI circuits [Rom89, Souf95].

Consider the XOR gate with two inputs (*a* and *b*) and output *c* shown in Figure 3.2. Let p_a and p_b be the probabilities of a one and zero on input *a* and *b* respectively. Suppose that there is no correlation between inputs *a* and *b*. Then, according to equation 3.6 the probability of a one (p_c) on output *c* of the XOR gate can be calculated as follows:

$$p_c = p_a + p_b - 2p_a p_b \tag{6.1}$$

It is easy to show that if p_a=0.5 in equation (6.1) then p_c=p_a=0.5 regardless of the value of p_b. This probabilistic property of the XOR gate can be described as follows.

If independent and equiprobable random signals are applied to one of the two inputs of the XOR gate then:

1. The output signals produced by the gate are equiprobable.
2. The probabilistic properties of the output sequence do not depend on the probabilistic properties of the input sequence applied to the other input.

Clearly, a fault effect can be transmitted easily through the XOR gate since any faulty signal applied to one of its inputs changes its fault-free output.

6.3.2 Sequential circuit designs for random-pattern testability

The probabilistic properties of the XOR gate can be used for at-speed random testing of synchronous sequential circuits [Souf95]. In this section asynchronous designs for random-pattern testability of sequential circuits are considered.

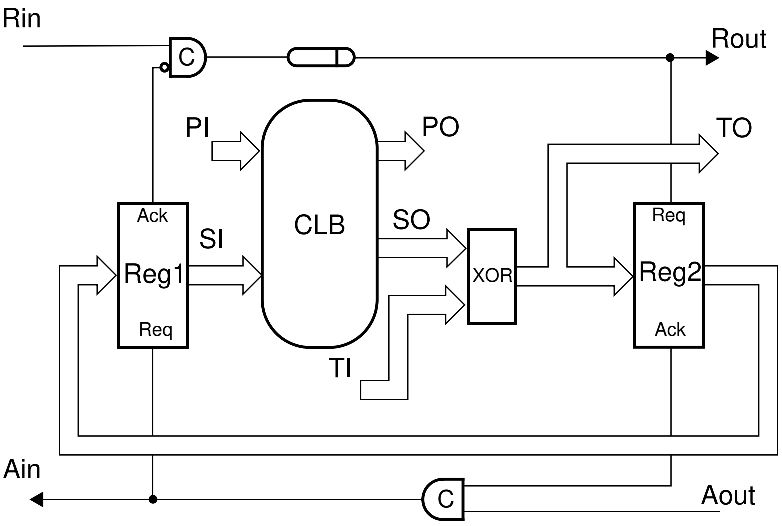


Figure 6.8: *Two-phase sequential circuit with parallel random testing*

Figures 6.8 and 6.9 show the designs of two-phase and four-phase random pattern testable sequential circuits. The block of XOR gates performs a bit-wise XOR operation between the *SO* outputs of the combinational circuit and additional test inputs (TI). The outputs of the XOR block are connected to the inputs of *Reg2* and test outputs (TO).

During normal operation mode the *TI* inputs are kept at zero and the sequential circuit operates according to its specification.

In test mode test vectors generated by the pseudo-random generator are applied to the *TI* inputs and the *PI* inputs. The test results are observed on the *PO* and *TO* outputs by the

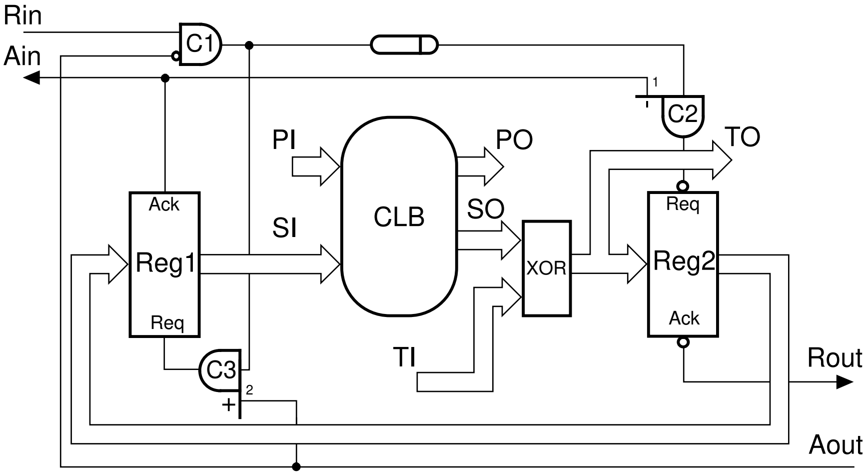


Figure 6.9: *Four-phase sequential circuit with parallel random testing*

signature analyser. Note that vectors produced by the pseudo-random generator are equiprobable and independent. Thus, according to the probabilistic properties of the XOR gate the output vectors produced by the XOR block are equiprobable and independent of the probabilistic properties of the *SO* outputs. As a result, equiprobable and independent stimuli are applied to the inputs of *Reg2* and fault effects from the *SO* outputs can be detected on the *TO* outputs. After the application of each pseudo-random test to the *PI* inputs of the sequential circuit the combinational circuit and registers *Reg1* and *Reg2* are tested concurrently. This random testing procedure is called ‘parallel’ because random test patterns are applied to the *PI* and *SI* inputs of the combinational circuit in parallel.

Stuck-at faults on the inputs and outputs of *Reg1* and *Reg2* are equivalent to the corresponding faults on the *SI* inputs of the combinational circuit. Stuck-at faults on the inputs of the XOR block can be easily detected on the *TO* outputs. As a consequence, the total test time for detecting stuck-at faults in the data paths of the sequential circuits (see Figure 6.8 and 6.9) is determined by the random test time of their combinational circuits.

Number of random patterns required for testing combinational circuits

There are two important characteristics of random testing:

- the number of patterns which must be produced by the test pattern generator to produce the desired set of test vectors;
- the probability of detecting all possible faults from the predetermined class of the circuit’s faults.

The first parameter reflects the practical usability of random testing or simply the random pattern testability of the circuit. The second parameter is a characteristic of the quality of random testing.

Suppose that to detect all the stuck-at faults in the predetermined set of the combinational network's stuck-at faults it is necessary to generate on its *N* inputs ($N=n+m$, where *n* and *m* are the number of *SI* and *PI* inputs to the combinational circuit respectively) a random test of length *L*.

The test confidence probability threshold (p_t) is the probability that all the stuck-at faults in the circuit will be detected during its random testing. The escape probability threshold of the test ($q_t = 1 - p_t$) is the probability that at least one stuck-at fault from the predetermined set of faults will not be identified.

Thus, the upper bound on the number of random test patterns (*L*) applied to the inputs of the combinational circuit can be estimated using the following formula [Savir84, Wag87]:

$$L \ge \frac{\ln (q_t / r)}{\ln (1 - p_d)}, \tag{6.2}$$

where p_d is the minimal detection probability of a fault from the set of the circuit's faults; *r* is the number of faults which have the minimal detection probability p_d.

Testing for faults in the control circuits

As was shown in chapters 1 and 5 stuck-at faults on the control lines of the two-phase sequential circuit illustrated in Figure 6.8 are easy to detect since they cause the circuit to halt. These faults are detected either in normal operation mode or test mode.

Most stuck-at faults on the control lines in the four-phase sequential circuit shown in Figure 6.9 violate the communication protocol between the circuit and the environment causing the circuit to halt. There are some faults which can cause premature firings. For example, fault 1-SA0 causes a premature falling event on the output of *C2*. Fault 2-SA1 causes a premature rising event on the output of *C3*.

Fault 1-SA0 cannot be detected by checking the order of events on the control inputs and the outputs of the circuit. This fault can be detected in test mode by converting the

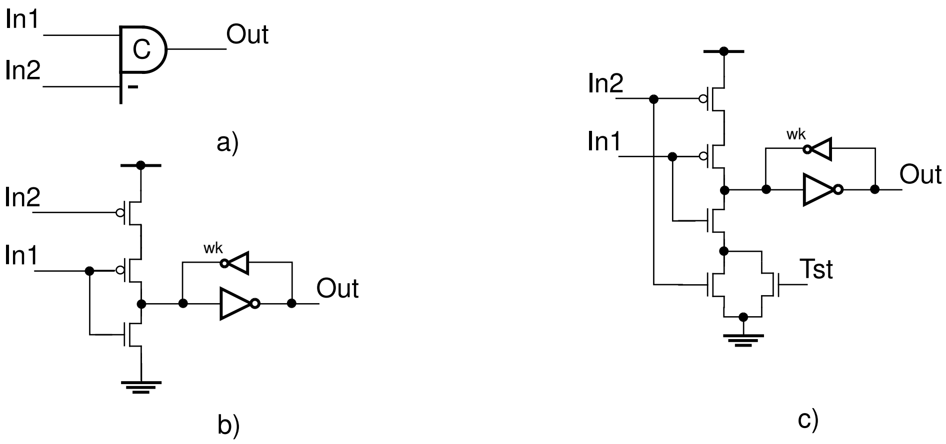


Figure 6.10: *Asymmetric C-element: a) symbol; b) CMOS implementation; c) testable CMOS implementation*

asymmetric C-element *C2* into a symmetric one as shown in Figure 6.10. The symbolic representation of the C-element *C2* and its CMOS implementation are illustrated in Figure 6.10a and 6.10b respectively. When inputs *In1* and *In2* are low the p-transistor stack is on and the output of the C-element is reset. If *In1*=1 the output of the C-element goes high. When *In1*=0 and *In2*=1 the state of the C-element remains unchanged. Figure 6.10c illustrates a CMOS implementation of the C-element which can operate as a symmetric C-element (*Tst*=0) and the asymmetric one (*Tst*=1) shown in Figure 6.10b.

After the initialization of all control wires *C2* is set to test mode (*Tst*=0) when it operates as the symmetric C-element. In the fault-free circuit, when the *Rin* and *Aout* inputs are set to high the *Ain* and *Rout* signals go high. In the presence of fault 1-SA0 the output of *C2* cannot be set to high. Thus, the *Rout* output remains unchanged, i.e. *Rout*=0. In order to check if *C2* acts as the symmetric or asymmetric C-element the *Rin* signal must be set to high after the initialization of the circuit. If *Rout* goes high *C2* operates as the asymmetric C-element, otherwise it acts as the symmetric one.

Fault 2-SA1 can be detected easily since it violates the order of events on the inputs and outputs of the circuit. After the initialization the *Rin* signal is set to high. In the presence of fault 2-SA1 the output of *C3* is set to high. As a result, *Ain* goes high whereas this must happen only after setting the *Aout* input to high.

6.3.3 Analysis of the parallel random testing technique

Advantages

- The implementation of the parallel random testing technique does not require the control structure of the sequential circuit to be changed.
- The total test time depends of the random test time of the combinational logic block and can be calculated easily.
- The sequential circuit is tested at its normal speed which allows a large number of random test vectors to be applied to the inputs of its combinational circuit.

Disadvantages

- The parallel random testing technique requires n extra test inputs and n extra test outputs, where n is the number of the SO outputs.
- The technique introduces a certain level of hardware redundancy which includes the block of two-input XOR gates, the pseudo-random generator and the signature analyser incorporated into the testable circuit design.
- The block of XOR gates in the feedback of the sequential circuit requires the introduction of an extra delay matching signal delays through the XOR gates.

6.4 Bit-serial random testing of sequential circuits

The general idea for alleviating the test problem of asynchronous sequential circuit shown in Figures 5.9 and 6.7 is common for all sequential circuits, i.e. during the test the whole sequential circuit must be divided into a combinational part and memory elements which are tested separately.

6.4.1 Two-phase sequential circuit design

Figure 6.11 illustrates the design of a testable two-phase sequential circuit. This circuit contains some additional elements such as a register (*Reg3*) for collecting the test data from the *SO* outputs of the combinational circuit, a block of XOR gates to mix the test data and the multiplexer to switch the data flow during the test phase. Also there are two XOR gates, multiplexers and a toggle element to provide the proper control signalling.

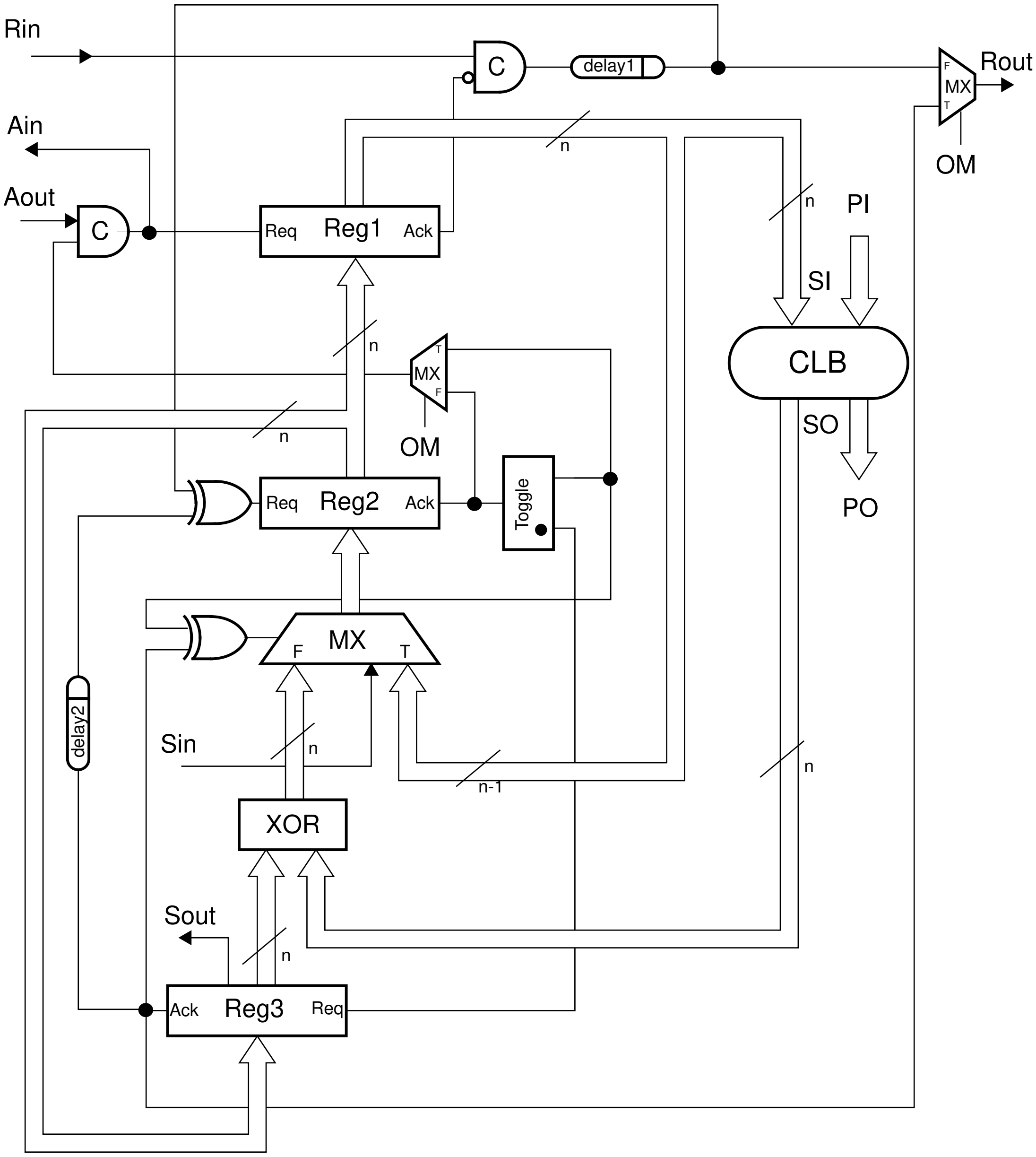


Figure 6.11: *Two-phase sequential circuit with bit-serial random testing*

The sequential circuit performs in two modes (normal operation and test mode) which are set by switching the Boolean signal on the operation mode input (*OM*). There are two additional pins (*Sin* and *Sout*) inserted in the design to scan test patterns into *Reg1* for stimulating the *SI* inputs of the combinational circuit and scan its responses out during the test.

Test mode

In test mode, the Boolean signal *OM* is set to high. The control part of the circuit shown in Figure 6.11 is reconfigured to provide the desired asynchronous test control interface. All the latches of *Reg1* are set to their initial states whereas all the latches of *Reg3*, all the C-elements and the toggle element are set to zero. The primary inputs (*PI*) of the combinational circuit and the *Sin* input of the circuit are connected to the outputs of the two-phase pseudo-random generator. The responses from the *PO* outputs of the combinational circuit and the *Sout* output of the sequential circuit are compressed by the two-phase signature analyser.

A request signal (*Rin*) from the generator is delayed for long enough for the output data to stabilize on the outputs of the combinational logic block. The data from the *SO* outputs of the combinational circuit is mixed with the output data of *Reg3* in the block of two-input XOR gates. The outputs of the XOR gates come through the multiplexer to the inputs of *Reg2* and are latched in *Reg2*. After receiving an acknowledge signal from *Reg2*, which is steered by the toggle element, the content of *Reg2* is copied into *Reg3*. When the data is captured by *Reg3* it generates an acknowledge signal on its output *Ack*. This signal causes the multiplexer to connect the first (*n-1*) most significant bits of *Reg1* and the scan-in input *Sin* of the circuit to the inputs of *Reg2*. Simultaneously, a request signal is produced for the signature analyser on the *Rout* output. The data from the outputs of the multiplexer is captured by *Reg2* when a new request signal appears on its request input *Req* (in fact, this is the acknowledge signal for *Reg3* which is delayed until the multiplexer has switched). A new acknowledge signal from *Reg2* is steered by the toggle element and passes to the corresponding input of the symmetric C-element where

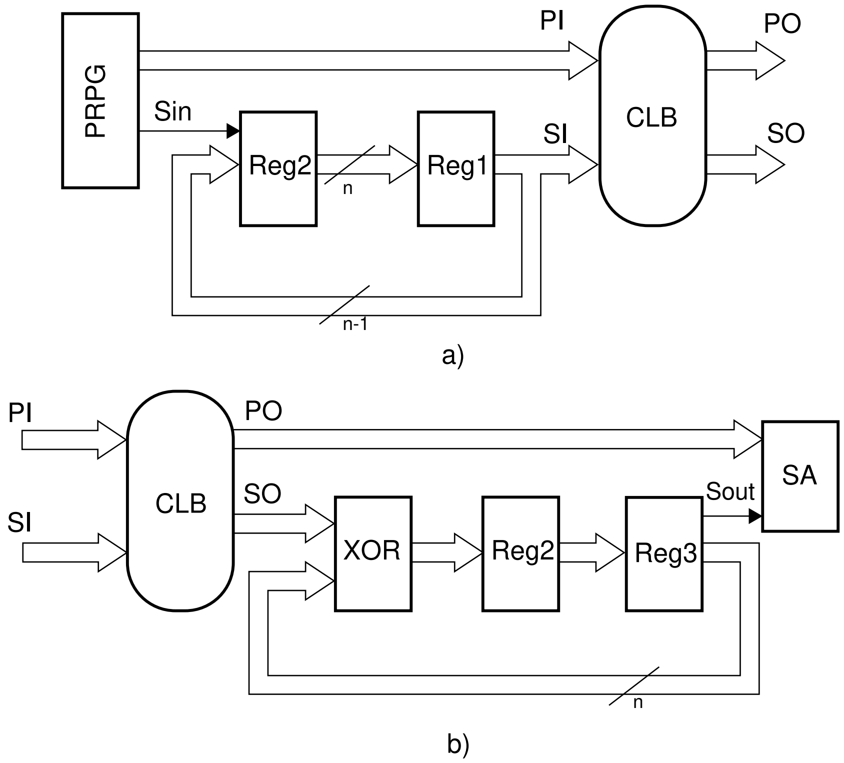


Figure 6.12: *The mechanism for a) applying test patterns to the inputs of the CLB and b) compressing the responses from the outputs of the CLB during the test*

it waits for an acknowledge signal from the signature analyser. The primary outputs of the combinational circuit and the scan-out output of *Reg3*, which is actually the *n*th bit of *Reg3*, are collected by the signature analyser. Once an acknowledge signal is received on the *Aout* input:

1) the content of *Reg2* is copied into *Reg1*;

2) an acknowledge signal is sent to the generator.

When the generator has finished producing a new test pattern, a new request signal is generated on the *Rin* input of the circuit and the test procedure described above is repeated again.

Figure 6.12 illustrates the mechanism for applying random test patterns to the inputs of the combinational circuit and compressing the responses from its outputs. The procedure for applying test patterns (see Figure 6.12a) assumes that random tests are applied

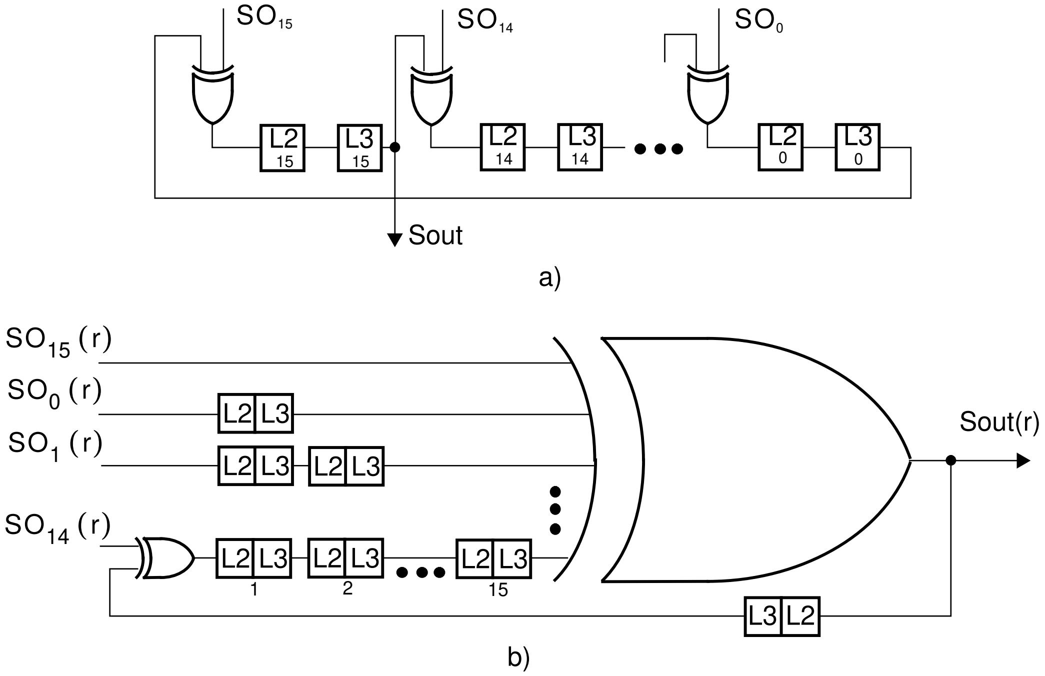


Figure 6.13: Compressing the test data from the internal outputs of the CLB: a) the structure of the signature analyser; b) the equivalent schematic of the signature analyser

both to the *PI* inputs of the combinational circuit and the *Sin* input of the sequential circuit. During the test registers *Reg1* and *Reg2* are configured to shift a new test bit to *Reg1* after receiving a request signal from the generator. As a result, random test bits are shifted in *Reg1* bit-serially from the *Sin* input of the circuit.

The process of collecting and compressing test data from the outputs of the combinational circuit (see Figure 6.12b) consists of two parts. The first one includes the direct analysis of the responses from the *PO* outputs of combinational circuit by means of the external signature analyser. The second part is a signature analyser which compresses the responses from the *SO* outputs of the combinational circuit. Registers *Reg2*, *Reg3* and the block of XOR gates are configured in such way that the current contents of *Reg3* are mixed (with the help of the XOR operation) with a new response which is produced on the *SO* outputs of the combinational circuit. The contents of *Reg3* are observed on its *n*-th output.

The signature analyser used for collecting the test data from the *SO* outputs of the combinational circuit is illustrated in Figure 6.13. The general structure of this signature analyser (see Figure 6.13a) is similar to the structure of the BILBO signature analyser (see Appendix B). The equivalent schematic of such a signature analyser (see Figure 6.13b) shows that the procedure for compressing the test data from the *SO* outputs of the combinational circuit is similar to the XOR operation. After receiving each request signal (*r*) the input bits are delayed for a different number of steps (request signals) depending on their position numbers and then XORed.

Normal operation mode

In normal operation mode, the *OM* input of the sequential circuit is reset. The outputs of the toggle element and the outputs of *Reg3* are held at zero permanently. Initially, all the C-elements are reset. The latches of *Reg1* are set to their initial states. After receiving a request signal at the *Rin* input from the sender data is processed by the sequential circuit in the same way as was described for the circuit shown in Figure 5.9.

6.4.2 Four-phase sequential circuit design

Figure 6.14 shows the design of a four-phase sequential circuit with bit-serial random testing. This circuit performs in two modes depending on the value applied to the Boolean signal *OM*: test (*OM*=1) and normal operation mode (*OM*=0).

Test mode

In test mode the *OM* input is set to high and all C-elements are reset. The latches of *Reg1* are set to their initial states. The latches of *Reg3* are reset. Note that registers *Reg1* and *Reg3* are closed whereas the latches of *Reg2* are transparent initially.

When a test vector has been applied to the *PI* and *Sin* inputs by the four-phase pseudo-random generator the *Rin* signal goes high setting the output of *C1* to high. This signal is delayed for long enough for the output data to stabilize on the *PO* and *SO* outputs of the combinational circuit. The data from the *SO* outputs are mixed with the contents of *Reg3* in the block of XOR gates and sent through the multiplexer (*MX1*) to the inputs of *Reg2* to be stored in its latches. The *Y1* output of the control circuit (*CC*) goes high. This signal is passed through the OR gate to the output of multiplexer *MX2* closing the latches of *Reg2.* As a result, the latches of *Reg3* become transparent since the output of *C4* is set to high by a rising signal produced on the acknowledge output of *Reg2*. A rising event from the *Ack* output of *Reg3* sets the *Y2* output of *CC* to high and the output of *C4* is reset.

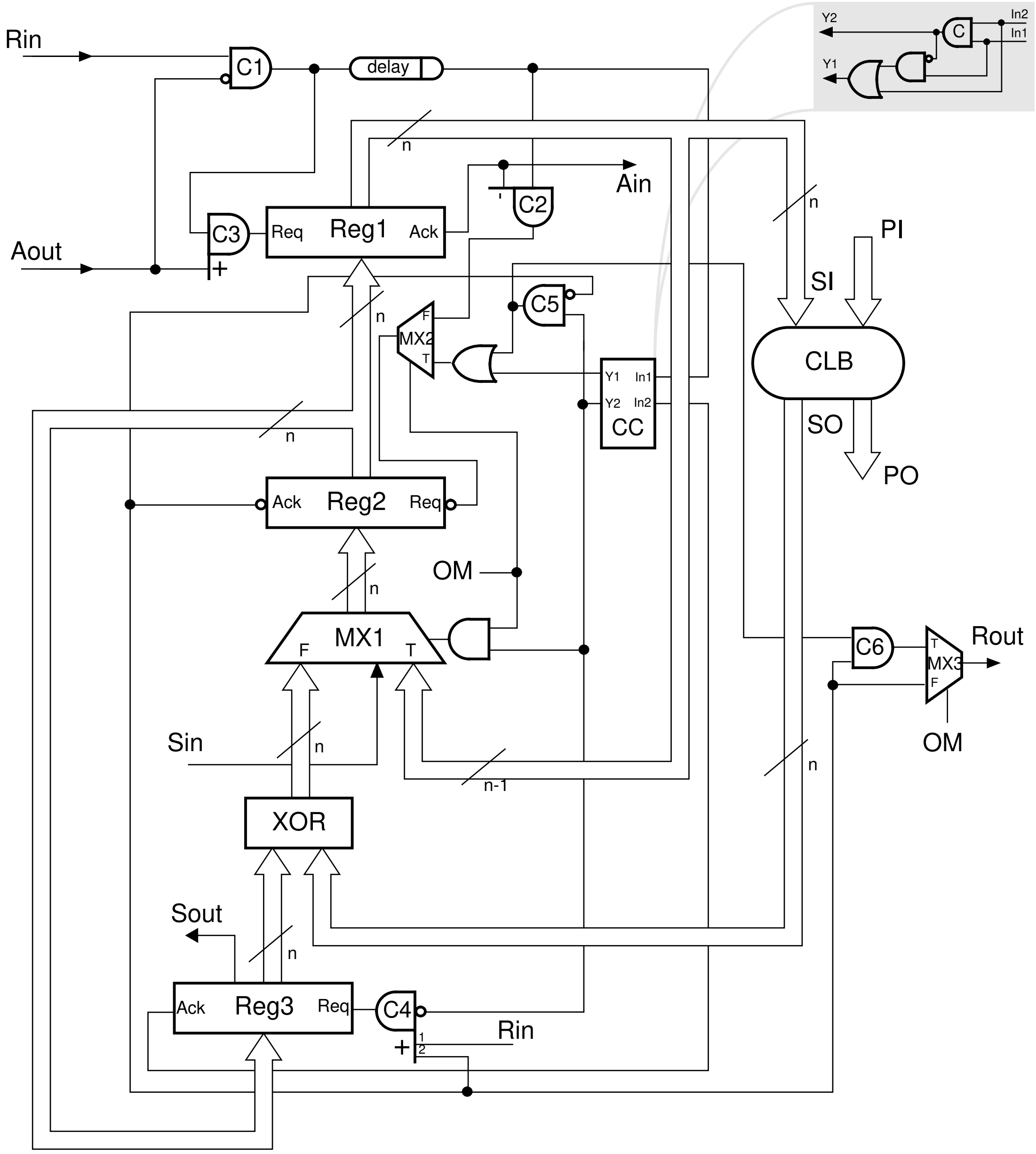


Figure 6.14: *Four-phase sequential circuit with bit-serial random testing*

The latches of *Reg3* become opaque storing in their memory a current signature of the data produced on the *SO* outputs of the combinational circuit. The mechanism for collecting the data form the *SO* outputs was discussed in the previous section of this chapter.

Note that a high signal from the *Y2* output of *CC* switches multiplexer *MX1* so that it connects the (*n*-1) less significant outputs of *Reg1* and the *Sin* input to the inputs of *Reg2*. A falling signal from the *Ack* output of *Reg3* resets the *Y1* output of *CC* making the latches of *Reg2* transparent. Afterwards a falling signal from the *Ack* output of *Reg2* sets the output of *C5* to high. As a consequence, the latches of *Reg2* become opaque again. Once the *Ack* output of *Reg2* is set to high the state of *C6* is changed to one producing a rising event on the *Rout* output of multiplexer *MX3*.

When the data form the *PO* outputs and the *Sout* output has been collected by the external signature analyser the *Aout* signal goes high. The output of *C3* is set to high making the latches of *Reg1* transparent. A rising signal from the *Ack* output of *Reg1* is passed to the *Ain* output of the circuit. Once the *Rin* signal is returned to zero the output of *C1* is reset changing the state of *C3* to zero. As a result, *Reg1* is closed storing a new test vector which is applied to the *SI* inputs of the combinational circuit. The mechanism for producing tests for the *SI* inputs of the combinational circuit is similar to that described in the previous section of this chapter.

The *Y2* output of *CC* is reset changing the state of *C5* to zero. The latches of *Reg2* become transparent and a falling event is generated on the *Ack* output of *Reg2*. Thus, *C6* moves to the zero state resetting the *Rout* output of the circuit. If the *Aout* input is reset the sequential circuit is ready to accept a new test from the generator and the sequence of events described above is repeated.

Normal operation mode

When the Boolean signal *OM* is low the sequential circuit performs in normal operation mode. In the initial state all the C-elements are reset. The latches of *Reg1* are set to their initial states. The output of the symmetric C-element of *CC* is kept high (see Figure 6.14) and the outputs of *Reg3* are kept low during normal operation mode. When the data is ready on the *PI* inputs and the *Rin* input is set to high the sequential circuit operates in the same manner as the circuit shown in Figure 6.7.

6.4.3 Analysis of the bit-serial random test technique

Advantages

The random pattern testable sequential circuits shown in Figures 6.11 and 6.14 have some important features which simplify their random testing.

Complexity of the test procedure. During the test the combinational part of the sequential circuit is tested separately from the memory elements which makes the testing of the circuit much easier.

Test performance. Compared to the scan test approach the bit-serial test procedure does not require a test pattern to be scanned into the shift register before the testing and the test data to be scanned out after the application of the test pattern. During the random testing of the sequential circuits (see Figures 6.11 and 6.14), test patterns are produced on the *SI* inputs of their combinational circuits with the help of shifting the contents of *Reg1* by one bit. A new test bit is loaded from the generator after receiving a request signal. The test data from the *SO* outputs of the combinational circuit is collected in *Reg3* after the application of each new test pattern to the inputs of the combinational circuit. There is no need to shift all the contents out of *Reg3* after applying a new test pattern to the inputs of the circuit (the test data is compressed and stored into register *Reg3* and observed on its *n*-th output after the application of each test pattern). In this case the random pattern testing of such a circuit is approximately (*n-1*) times faster then a traditional scan test method, where *n* is the number of latches of *Reg1*.

Number of random test patterns. The analysis of the circuits illustrated in Figures 6.11 and 6.14 shows that the number of random test patterns required to detect all their single stuck-at faults in the data paths is equal to the number of test patterns for detecting all the stuck-at faults in their combinational parts. This is because of the following properties:

- all the stuck-at faults on the inputs/outputs of registers *Reg1* and *Reg2* are equivalent to the corresponding faults on the *SI* inputs of the combinational logic block;
- all the stuck-at faults on the inputs/outputs of the block of XOR gates and *Reg3* are detected easily during the test of the combinational circuit (the circuitry which collects the test data from the internal outputs of the combinational circuit (see Figure 6.13) is similar to the BILBO register);

The use of either equiprobable or weighted random test patterns. The bit-serial random testing allows either equiprobable or weighted random test vectors to be applied to the inputs of the combinational circuit during its testing.

Random testing using only equiprobable random test patterns is not always the optimal test procedure for obtaining the minimal (or close to minimal) number of random test patterns in order to guarantee the detection of all the circuit's stuck-at faults. In order to reduce the number of random test patterns, special methods have been derived for achieving optimal output signal probabilities for generators of weighted pseudo-random test patterns [Agra76, ChinTR84, Waic89]. The upper bound for the random test length L can be calculated using formula (6.2).

Disadvantages

Complex circuit control. Two-phase and four-phase implementations of the bit-serial random testing technique require the initial control circuit of the corresponding sequential circuit to be changed. The introduction of extra elements into the control circuit can make its testing more difficult.

A fault simulation analysis of the two-phase and four-phase control circuits was carried out using *SIMIC* design verification tools [Sim94]. It was observed that stuck-at faults on the control lines of the two-phase sequential circuit (see Figure 6.11) are easy to detect since either they cause the circuit to halt or they change the data flow during the test which can be identified easily. There are some stuck-at faults in the two-phase control circuit (on the false inputs of the control multiplexers) which cannot be detected in test mode. They will manifest themselves during normal operation mode by preventing any activity on the *Rout* and *Ain* outputs, hence causing the whole circuit to deadlock.

In the four-phase sequential circuit shown in Figure 6.14 most stuck-at faults on the control lines manifest themselves during the test by causing the circuit to halt. A stuck-at fault on the control input of *MX1* can be detected easily since it changes the data flow during the test. Note that stuck-at faults on the control wires which are not involved in test mode can only be identified during normal operation mode by causing the circuit to halt.

The stuck-at faults which cause premature firings on the outputs of *C2* and *C3* can be tested in the same manner described in section 6.3.2. Faults 1-SA1 and 2-SA1 cause premature rising events on the output of *C4*. These faults can be detected by converting the asymmetric C-element into a symmetric one using an additional control input (*Tst*) as shown in Figure 5.13. After the initialization the following sequence of steps can be used to identify these faults:

1. *OM*=1.
2. *Tst* is set to high. *C4* operates as a symmetric C-element.
3. *Rin*↑

   The *Ack* output of *Reg2* goes high setting the output of *C4* to high. As a result, the *Y2* output of *CC* goes high.

3. *Aout*↑ and *Rin*↓

   The output of *C2* is reset.

4. When the *Ain* output has reset *OM*=0.

   The Multiplexer *MX2* is switched connecting the output of *C2* to the *Req* input of *Reg2*. Thus, *C4* goes to the zero state if it is fault-free and the output of *C4* remains unchanged in the presence of faults 1-SA1 and 2-SA1. Note that the output of *C5* is set to high.

5. *OM*=1.

   Fault-free behaviour. The outputs of *CC* are reset. The *Ack* output of *Reg2* goes high changing the state of *C6* to one. Afterwards, the output of *C5* is reset and the output of *C6* is set to low. As a result, the *Rout* output goes high and then low.

   Faulty behaviour. The outputs of *CC* remain unchanged (*Y1*=*Y2*=1). The output of *C6* is set to high and the *Rout* signal goes high.

The operation mode of the C-element *C4* (symmetric or asymmetric) can be checked by setting the sequential circuit in test mode and applying the sequence of events described in section 6.4.2. If *C4* operates as the symmetric C-element the whole circuit will halt.

Hardware redundancy. The hardware redundancy of the testable sequential circuits consists of register *Reg3*, the block of *n* XOR gates, three multiplexers, the toggle element and extra control elements to provide for the corresponding signalling. Clearly, the overall hardware redundancy of the testable circuit heavily depends on the complexity of its combinational logic block: the more complex the combinational circuit is, the less redundancy the testable sequential circuit has. As a result, this method of designing testable asynchronous sequential circuits is more effective in terms of hardware redundancy for complex sequential circuits.

Performance degradation. There is some degradation in the performance of the testable sequential circuits during normal operation mode. This is caused by the additional circuits incorporated in the data paths and the control circuit which inevitably slows down the circuit performance.

6.5 Handshake implementations of a sequential circuit for random pattern testability

6.5.1 The design of a handshake sequential circuit

A sequential circuit can be implemented as the handshake circuit shown in Figure 6.15. This circuit has one input communication channel (*pi*) along which *m*-bit input data is sent and one output channel (*po*) from which *k*-bit output data is read by the environment. The sequential circuit comprises two storage elements (*R1* and *R2*) to store the states of the circuit and the combinational block (CL) which performs the logic function producing the outputs and new states for the sequential circuit.

Initially, the *R2* storage element is reset into its initial state according to the specification of the sequential circuit. The handshake circuit is activated along its activation channel (⇨). As a result, the contents of the *R2* storage element is copied into *R1*. Afterwards, the passive port of the combine element (<<>>) is triggered by the corresponding sequencer.

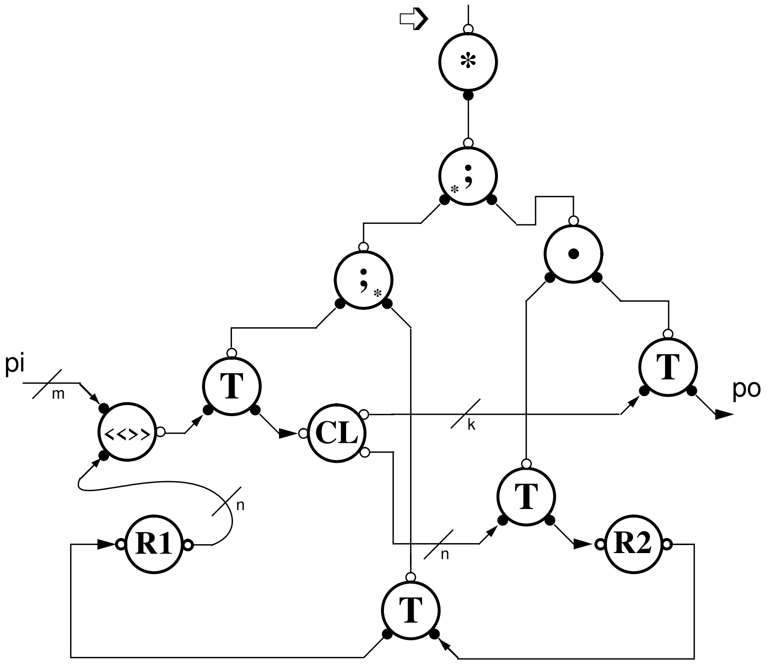


Figure 6.15: *Handshake implementation of a sequential circuit*

An implementation of the combine element is shown in Figure 6.16a [FarnTR96]. The combine element is activated along its channel *a* which is the passive channel of the fork (the design of the fork is illustrated in Figure 6.16b). The fork passes a request signal from channel *a* to the request outputs of channel *a* and *b* in parallel. When the data has ready on the data inputs of channels *b* and *c* rising acknowledge signals are sent to the fork setting the output of the symmetric C-element to high (see Figure 6.16b). When the data has been read from the data outputs of the combine element the request wire of channel *a* is returned to zero resetting the corresponding request wires of channels *b* and *c*. A handshake along input and output channels of the combine element is completed when the acknowledge signals of channels *b* and *c* are returned to zero.

The combine element reads the data from the outputs of *R1* and inputs of the *pi* channel. The transferer sends the data from the outputs of the combine element to the inputs of the combinational circuit. A handshake implementation of the combinational circuit is illustrated in Figure 6.16c. When the data has arrived at the inputs of the combinational circuit a_r goes high. This signal is delayed for long enough for the output data to stabilize on the outputs *b* and *c*. Note that the outputs *b* and *c* of the combinational circuit correspond to the primary and state outputs of the circuit respectively. A handshake along the input channel is completed when the a_r and a_a signals are returned to zero. The output data produced by the combinational circuit is read by activating the corresponding output channels.

When the data is ready to be read from the outputs of the combinational circuit the top sequencer triggers the fork which activates the corresponding transferers (see Figure 6.15). The data from the primary and the state outputs of the combinational circuit is transmitted in parallel to the *po* channel and the *R2* storage element respectively. When the output data is stored in *R2* and read by the environment the contents of *R2* is copied to *R1*. As a result, the sequential circuit generates a request for new data to be sent to its *pi* channel.

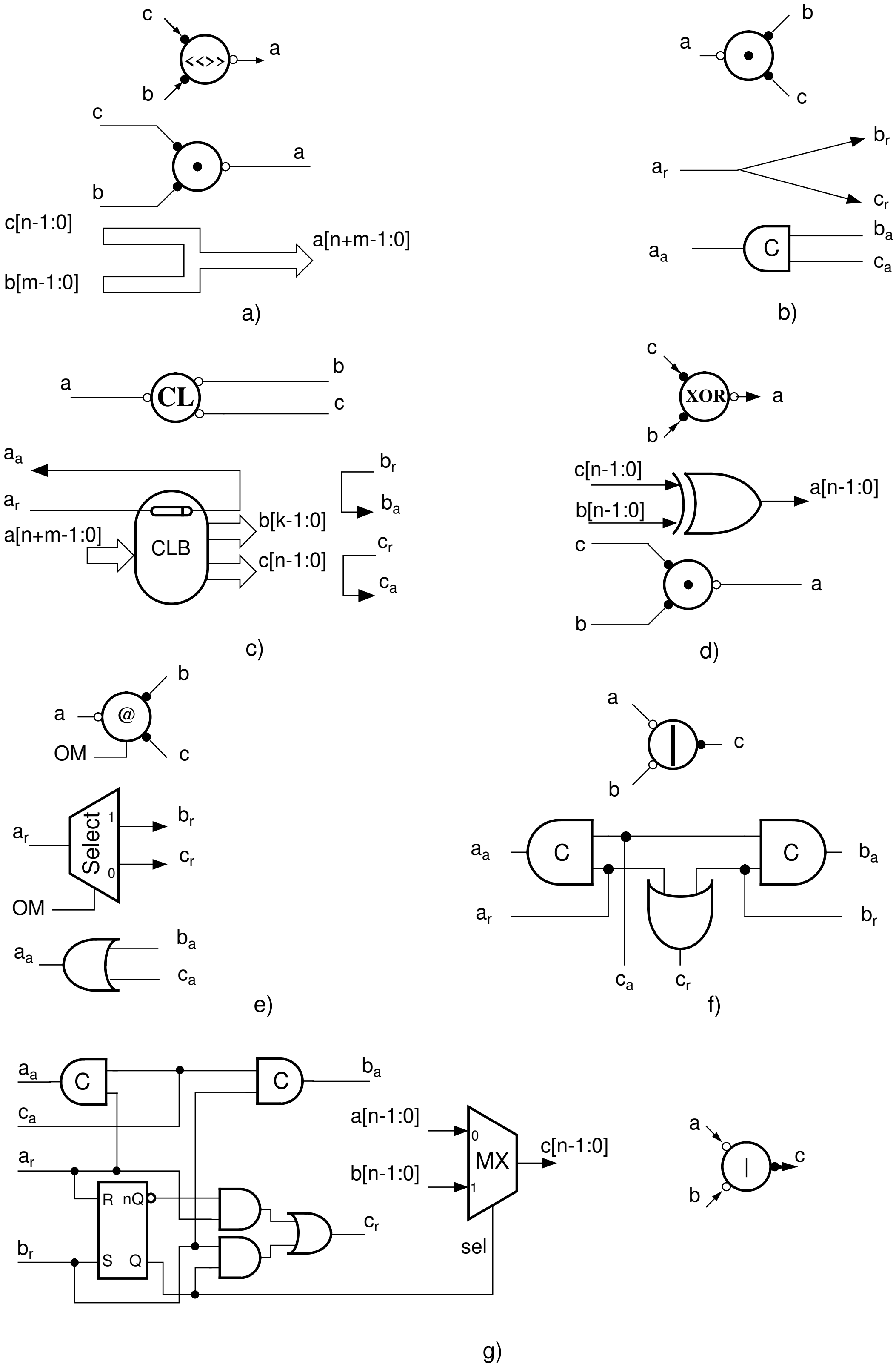


__Figure 6.16:__ Handshake implementations of the a) combine element; b) fork; c) combinational circuit; d) bitwise XOR operation; e) case element; f) mixer; g) multiplexer

```
SC: proc(pi?(0 .. $2^m - 1$) & po!(0 .. $2^k - 1$ ))
begin
        input: var (0 .. $2^m - 1$)
        r1, r2: var (0 .. $2^n - 1$)
        ResetState: proc() r2:=0
|
        ResetState();
        forever do
                r1:=r2;
                pi?input;
                r2:=CombFunction1(input,r1) || po!CombFunction2(input,r1)
        od
end
```

Figure 6.17: *Procedure SC performed by the sequential circuit shown in Figure 6.15*

Figure 6.17 shows an example of the Tangram procedure *SC* performed by the sequential circuit illustrated in Figure 6.15. According to this procedure the input data is assumed to be stored by the environment in a variable called *input*. Initially, the variable *r2* is reset by the *ResetState* procedure. Afterwards, the set of commands written between the ***forever do*** and ***od*** lines is performed infinitely. Thus, the contents of variable *r2* is copied into *r1* and new data is read from *input* along the *pi* channel. A new state vector produced by function *CombFunction1* is written in *r2* and the output data produced by function *CombFunction2* is transmitted from the primary outputs of the combinational circuit to the *po* channel concurrently. Note that the function of the combinational circuit shown in Figure 6.16c is the union of combinational functions *CombFunction1* and *CombFunction2*.

Fault model

In this section two single stuck-at fault models are considered in the design of the handshake sequential circuit:

- the stuck-at output faults on the control lines of its handshake components;
- the stuck-at input and output faults in the circuit data paths.

As was described in chapter 2 the stuck-at output faults on the control lines of a handshake circuit cause this circuit to halt. The detection of these faults is straightforward. In order to detect all possible single stuck-at faults in the data paths of the sequential circuit the parallel and bit-serial random test techniques can be used.

6.5.2 Parallel random testing

Figure 6.18 shows the design of a handshake sequential circuit with parallel random testing. This circuit has an additional block of XOR gates and the test channel (*tc*) which is activated in test mode.

Test mode

Initially the *r2* storage element is set into its initial state. The sequential circuit is activated along its activation channel. As a result, the content of the *r2* storage element is copied into *r1*. When the data has read from the *pi* channel and from the outputs of *r1* the outputs of the combinational circuit are changed. Afterwards, the output data from

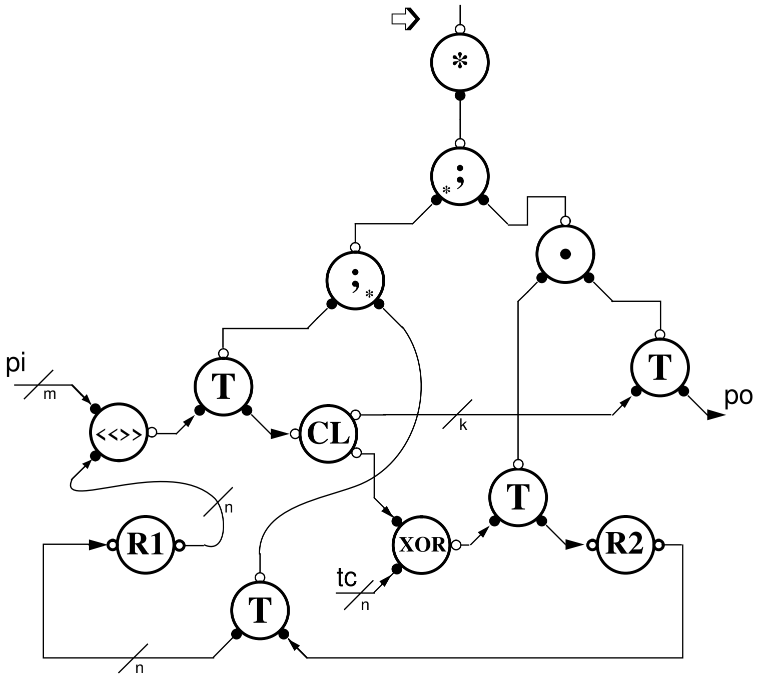


Figure 6.18: *Handshake sequential circuit with parallel random testing*

the primary outputs of the combinational circuit is read along the *po* channel and the block of XOR gates is triggered by the corresponding transferer.

A handshake implementation of the block of XOR gates is illustrated in Figure 6.16d [FarnTR96]. If the passive channel *a* of the block is activated the corresponding channels *b* and *c* generate requests for the input data. When the data has arrived at the data wires of channels *b* and *c* the block of XOR gates performs a bitwise XOR operation and produces an acknowledge signal along its passive channel *a*.

Thus, the block of XOR gate reads the data from the *SO* outputs of the combinational circuit and a new random pattern test vector is produced on the data outputs of the *tc* channel. The results of the bitwise XOR operation are stored into the *r2* storage element. When the data from the *PO* outputs of the combinational circuit has been read along the *po* channel and the data from the outputs of the block of XOR gates has been stored in *r2* the sequential circuit copies the contents of *r2* into *r1* and the sequence of events described above can be repeated.

Thus, the data paths of the handshake sequential circuit shown in Figure 6.18 are tested in the same manner as was described in section 6.3.

```
SC_PRT: proc(pi?(0 .. $2^m - 1$) & tc?(0 .. $2^n - 1$) & po!(0 .. $2^k - 1$))
begin
        input: var (0 .. $2^m - 1$)
        r1, r2, test: var (0 .. $2^n - 1$)
        ResetState: proc() r2:=0
|
        ResetState();
        forever do
                r1:=r2;
                pi?input;
                tc?test; r2:=CombFunction1(input,r1)$\oplus$ test ||
                        po!CombFunction2(input,r1)
        od
end
```

Figure 6.19: *Parallel random testing procedure SC_PRT*

Normal operation mode

In normal operation mode the data wires of the *tc* channel are kept at zero permanently. As a result, the sequential circuit operates in the same manner as the one illustrated in Figure 6.15.

Figure 6.19 shows an example of the Tangram program for the parallel random testing procedure *SC_PRT* performed by the handshake sequential circuit illustrated in Figure 6.18. Compared to the *SC* procedure the program *SC_PRT* has an extra variable called *test*. The variables *input* and *test* are changed by the environment simultaneously. These variables can be produced by using either one pseudo-random pattern generator or two separate generators. In normal operation mode the *test* variable is reset permanently so that the *SC_PRT* procedure is equivalent to the *SC* program.

6.5.3 Bit-serial random testing

Figure 6.20 illustrates the design of a handshake sequential circuit which is tested using the bit-serial random testing. This circuit performs in test and normal operation mode depending on the Boolean signal *OM*.

The *OM* input of the sequential circuit controls the case element. An implementation of the case element is shown in Figure 6.16e [FarnTR96]. If *OM*=1 or *OM*=0 a request signal from the a_r input is steered by the select block to the b_r output or c_r output respectively. Acknowledge signals from the b_a and c_a inputs are passed through the OR gate to the a_a output.

Test mode

In test mode the *OM* input is set to high. The storage element *R2* is set to its initial state. The storage element *R3* is reset. When the sequential circuit shown in Figure 6.20 has been activated along its activation channel the contents of *R2* is copied into *R1*. After the data has read along the *pi* channel and from the outputs of *R1* the upper active channel of the case element is triggered by the corresponding sequencer. As a result, the request signal from the case element is steered by the corresponding pair of sequencers through the mixer to the transferer.

An implementation of the mixer is illustrated in Figure 6.16f [FarnTR96]. The mixer has two passive (*a* and *b*) and one active (*c*) channels. When a rising event is generated on one of the passive channels of the mixer the c_r signal goes high. Once acknowledged by a rising event on the c_a input the output of the corresponding C-element is set to high. The sequence of events is repeated when the control signals are returned to zero. Note that both the passive channels of the mixer cannot be activated simultaneously.

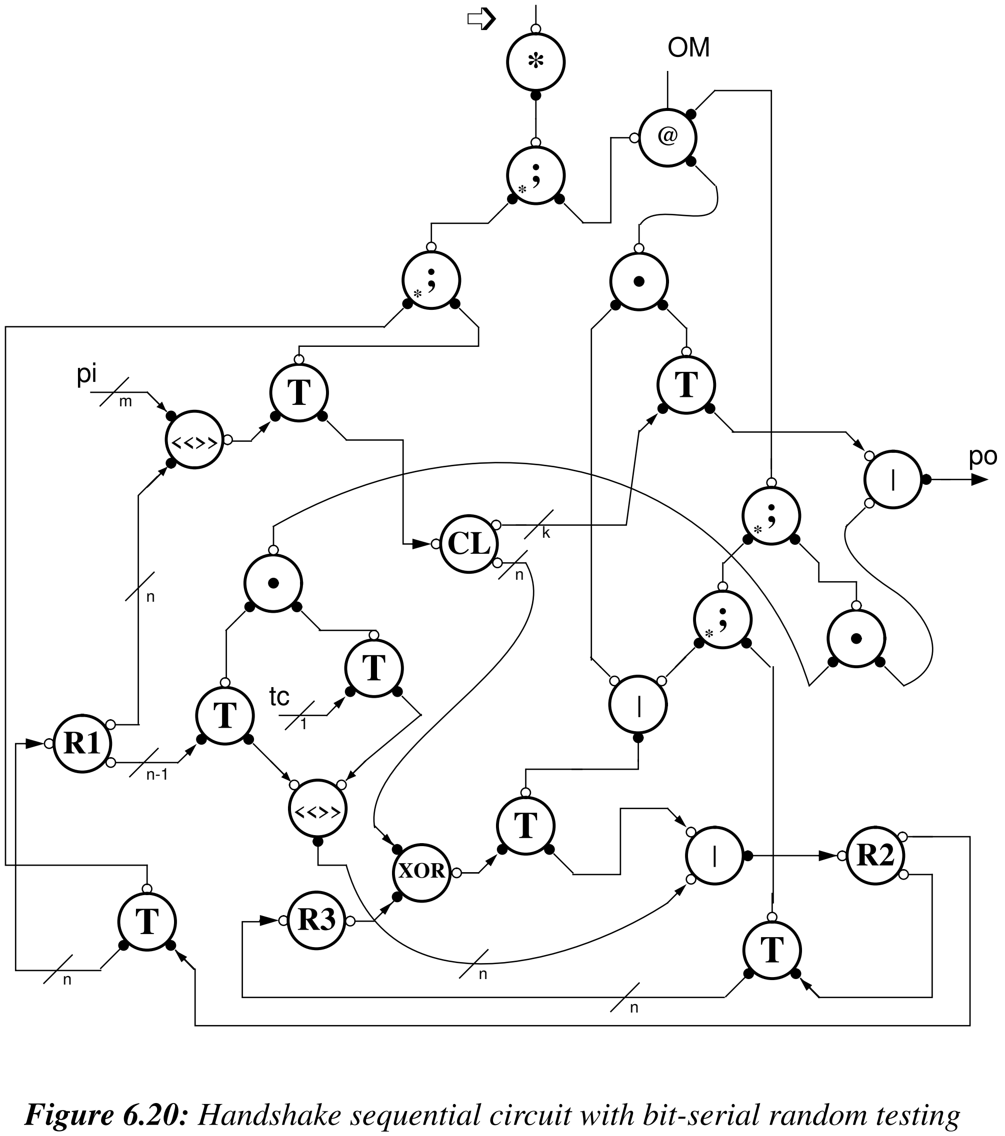


Figure 6.20: *Handshake sequential circuit with bit-serial random testing*

The results of the bitwise XOR operation between the data from the *SO* outputs of the combinational circuit and the contents of *R3* are passed through the multiplexer and copied into *R2*.

The design of the multiplexer is illustrated in Figure 6.16g. The multiplexer operates in a similar way to the mixer shown in Figure 6.16f [FarnTR96]. The difference between them is in the presence of an additional circuit which controls the multiplexer. When the data is stable on bus *a* channel *a* is activated. Thus, the RS flip-flop is reset and the *sel* signal is set to low connecting bus *a* to the outputs of the multiplexer. If channel *b* is activated the *sel* signal is set to high and the data from bus *b* is transmitted to the outputs of the multiplexer.

The contents of *R2* are copied to the storage element *R3* which collects the current signature of the responses produced on the *SO* outputs of the combinational circuit. When the data has been stored into *R3* the *po* channel is activated along which the test results from the *PO* outputs of the combinational circuit and the most significant bit of *R3* are read by the external signature analyser. Concurrently, a new random test vector is stored into *R2* as a result of combining the (*n*-1) most significant bits of *R1* and one bit which is read along the test channel (*tc*).

When a handshake along the *po* channel has completed and the new test vector has been stored in *R2* the contents of *R2* is copied into *R1* and the *pi* channel is triggered ready to accept a new random test pattern. Thus, the random test procedure described above can be repeated.

Figure 6.21 shows an example of the Tangram program for the bit-serial random testing procedure *SC_BST* performed by the sequential circuit illustrated in Figure 6.20. In this program variables *r1* and *r3* are used to store new tests applied to the *SI* inputs and collect current signatures from the *SO* outputs of the combinational circuit. The 1-bit variable *test* is used by the *tc* channel to shift a new test bit in *r1*.

In the *SC_BST* procedure the variables *r2* and *r3* are reset after the initialization of the circuit. The content of *r2* is copied into *r1*. A new random test pattern is read from the

```
SC_BST: proc(pi?(0 .. $2^m - 1$) & tc?(0 .. 1) & po!(0 .. $2^k - 1$))
begin
    input: var (0 .. $2^m - 1$)
    r1, r2, r3: var (0 .. $2^n - 1$)
    test: var (0..1)
    ResetStates: proc() r2:=0; r3:=0
|
    ResetStates();
    forever do
        r1:=r2;
        pi?input;
        r2:=CombFunction1(input,r1);
        r3:=Signature(r2,r3);
        tc?test;
        r2:=<<test.r1.n-1. .. r1.1>> //
            po!<<CombFunction2(input,r1).r3.n-1>>
    od
end
```

Figure 6.21: *Bit-serial random testing procedure SC_BST*

input variable along the *pi* channel. A new state of the circuit is produced by function *CombFunction1* and stored into *r2*. The result of function *Signature* is stored in *r3* (function *Signature* is the same function which is performed by the circuit shown in Figure 6.13). Afterwards, the content of variable *test* is read along the *tc* channel. The (n-1) most significant bits of variable *r1* with the content of the *test* variable is stored in *r2*. Simultaneously, the results of function *CombFunction2* and the most significant bit of *r3* are sent along the *po* channel to the external signature analyser. After this the *SC_BST* procedure is repeated.

The data paths of the circuit shown in Figure 6.20 are tested in the same way as was described in section 6.4. Stuck-at output faults on the control lines of the handshake sequential circuit which are not involved in test mode are not tested. These faults can be tested when the circuit performs in normal operation mode.

Normal operation mode

The Boolean *OM* is reset in normal operation mode allowing the lower active channel of the case element to be controlled from its passive channel. The outputs of the storage element *R3* are held at zero permanently. As result, the handshake sequential circuit shown in Figure 6.20 performs in the same manner as the one described in Figure 6.15.

6.6 A case study of the AMULET2e memory controller

An extended version of the AMULET2 microprocessor called AMULET2e has been implemented to connect the microprocessor to a wide variety of static and dynamic memories and peripheral chips. Figure 6.22 shows a block diagram of the AMULET2e microprocessor which incorporates the AMULET2 processor core together with an on-chip cache and an external memory interface.

The memory interface of the AMULET2e microprocessor includes a memory controller. The memory controller is a finite state machine which has seven inputs (m=7), four outputs (k=4) and 3-bit coded states (n=3). The state table for the AMULET2e memory controller is shown in Table D.1 in Appendix D.

Two-phase, four-phase and handshake designs of the memory controller have been implemented in CMOS technology on a $1\mu m$ double layer metal process and simulated

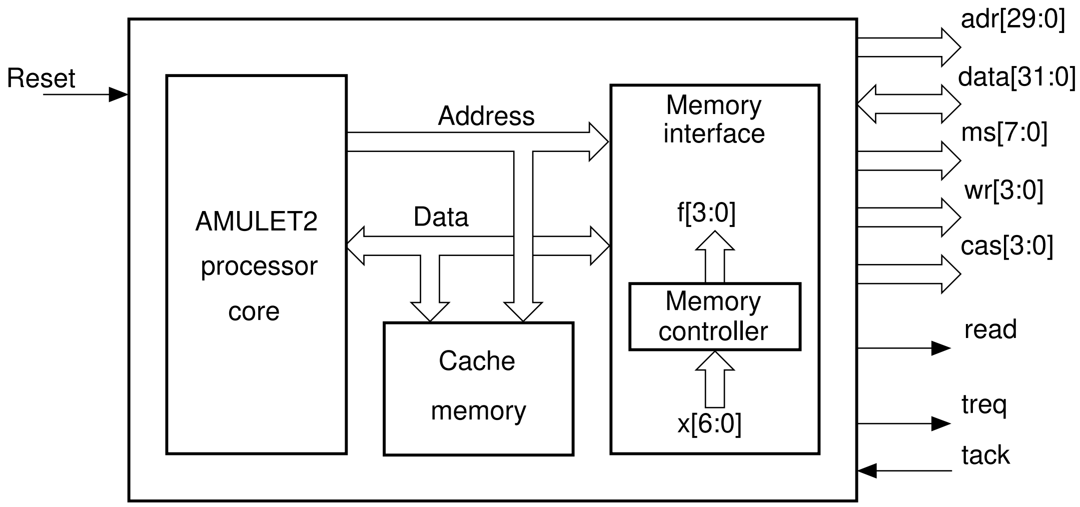


Figure 6.22: *Block diagram of the AMULET2e microprocessor*

using *SIMIC* design verification tools. Simulation results of the three versions of the memory controller, i.e. the controller without testability features, with bit-serial and parallel random testing, implemented as two-phase, four-phase and handshake sequential circuits can be found in Tables 6.2, 6.3 and 6.4 respectively. The notation used in these tables is as follows:

> PD is the performance degradation, AO is the area overhead, NM and TM is normal operation mode and test mode respectively.

Table 6.2: *Two-phase implementations of the AMULET2e memory controller*

Design	Performance, ns/test		PD, %	Area, $\times 10^{-2} mm^2$	AO, %	Power consumption, nJ/test	
	NM	TM				NM	TM
Without testability	16.8	n/a	n/a	10.6	n/a	48.1	n/a
Bit-serial	18.8	33.5	12%	13.4	26%	54.2	73.1
Parallel	17.9	17.9	7%	10.9	3%	50.3	59.4

Table 6.3: *Four-phase implementations of the AMULET2e memory controller*

Design	Performance, ns/test		PD, %	Area, $\times 10^{-2} mm^2$	AO, %	Power consumption, nJ/test	
	NM	TM				NM	TM
Without testability	14.7	n/a	n/a	9.9	n/a	50.6	n/a
Bit-serial	17.1	30.4	15%	12.1	22%	63.8	88.2
Parallel	15.6	15.6	6%	10.2	3%	52.9	60.2

Table 6.4: *Handshake implementations of the AMULET2e memory controller*

Design	Performance, ns/test		PD, %	Area, $\times 10^{-2} mm^2$	AO, %	Power consumption, nJ/test	
	NM	TM				NM	TM
Without testability	18.1	n/a	n/a	10.0	n/a	53.9	n/a
Bit-serial	22.5	38.7	24%	12.9	29%	69.0	94.2
Parallel	19.3	19.3	7%	10.3	3%	56.5	64.2

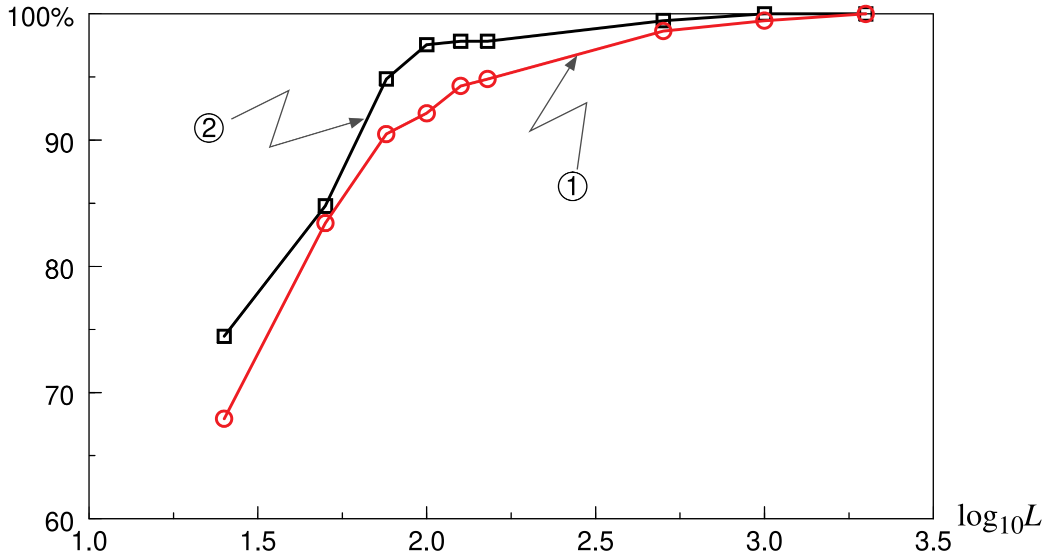


Figure 6.23: *Graph dependencies between the percentage of detected faults and the number of random tests applied to the inputs of the memory controller without testability features (graph 1) and the one designed for testability (graph 2)*

Comparing different implementations of the memory controller the bit-serial DFT implementations exhibit the highest performance degradation in normal operation mode which varies from 12% for the two-phase design to 24% for the handshake implementation. This is because the control circuits used in the bit-serial DFT designs of the memory controller have to be changed by adding extra control elements which slow them down. As a result, the bit-serial DFT designs consume more switching energy demonstrating the largest power consumption.

The parallel DFT designs of the memory controller have the minimal area overhead (3%) contributed by the block of three XOR gates. Note that the data for the parallel DFT designs does not include the area overhead of the internal pseudo-random generator and signature analyser together with extra test wires required for stimulating the inputs of the XOR gates and observing the test results from the their outputs. Thus, the total area overhead of the parallel DFT designs can be much larger.

It can be seen from Tables 6.2, 6.3 and 6.4 the designs of the memory controller with testability features dissipate more energy than the implementations without testability (see chapter 3).

Finally, the four-phase designs of the memory controller exhibit the best performance in both test and normal operation modes whereas the implementations of the two-phase designs exhibit the lowest performance degradation.

Figure 6.23 shows graphs 1 and 2 which demonstrate the dependencies between the percentage of detected faults in the combinational circuit of the memory controller and the number of random test patterns applied to the inputs of the controller without testability features and its parallel and bit-serial DFT designs respectively. The fault simulation analysis was carried out with the help of *SIMIC* design verification tools. Note that the number of tests in Figure 6.23 is represented in a logarithmical scale on the base 10. As follows from Figure 6.23 the percentage of detected faults in the memory controller and, hence, its random pattern testability is higher in the second case (graph 1 lies below graph 2).

In order to calculate the number of equiprobable random tests required for testing the combinational circuit of the memory controller formula (6.2) can be used. A fault simulation analysis carried out with the help of the *SIMIC* fault simulator showed that there are 8 stuck-at faults (r=8) in the combinational part of the controller which have the minimal detection probability ($p_d = 1.95 \times 10^{-3}$). As a result, the upper bound of the random pattern test length is equal to L=2243 and L=3422 for p_t=0.9 and p_t=0.99 respectively. According to Figure 6.23 all the stuck-at faults in the combinational circuit of the controller are detected after the application of 2000 random tests and 1000 random tests for case 1 and 2 respectively.

6.7 Built-in self-testing of micropipelines

In this section the BIST design of a micropipeline based on the BILBO technique (see Appendix B) is considered.

6.7.1 Asynchronous BILBO register design

The CMOS implementation of one of the latches from which an asynchronous BILBO register is built is illustrated in Figure 6.24. 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 5.3. 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 opened by a low signal applied to its *En* input. When the data is stable on the *Din* input of L_1 the *En* input is set to 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

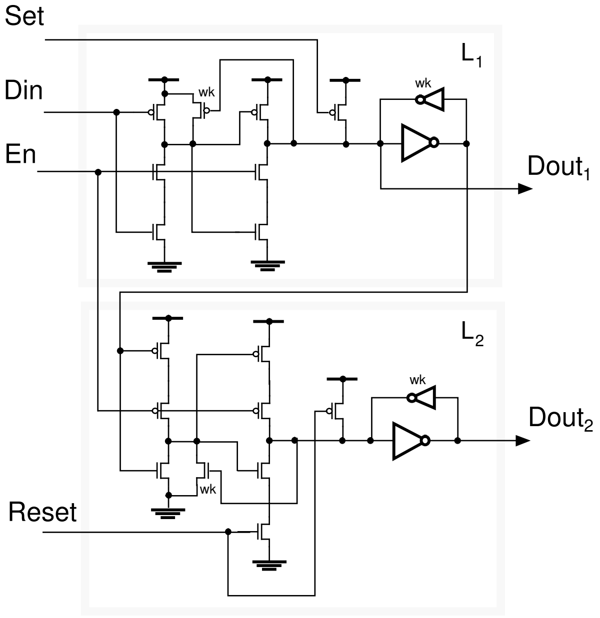


Figure 6.24: *CMOS implementation of the BILBO register latch*

$Dout_1$ 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 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 to 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 6.25. 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 6.24) 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 of 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* clocks are generated by the latch control circuit which can operate using either two-phase or four phase signalling (see chapter 5).

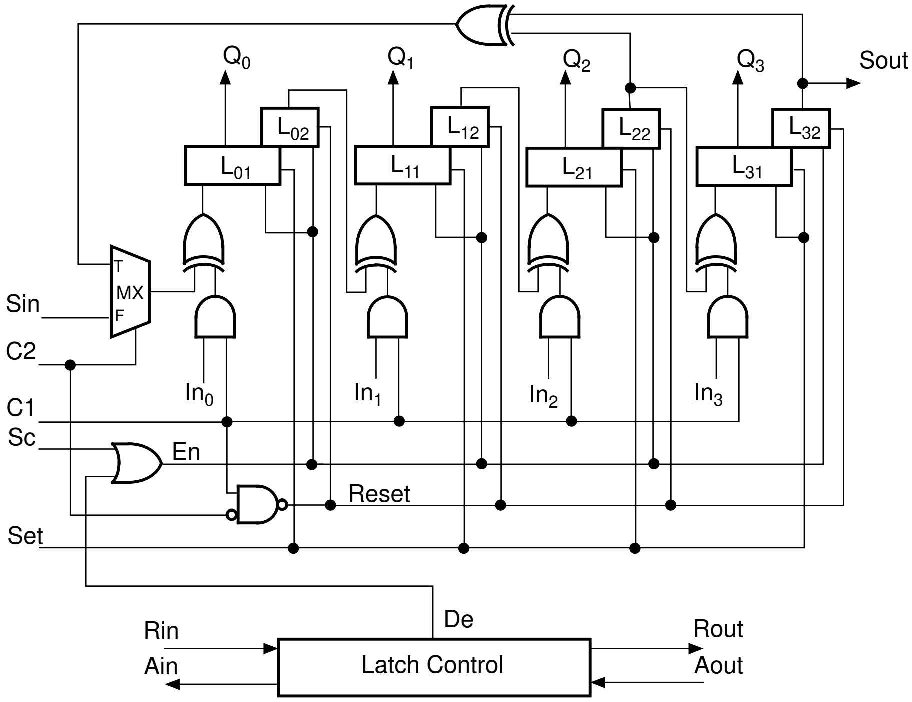


Figure 6.25: *Asynchronous BILBO register structure*

***Table 6.5:** Operation modes of the BILBO register in Figure 6.25*

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

The BILBO register shown in Figure 6.25 can act in four distinct operation modes depending on the control signals *C1* and *C2*: normal, shift, LFSR and signature analyser operation modes. Table 6.5 contains the values for the control signals and the correspond 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 to low holding the outputs of 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 is shifted into the register latches in the same manner as a normal shift register operates. 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 to 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 6.25) can perform as the 4-bit LFSR illustrated in Figure A.2a.

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) to high. When the register outputs are stable the *set* input is set to high. As a result, the ‘all ones’ state is the initial state of the LFSR. The LFSR generates a new output vector every time when a new clock signal is produced by the latch control block on its *De* output.

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 to 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. As a result, the BILBO register is configured as the 4-bit signature analyser shown in Figure A.4.

In the initial state the outputs of the signature analyser are set to 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 to high. Once the data are 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 is mixed with the new input data in the same manner described in Appendix A.

The design of the asynchronous BILBO register shown in Figure 6.25 is similar to the one considered in Appendix B. As a consequence, the testability properties of the asynchronous BILBO register towards its stuck-at faults is the same as that of the synchronous one. Note that all stuck-at faults of 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.

6.7.2 Micropipeline structure with BIST features

Figure 6.26 shows a two-stage micropipeline structure with BIST features. The stage registers of this micropipeline are built from the asynchronous BILBO register illustrated in Figure 6.25. The outputs of the micropipeline can be connected to its inputs depending 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 used in an asynchronous interface to shift the data in and out of the stage registers. These signals have the same meaning as the ones described in chapter 5.

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 to high and low respectively. Its *Set* and *Sc* outputs are set high and low respectively. In this mode the micropipeline acts in the same manner described in chapter 1, section 1.4.

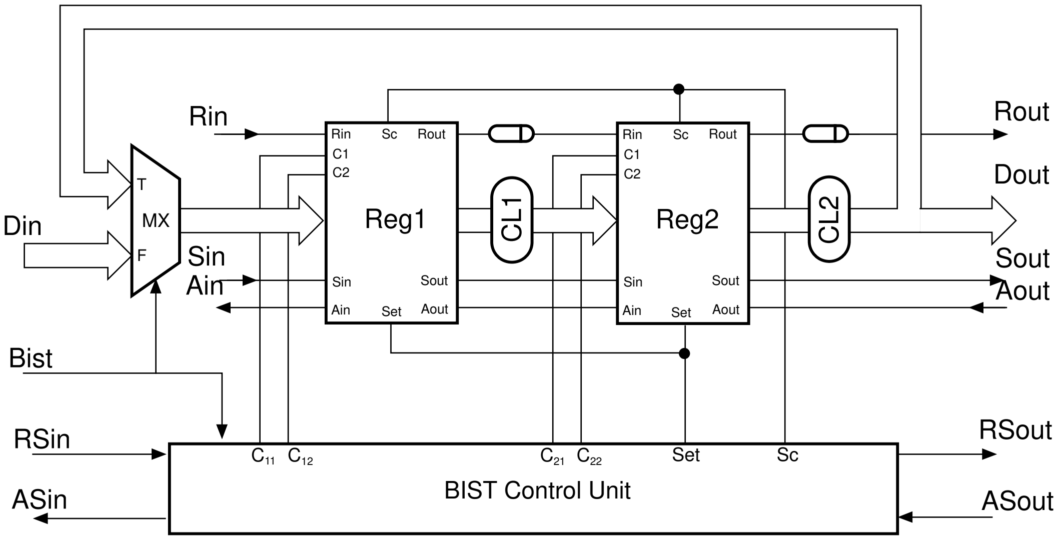


Figure 6.26: A two-stage micropipeline with BIST features

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 6.26 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 control unit.

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 6.25). 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. This test procedure is similar to the one shown in Figure B.5. The *Rin* input is triggered enough times for the LFSR to generate on the inputs of *CL1* the required number of random test vectors. The number of random tests can be calculated using formula (6.2). The combinational circuit *CL2* is tested in the same way described above when *Reg2* acts as the LFSR and *Reg2* is the signature analyser. The responses from *CL2* are passed through the multiplexer to the inputs of *Reg2*. Note that the signatures produced by the signature analysers are shifted out to the *Sout* output of the micropipeline every time when 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.

The use of the above test algorithm allows most of stuck-at faults in the micropipeline to be detected. Note that faults *reset*-SA1 in the register latches (see Figure 6.24) cannot be identified in BIST mode whereas they can be tested in normal operation mode. The test algorithm is simple:

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 fault *reset*-SA1 in the *i*-th latch of the *j*-th (*j*=1,2) stage register the response bit latched in its (*i*+1)-th latch will be inverted (see Figure 6.25).

Using the micropipeline structure shown in Figure 6.26 delay faults in its combinational circuits can be tested. In this case the number of latches in the BILBO registers (see Figure 6.25) 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 of the double length. In addition, an NOR gate can be added to the design of the LFSR (see Figure A.2b) in order to allow the LFSR to go through all its possible states.

According to Lemma 6.1 the modified LFSR can produce all possible combinations between any two 2-bit binary vectors chosen sequentially on its outputs when the LFSR has passed through all its possible states. Thus, any possible pair of test vectors can be generated on the inputs of the combinational circuits during their testing. As a result, delay faults and stuck-at faults can be detected in the combinational circuits of such a micropipeline (see chapter 5).

6.7.3 Analysis of the BIST micropipeline structure

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

Advantages

- The BIST micropipeline structure shown in Figure 6.26 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 section 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 6.25) 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.

6.8 Summary

Different asynchronous implementations of the pseudo-random pattern generator and signature analyser have been proposed in this chapter. These designs allows the synchronous LFSR to be used to build either the generator or signature analyser.

Two structural DFT approaches to designing asynchronous sequential VLSI circuits have been discussed. During test mode the asynchronous sequential circuit is tested asynchronously in a manner similar to well-known DFT techniques: the combinational logic block and all the storage elements are tested independently which simplifies the test greatly.

The parallel random pattern testing technique described in this chapter uses the probabilistic properties of the 2-input XOR gate. According to this technique a block of XOR gates is placed in the feedback data path of the sequential circuit and random test vectors are applied to the second inputs of the XOR gates. The outputs of the XOR gates are observed on the extra test outputs. As a result, the state registers and the combinational circuit are tested in parallel. The proposed bit-serial random test technique provides for the bit serial scanning of test patterns into the state registers of the sequential circuit and the bit serial scanning out of the responses of the combinational logic block from the internal signature register. This makes the testing faster then a traditional scan test. The random pattern test length for the testable asynchronous sequential circuit is equal to the test length of the random testing of the combinational circuit and can be estimated easily. The hardware redundancy of the proposed approach depends greatly on the complexity of the combinational logic block.

The AMULET2e memory controller has been implemented as two-phase, four-phase and handshake circuits using both the parallel and bit-serial design for random pattern testability techniques. The results show that the proposed design methods have practical feasibility which allows them to be used to build various kinds of asynchronous sequential circuits for random pattern testability.

Finally, the proposed BIST micropipeline design can be used in asynchronous VLSI circuits where the pseudo-random pattern generator and the signature analyser are placed on the chip.

Chapter 7 : Design for Testability of an Asynchronous Adder

There are several different ways to implement an asynchronous adder, and each has particular testability characteristics. In this chapter the stuck-at fault model is used to describe fault effects in various adder implementations. We show that stuck-at faults on the data dependent control lines of the single-rail adder can cause both premature and delayed firings of its control outputs. The choice of single-rail, dual-rail or combined single and dual-rail (hybrid) data encoding techniques brings different trade-offs between the testability, performance and area overhead. A case study of an asynchronous comparator demonstrates that a hybrid implementation brings a reasonable compromise between the area overhead, performance degradation and testing costs.

7.1 AMULET1 asynchronous adder

An asynchronous ALU is a major element in the AMULET1 microprocessor. It has been shown that about 80% of the operations performed by the ALU require different forms of addition [Gars93]. The correct performance of the adder as the ‘busiest’ part of the asynchronous ALU is therefore important for the correct functioning of the AMULET1 design as a whole.

Three input bits are used to implement a one-bit addition: two data bits and one *carry-in* bit which is effectively the *carry-out* signal from the previous stage of the multi-bit adder. The complete truth table of a 1-bit full adder is shown in Table 7.1. The performance of the multi-bit adder depends on the propagation speed of the *carry* signal through

Table 7.1: *Truth table for the full adder*

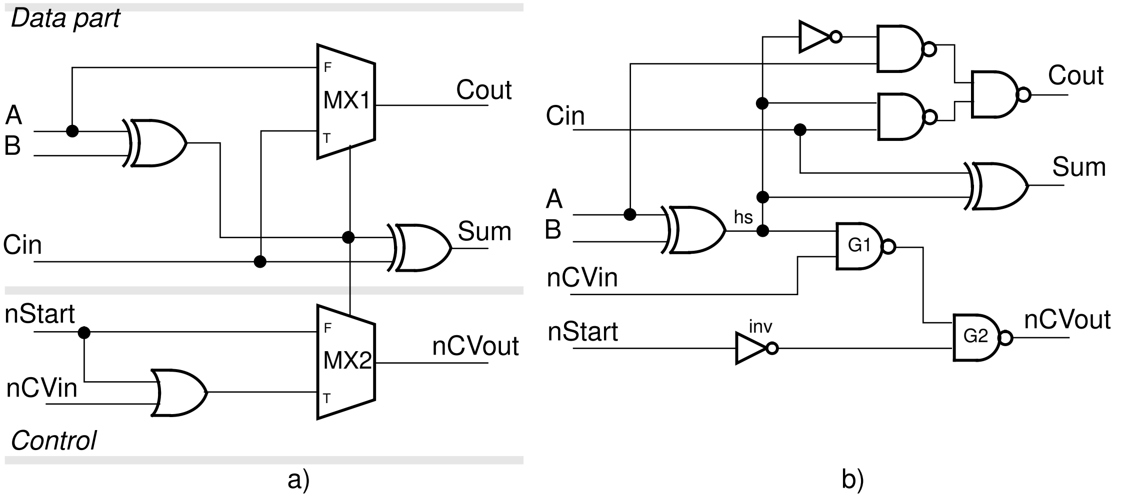
Inputs			Outputs	
A	B	Cin	Sum	Cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Table 7.2: *Truth table for the full adder carry output*

Inputs		Output
A	B	Cout
0	0	0
0	1	Cin
1	0	Cin
1	1	1

its stages. Table 7.2 illustrates the truth table for the carry output of the 1-bit full adder. According to this table the *carry-out* signal can be predicted in half of the possible input combinations. This allows the correct *carry-out* signal to be generated without waiting until a *carry-in* signal is produced by the previous stage of the adder. This technique has been used in the implementation of the AMULET1 adder [Gars93].

In the AMULET1 asynchronous adder, addition results are ready when all the *carry-out* signal are ready. The carry chain of the adder is implemented using dual-rail data encoding where the readiness of the *carry-out* signal is identified by a transition on one of its two data wires. Since the *carry-out* signal of the AMULET1 adder is data dependent and



a) b)

Figure 7.1: *Single-rail implementation of an asynchronous 1-bit full adder: a) using multiplexers; b) using logic gates*

data values which cause long carry propagation paths are relatively rare the adder itself exhibits average rather than worst case performance [Gars93].

7.2 Single-rail asynchronous adder

Figure 7.1a shows the implementation of a single-rail asynchronous 1-bit full adder using multiplexers. The adder design consists of distinct data and control parts. The data path of the adder produces an addition result on its *Sum* output and generates a *carry-out* signal on its *Cout* output. Note that the *carry-out* function is implemented according to Table 7.2. The control part of the adder is designed to indicate when a *carry* output is ready to be read by the environment. When the data is ready on inputs *A* and *B* a *start* signal is generated on the *nStart* input which is active low. If the values on the *A* and *B* inputs are equal the *start* signal is passed to the *carry-valid* output of the adder. If not, an active low *carry-valid-in* signal is transmitted from the *nCVin* input through the OR gate and multiplexer *MX2* to the *nCVout* output. The control part of the adder follows the four-phase signalling protocol, i.e., when the addition result has been read by the environment signal *nStart* is set to high and then output *nCVout* goes high. Figure 7.1b illustrates the gate level representation of this asynchronous 1-bit adder with single-rail data encoding. This adder performs in the same manner as described above.

The design of an asynchronous single-rail 8-bit adder is shown in Figure 7.2. In this design all the 1-bit full adders are connected together in a chain where the *carry* output and the *carry-valid* output of the previous 1-bit adder are connected to the *carry* input and the *carry-valid* input of the following 1-bit adder respectively. The *carry-valid* output of the adder (*Ack*) is produced on the inverted output of the 8-input symmetric C-element, the inputs of which are connected to the corresponding *nCVout* outputs of the 1-bit adders. The *carry-out* signal of the last 1-bit adder is used as the *carry* output of the 8-bit adder.

The global *start* signal is connected to all of the 1-bit adders. The first adder (*Ads0*) does not have a *start* input since its *carry-valid* input is connected to the global *start* signal.

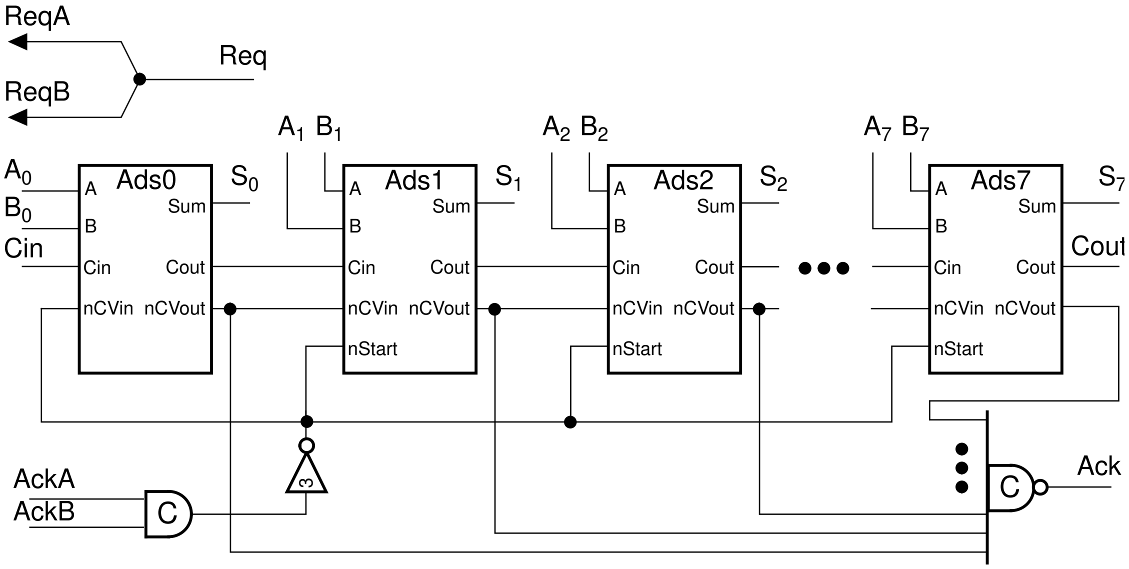


***Figure 7.2:** Asynchronous 8-bit adder with single-rail data encoding*

The *start* signal from the *nCVin* input of adder *Ads0* is delayed for enough time for the *carry-out* signal to be stable before it is passed directly to the *nCVout* output.

A request for addition is sent by the environment on the *Req* input of the adder. When the data is ready on the *A* and *B* inputs two acknowledge signals are generated on inputs *AckA* and *AckB* of the two-input symmetric C-element. When the output of the C-element is set high an active low *start* signal is transmitted to the corresponding inputs of all of the 1-bit adders. A rising event on the *Ack* output of the 8-input C-element acknowledges the completion of the addition. Once the results are read the request signal is returned to zero on the *Req* input. As a result, acknowledge signals on inputs *AckA* and *AckB* are set to zero. The two-input C-element is reset and the global *start* signal goes to high. The handshake procedure is completed when the acknowledge signal on output *Ack* of the adder is reset.

Note that a control signal which fires when the *carry-in* signal (*Cin*) is ready can be implemented separately (for instance, using extra signals *AckC* and *ReqC*), or the *Cin* signal can be transmitted together with one of the operands (*A* or *B*) as demonstrated in Figure 7.2. The choice between these techniques depends on the particular environment in which the adder performs. Hereafter, the *carry-in* signal for the adder is assumed to be transmitted together with one of the operands.

7.3 Testing of a single-rail asynchronous adder

In this chapter, the single stuck-at fault model including stuck-at-input and stuck-at-output faults is considered [Russ89]. In order to test the adder shown in Figure 7.2 a set of test patterns must be applied to its inputs. The test results are observed on the outputs of the adder. It is assumed that the inputs of the asynchronous adder are controllable and its outputs are observable by the environment.

Consider the design of the asynchronous 8-bit adder shown in Figure 7.2. The detection of stuck-at faults in the data part of each 1-bit adder is trivial since its data inputs and outputs are controllable and observable during the test. The control part of the 8-bit adder is illustrated in Figure 7.3. Inputs *hs* are the outputs of the corresponding XOR gates of the 1-bit adders (see Figure 7.1b). As was mentioned above the control part of the first adder is simply a delay matching the *carry-valid* input to *carry-valid* output delay (see Figure 7.3).

Stuck-at faults in the control part of the adder can be divided into three distinct classes:

1. Stuck-at faults which are detectable by logic testing. For instance, stuck-at-0 or stuck-at-1 faults on the *CVout* outputs are easy to detect since they violate the hand-

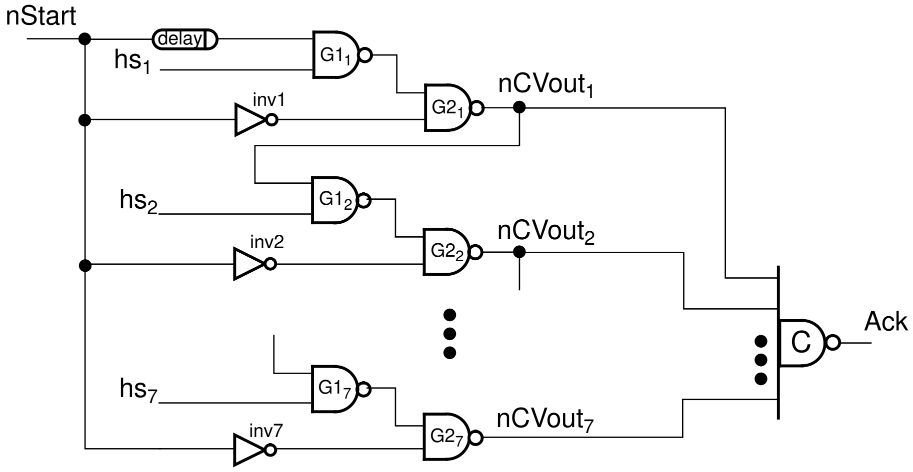


Figure 7.3: Control part of the single-rail 8-bit adder

shake communication protocol between the adder and its environment. Note that a stuck-at-1 or a stuck-at-0 fault on an input of the symmetric C-element shown in Figure 7.3 is equivalent to a stuck-at-0 or stuck-at-1 fault on its output respectively.

2. Stuck-at faults which can cause a premature firing on output *Ack*. A stuck-at-1 fault on the output of NAND gate $G1_i$ (i=1, 2, ... , 7) does not change the logic function of the control part of the adder but causes a premature firing on the output of gate $G2_i$ when hs_i=1. This fault may or may not cause the environment to latch wrong data from the outputs of the adder depending on how fast or slow the environment performs. This fault is hard to detect by delay testing due to the absence of synchronization clocks.

3. Stuck-at faults which can cause delayed firings on the control output of the adder. These faults do not change the logic function of the control part of the adder but reduce its performance. The detection of such faults requires delay testing. For instance, a stuck-at-1 fault on input hs_i (i=1, 2, ... , 7) causes a delayed response from the adder.

Let us consider the Boolean function of output $nCVout_1$:

$$nCVout_1 = \overline{\overline{nStart \cdot hs_1} \cdot \overline{nStart}} = nStart \cdot hs_1 + nStart = nStart \qquad (7.1)$$

It is easy to show that

$$nCVout_i = nStart \cdot hs_i + nStart = nStart, \qquad (7.2)$$

where i=1, 2, ... ,7.

Thus, the control part of the adder has logic redundancy. Redundant logic elements are necessary to ensure the proper timing function of the control part of the adder. This makes some of its stuck-at faults impossible to detect by logic testing.

7.3.1 Design for testability of the single-rail asynchronous adder

In order to make the asynchronous adder shown in Figure 7.2 testable, the logic redundancy of its control part must be removed during the test. Figure 7.4 shows the design of a testable asynchronous 1-bit adder. It performs in two modes: normal operation mode and test mode. The mode of the adder is changed by the Boolean signal *Tst* which is high in test mode and low in normal operation mode. Input *Tst* and the output of the XOR gate (*G3*) are connected to the inputs of the asymmetric C-element.

Figure 7.5 illustrates a CMOS implementation of the asymmetric C-element. If both inputs *In1* and *In2* are high the state of the asymmetric C-element is one. The output (*Out*) of the C-element is set to zero when its input *In1* is low. The C-element keeps its current state under the application of any other input combinations. A stuck-at-0 fault on one of the inputs of the asymmetric C-element prevents its output from being changed from low to high. A stuck-at-1 fault on input *In1* prevents the output of the asymmetric C-element from being changed from one to zero. To detect a stuck-at-1 fault on input *In2* of the C-element the following pair of tests must be applied to its inputs: (00, 10). If the state of the C-element is one the circuit is faulty, otherwise it is fault-free.

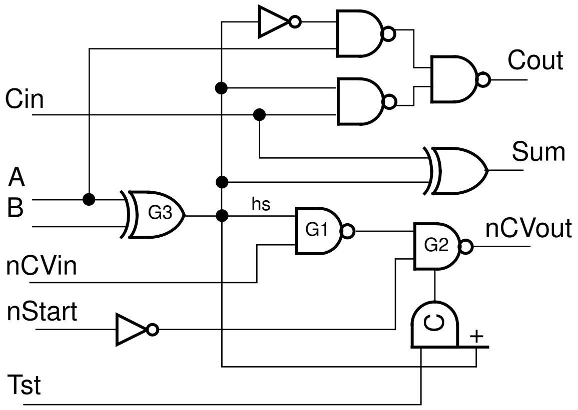


Figure 7.4: *Testable asynchronous 1-bit full adder with single-rail data encoding*

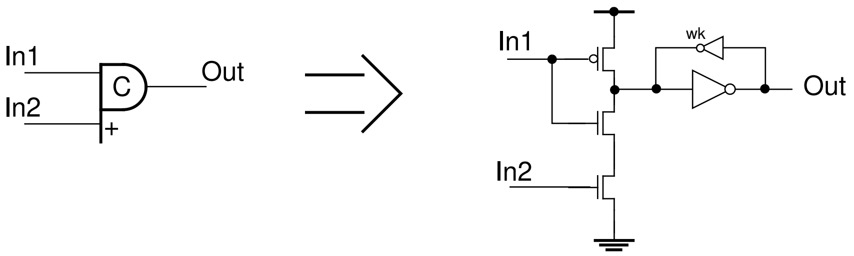


Figure 7.5: *Asymmetric C-element notation*

The output of the asymmetric C-element controls the NAND gate (*G2*) which can perform either as an NAND gate or as an inverter depending on the value of its control signal. A CMOS implementation of gate *G2* is illustrated in Figure 7.6. If the mode control signal (*Tst*) is low the gate performs as a two-input NAND gate. If the Boolean *Tst* is high, input *b* of the gate is blocked and it performs as an inverter of input *a*.

In normal operation mode the control part of the 8-bit adder is identical to the circuit shown in Figure 7.3. A fault analysis of the control part of the adder has been carried out with the help of automatic test generation program tools designed in the department of Electrical Engineering at Virginia Polytechnic Institute [LeeTR93]. As a result, 27 redundant stuck-at faults have been identified. The fault coverage of the tests generated for detecting faults in the control part of the adder is 53%.

In order to set the adder to test mode signal *Tst* and outputs hs_i of XOR gates $G3_i$ (i=1, 2,..., 7) are set to high. In test mode the control part of the adder is identical to an AND

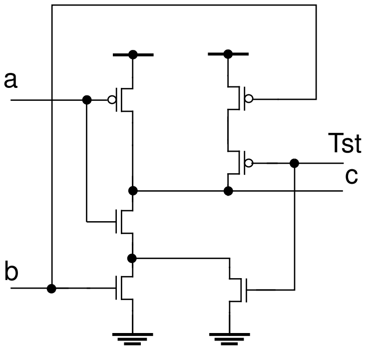


Figure 7.6: *Transistor level implementation of the NAND/INV gate*

gate with output $nCVout_7$ as shown in Figure 7.7. Stuck-at faults in such a circuit can be detected easily by a standard set of (n+1) tests for an n-input AND gate: one 'all ones' test and n 'running zero' tests. For the circuit illustrated in Figure 7.7 n=7. Note that signal *nStart* is a control signal which must be returned to zero after the application of each test vector. Moreover, the application of 7 'running zero' tests detects whether or not all gates *G2* of the control part perform as inverters.

To detect stuck-at faults on the $nStart_i \rightarrow Ack$ path (i=1, 2,..., n) in the control part of the i-th 1-bit adder, the following test algorithm can be used:

1. i=1.

2. Tst=0; hs_i=0; hs_j=1 (for all $j \neq i$).

3. Tst=1. Gate $G2_i$ performs as a NAND gate whereas gates $G2_j$ ($j \neq i$) perform as inverters (see Figures 7.3 and 7.4).

4. Signal *nStart* is set to low and then to high.

5. If output *Ack* has been changed twice, path $nStart_i \rightarrow Ack$ is fault free, then go to step 6 else go to step 9.

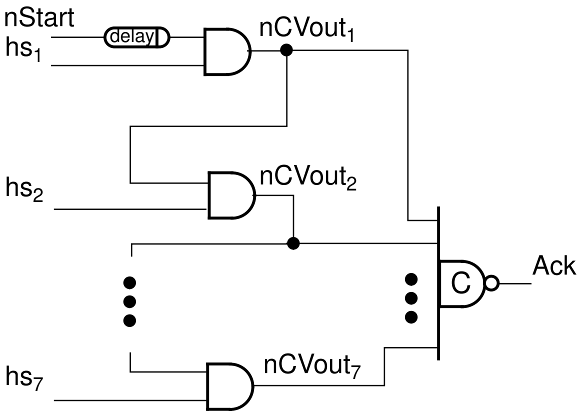


Figure 7.7: *Control part of the single-rail 8-bit adder in test mode.*

6. i=i+1.

7. If $i > n$ then go to step 8 else go to step 2.

8. The circuit is fault free. Go to step 10.

9. The circuit is faulty. Go to step 10.

10. End.

In summary, the logic testing of the asynchronous adder illustrated in Figure 7.2, which contains the testable 1-bit adders shown in Figure 7.4, is difficult due to the test complexity of its control part. For instance, 8 test vectors are required to test the data path of the adder whereas the number of tests required to test the data dependent control part is almost twice this number.

7.4 Dual-rail implementation of an asynchronous adder

A dual-rail implementation of an asynchronous 1-bit adder is shown in Figure 7.8a. It contains a single-rail to dual-rail data conversion block (*SDC*), dual-rail and single-rail XOR gates and a dual-rail multiplexer.

The single-rail conversion block modifies the single-rail data from inputs *A* and *B* into the dual-rail data format. A gate level implementation of the conversion block is shown in Figure 7.8b. When signal *nStart* is high the outputs of the conversion block are kept low. If the data is ready to be transmitted to the adder signal *nStart* is set low and the single-rail data from inputs *A* and *B* is converted into the dual-rail format. Designs of the dual-rail multiplexer and XOR gate are illustrated in Figures 7.8c and 7.8d respectively. The use of symmetric C-elements in the design of the XOR gate ensures its delay-insensitivity which, in turn, simplifies its testing. As was mentioned above a stuck-at fault on the inputs of the symmetric C-element is equivalent to the corresponding stuck-at fault on its output.

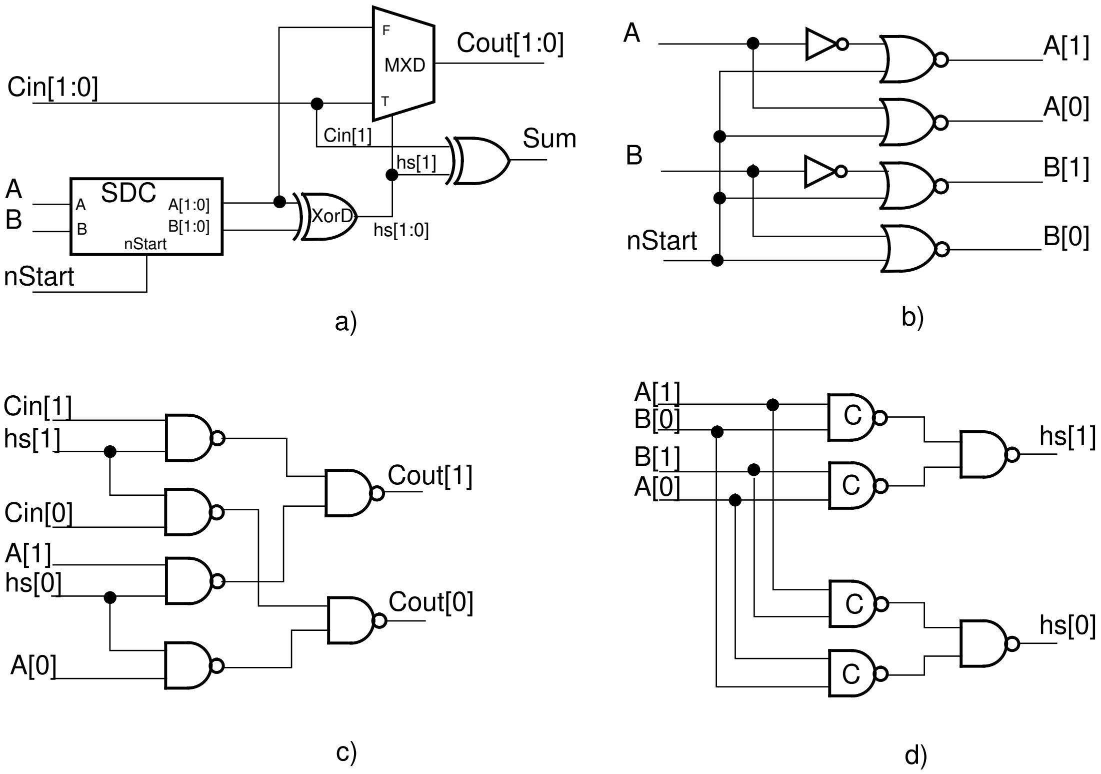


Figure 7.8: *Implementations of a) a dual-rail asynchronous 1-bit full adder; b) a conversion element between single-rail and dual-rail data encoding; c) a dual-rail multiplexer; d) a dual-rail XOR gate.*

The single-rail result (*Sum*) of addition is produced by XORing signals *Cin[1]* and *hs[1]*. Note that a control part which indicates when the *carry-out* signals are ready is no longer needed. The completion of the addition is identified by a rising transition on one of the *carry* outputs.

An example of the dual-rail implementation of an asynchronous 8-bit adder is shown in Figure 7.9. The inputs and outputs of the adder are single-rail encoded. When a single-rail data is ready on inputs *A*, *B* and *Cin* acknowledge signals *AckA* and *AckB* are set high. As a result, signal *nStart* goes low and addition is started. The output data is ready if the dual-rail carry outputs (*Cout[1]* and *Cout[0]*) of all the 1-bit adders are different which is indicated by a rising transition on output *Ack*. The actual *carry* output is taken from output $Cout_7[1]$ of the last 1-bit adder. After latching the outputs of the adder the environment returns request signal *Req* to zero. Acknowledge signals go low and signal *nStart* is set high. As a consequence, all the outputs of the adder are set to zero.

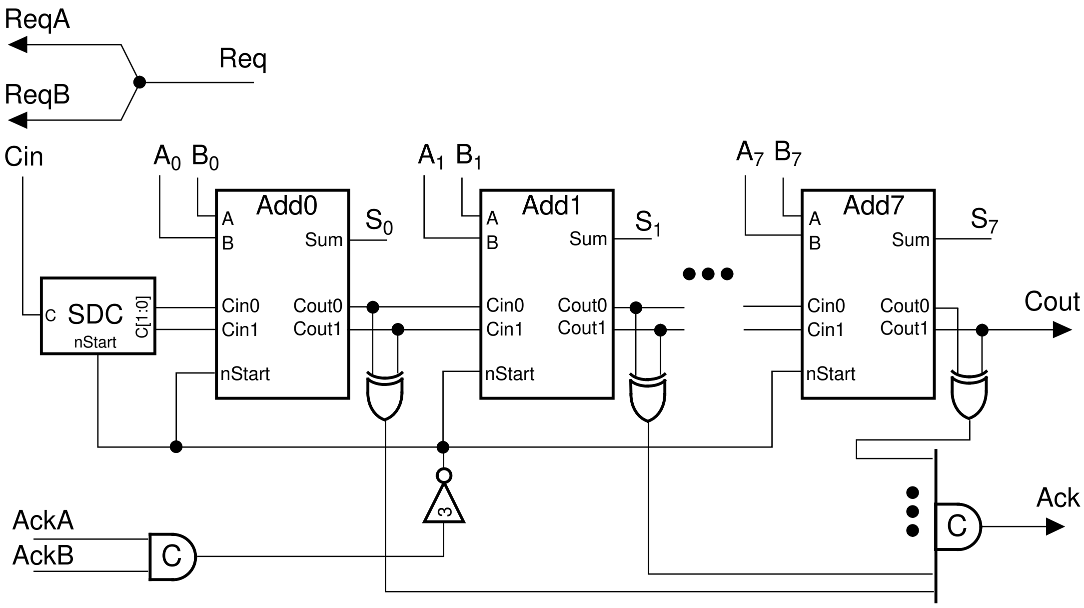


Figure 7.9: Dual-rail implementation of an asynchronous 8-bit adder

A fault analysis of the dual-rail implementation of the 8-bit adder shown in Figure 7.9 was carried out using *SIMIC* design verification tools developed by Genashor Corporation [Sim94, Ashki94]. The results show that the dual-rail adder is fully testable for its stuck-at faults after the application of 29 test vectors during normal operation mode.

7.5 Hybrid implementation of an asynchronous adder

The implementation of the dual-rail asynchronous adder described in the previous section requires more silicon area than that of the single-rail adder but it can be tested in its normal operation mode. In this section a hybrid implementation of an asynchronous adder is discussed. The design is called ‘hybrid’ because, firstly, some of the blocks of the adder perform using dual-rail input data and, secondly, the hybrid adder has a control part similar to that of the single-rail adder. As a result, the high level implementation of the hybrid adder is identical to that of the single-rail adder.

The implementation of a hybrid 1-bit full adder is illustrated in Figure 7.10. It converts the single-rail data from inputs *A* and *B* using the conversion block which is controlled by the active low *start* signal (*nStart*). Output *hs[1]* of the dual-rail XOR gate (*XorD*)

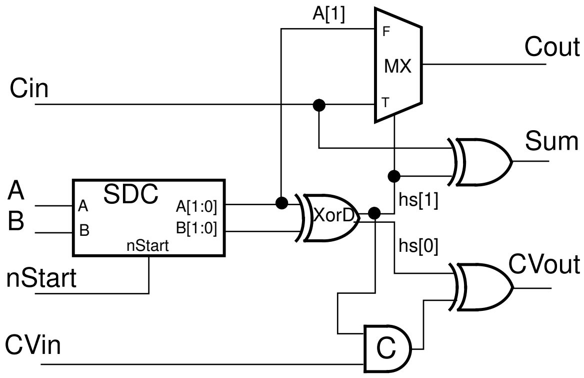


Figure 7.10: *Hybrid implementation of an asynchronous 1-bit full adder*

controls the single-rail multiplexer (*MX*) which connects the *carry* input of the adder or output *A[1]* of the conversion block to its output. The control part of the adder uses both outputs of the dual-rail XOR gate to generate a *carry-valid* signal which is active high. When *hs[0]*=1, i.e., inputs *A* and *B* are equal, output *CVout* of the adder goes high indicating the completion of the addition. When input bits *A* and *B* are different *hs[1]*=1 and the symmetric C-element is primed (see Figure 7.10). The output of the C-element is set to high when input *CVin* goes high. As a result, a rising event is generated on the *CVout* output of the adder.

As was mentioned above the design of the hybrid asynchronous adder is similar to that of the single-rail adder shown in Figure 7.2. When the data is ready on the inputs of the adder signal *nStart* is set to zero and the input data is converted to the dual-rail format. The completion of the addition is indicated by a rising event on output *Ack* of the multi-input symmetric C-element. When the *start* signal is returned to zero the data and control outputs of the adder are reset. In order to return the *carry-valid* output of the hybrid asynchronous adder to zero all the symmetric C-elements in the control paths of the 1-bit adders must be set to zero (see Figure 7.10). If all $hs_i[1]$ (i=0, 1, ... , 7) were set to one all C-elements in the control part of the hybrid adder are returned to zero sequentially starting from the first 1-bit adder and finishing in the last one. This is the worst case performance of the hybrid asynchronous adder.

A fault analysis of the hybrid asynchronous 8-bit adder shows that all its stuck-at faults are testable and fault detection requires the application of 33 test vectors during its normal operation mode.

7.6 A case study of an asynchronous comparator

In this section the design of an asynchronous 8-bit comparator is considered. The comparator is used as a comparison block for a pair of 8-bit input vectors in an asynchronous block sorter the implementation of which is described in chapter 8. Figure 7.11 illustrates the design of the asynchronous comparator. It contains an asynchronous 7-bit adder to perform subtraction of the data from inputs *A* and *B* as follows

$$\overline{Cout} = A + \bar{B} + Cin \tag{7.3}$$

where $A=A[7:1]$ and $B=B[7:1]$.

The *carry* input of the adder is generated by ORing the least significant bits of 8-bit operands *A* and $\bar{B}$, i.e.

$$Cin = A[0] \vee \bar{B}[0] . \tag{7.4}$$

If *Cout* is low then *A* is greater or equal to *B* otherwise *A* is less than *B*. Note that the 7-bit adder of the comparator does not produce the results of subtraction. The comparator shown in Figure 7.11 performs in a similar way as was described in previous sections for a multi-bit asynchronous adder.

The 8-bit comparator was designed and implemented using a 1μm double metal CMOS process with the help of *Cadence* CAD tools. Several versions of the comparator with different implementations of its 7-bit adder have been simulated using *SIMIC* design verification tools. Simulation results are shown in Table 7.3. The single-rail adder without testability features is taken as a base for estimating the relative characteristics of the other adder designs since it requires the minimal silicon area and demonstrates the highest performance. The performance of each version of the comparator was calculated in

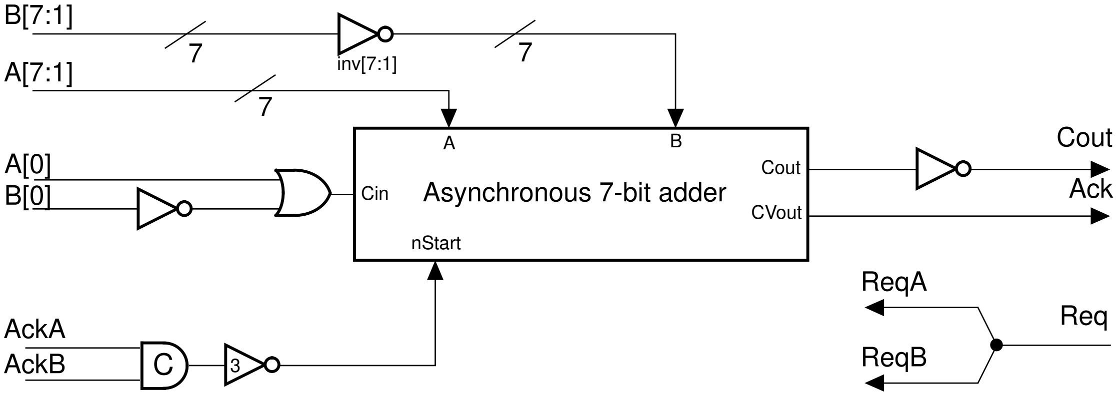


Figure 7.11: Asynchronous 8-bit comparator

normal operation mode by applying an identical set of 128 tests generated by a pseudo-random pattern generator.

According to the simulation results shown in Table 7.3 the comparator with the dual-rail adder demonstrates the largest area overhead (138%) compared to the comparator which uses the single-rail adder without testability features. The comparator with the hybrid adder shows the lowest performance which is close to that of the dual-rail comparator. The comparator with the testable single-rail adder demonstrates the minimal area overhead and performance degradation but requires a special test mode. The use of the hybrid adder in the comparator brings a compromise between area overhead, performance degradation and testability. It is easy to test since 100% of its stuck-at faults are detected in normal operation mode. However, it is 30% slower and its implementation is almost twice as large as the comparator which uses the single-rail adder without testability features.

7.7 Summary

Different designs of an asynchronous adder and their testability properties have been investigated in this chapter. The single-rail implementation of an asynchronous adder is least complex in terms of number of gates, and is fast, but it demonstrates low stuck-at fault testability due to the logic redundancy in its control part. The logic testing of a sin-

Table 7.3: *Simulation results of the comparator using different adder designs*

Adder design of the comparator		Area, 10^{-2} mm^2	Performance, ns/test	AO[a], %	PD[b], %	No. extra pins
Single-rail adder	untestable	3.85	24.15	-	-	-
	testable	4.60	24.55	19	2	1
Dual-rail adder		9.17	30.48	138	26	0
Hybrid adder		7.40	31.50	92	30	0

a. AO is the area overhead
b. PD is the performance degradation

gle-rail asynchronous adder requires a special test mode to be implemented in order to remove its logic redundancy. As a consequence, stuck-at faults which have not been detected in normal operation mode can be identified in test mode. The dual-rail and hybrid implementations of the asynchronous adder are fully testable for stuck-at faults in normal operation mode but they require more area and exhibit lower performance. The dual-rail implementation of an asynchronous adder is faster than the hybrid adder but requires more silicon area. The dual-rail and hybrid adders can be used in asynchronous VLSI designs where performance and area overhead are not critical but testability in operation normal mode is important. The testable single-rail version of the adder can be used in asynchronous VLSI circuits which can be tested in both normal operation mode and test mode.

Chapter 8 : The Design and Test of an Asynchronous Block Sorter

The design of an asynchronous block sorter and issues relating to its testability are discussed in this chapter. The sorter takes an input data stream and sends it to the output sorted in a descending order. The scan test and BIST design methodologies are used to implement the block sorter for testability.

8.1 Design of the asynchronous block sorter

The design of the asynchronous block sorter is shown in Figure 8.1. It consists of a head cell, 64 sorting cells and a tail cell which are connected in a chain [Farn95, FarnTR96]. All these blocks are fully asynchronous and operate autonomously.

The head takes 16-bit vectors from its input and passes 17 bit vectors to the first sorting cell. An extra boolean flag is added by the head to the input data. The first 63 input vectors have zero flags and only the last 64-th input vector has a boolean flag set to one. Flag one means the end of the block. The head contains an asynchronous modulo-63 counter which counts the number of input vectors passing through it. After completing 63 handshakes the head takes the last input vector and changes its boolean flag to one. When 64 input vectors have been passed through the head to the first sorting cell the head is ready to take a new block of input vectors.

Each sorting cell of the block sorter compares the 8 most significant bits of each new input vector[1] with a value stored in a register within the cell. Afterwards, each cell

---

[1] According to the specification of the block sorter

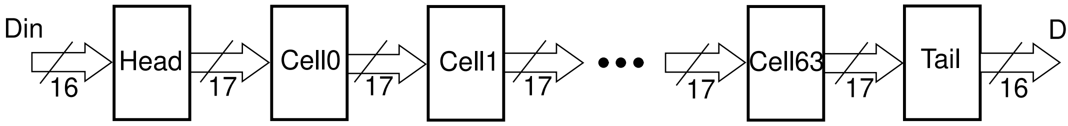


***Figure 8.1:** High level design of the block sorter*

passes the minimum vector to its output and stores the maximum one in its internal register. As a result, the set of 64 input vectors are stored in 64 sorting cells in a descending order.

The sorted block of 64 17-bit vectors are sent to the tail. The tail strips the boolean flag off its input value. The Tangram program for the asynchronous four-stage block sorter and handshake implementations of its basic components can be found in Appendix E[1].

All the sorting cells of the block sorter are identical. A block diagram of the sorting cell is shown in Figure 8.2. When a new block of input data is sent by the head to the sorting cells, the first 17-bit input vector of the block is passed from the *Din* inputs to the inputs of multiplexer *MX1* and register *Reg2*. Note that the first vector of each new block of

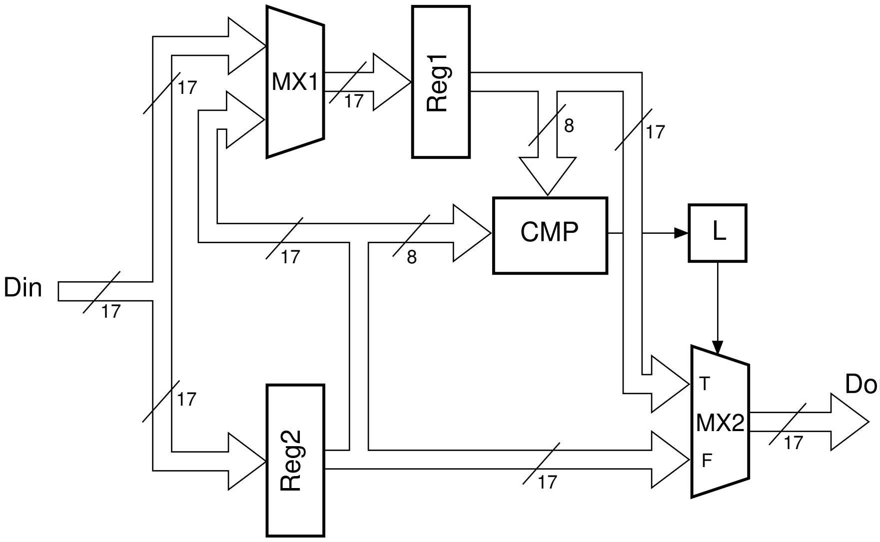


***Figure 8.2:** Block diagram of the sorting cell*

---

[1] By courtesy of Craig Farnsworth.

data is always stored in register *Reg1* first. A new input vector is latched by register *Reg2*.

The 8 most significant bits of registers *Reg1* and *Reg2* are compared by comparator *CMP*. The design of the comparator was considered in chapter 7. In particular, the comparator *CMP* was designed using the hybrid implementation of its asynchronous adder. The result of the comparison is stored in latch *L* which controls multiplexer *MX2*. If the value stored in *Reg1* is greater than or equal to the current value of *Reg2* then the state of latch *L* is zero and the vector written in *Reg2* is sent through multiplexer *MX2* to the output of the cell. If the value of *Reg1* is less than the value written in *Reg2* then the state of latch *L* is changed to one. As a result, the contents of *Reg1* are passed through multiplexer *MX2* to its outputs. The contents of *Reg2* are copied into *Reg1* and the cell is ready to write a new vector into *Reg2* in order to compare it with the contents of *Reg1*. The sorting procedure described above is repeated.

8.2 Testing the block sorter

8.2.1 Fault model

To design the asynchronous block sorter for testability we assume the stuck-at fault model. In particular we consider two types of stuck-at faults:

- gate level stuck-at output faults inside the control circuits of the block sorter;
- gate level stuck-at faults (including stuck-at input and output faults) inside the data processing blocks of the sorter.

Stuck-at faults on the control wires of the sorter are detected during its normal operation mode since they violate the handshaking protocol causing the whole circuit to halt [Dav90, Ron94]. The detection of stuck-at faults on the data paths of the circuit requires more care to be taken since they may or may not manifest themselves by producing wrong responses on the outputs of the block sorter.

8.2.2 Testable implementation of the sorting cell

The testing of data paths inside sorting cells is difficult since the controllability and observability of their internal nodes are different and dependent on the position of each cell in the chain. The testability of the block sorter can be increased by making the states of the registers inside the sorting cells controllable. One of the approaches to increase the controllability of the memory elements of the circuit under test is scan testing.

In principle, the scan testing technique assumes that all the registers of the testable design can be transformed into a shift register to scan the test vectors in and scan the test results out. As can be seen from the design of the sorting cell (Figure 8.2) the main data processing block which must be tested is the comparator. The full scan test implementation of the sorting cell requires the use of master-slave registers instead of *Reg1* and *Reg2* which effectively doubles their sizes. These registers would be used just to scan the test vectors in since the test results are not stored in them (Figure 8.2). Thus, the use of scannable registers is not efficient to test the sorting cell.

The structure shown in Figure 8.3 uses the system data paths of the sorting cell to send test vectors to registers *Reg1* and *Reg2*. The state of latch *L* can be observed on an additional output *CmpRes*. An extra multiplexer *MX3* is required to send either test or nor-

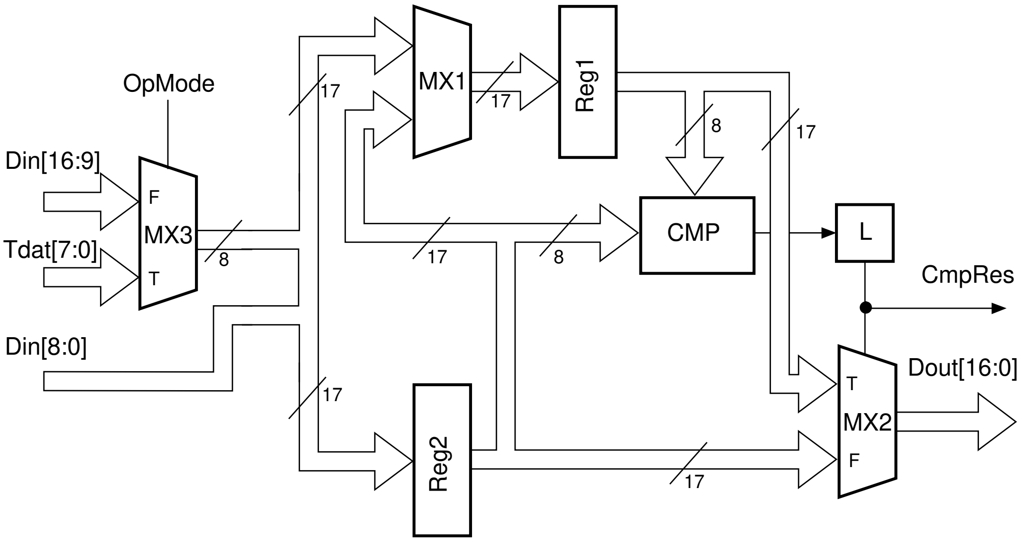


Figure 8.3: *Testable sorting cell*

mal 8-bit input vectors to the registers of the cell depending on the mode of operation. An additional boolean signal *OpMode*, which is high in test mode, controls multiplexer *MX3*.

In test mode multiplexer *MX3* passes 8-bit test vectors to the inputs of multiplexer *MX1* and register *Reg2*. The first and the second test vectors are written in the corresponding latches of registers *Reg1* and *Reg2* respectively. The data on bus *Din[8:0]* remains unchanged since it is not used to test the comparator. After the application of the pair of test vectors the test result is stored in latch *L* and can be observed on its output *CmpRes*. Thus, stuck-at faults inside the comparator can be detected during test mode.

The sorting cell is set to normal operation mode (*OpMode*=0) to test the rest of the circuit. Two blocks of tests are required to detect stuck-at faults on the data paths of the sorting cell:

- a block of ‘all ones’ tests;
- a block of ‘running one’ tests which starts with ‘all zeros’ test.

The first block of tests detects all stuck-at zero faults on the data paths involved in transferring the data to the outputs of the cell except the internal bus which connects the outputs of *Reg2* with the inputs of multiplexer *MX1*. The second block of tests detects the rest of the stuck-at faults which have not been identified by the previous test block. Since the test blocks must be applied to the inputs of the block sorter the size of each test block must be equal to the number of the sorting cells, i.e., 64.

8.2.3 Design for testability of the block sorter

Figure 8.4 illustrates the scan testable implementation of the asynchronous block sorter. The operation mode of the circuit is changed asynchronously along a two-phase *ChMode2* channel. The mechanism for changing the operation mode of the block sorter will be discussed later.

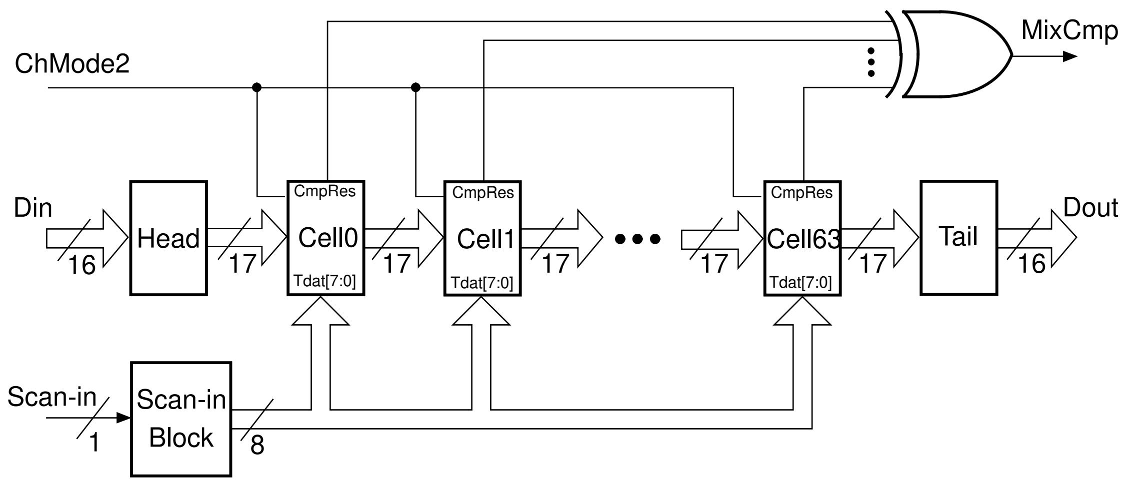


Figure 8.4: Scan testable design of the block sorter

In normal operation mode, the two blocks of test vectors described above are applied to inputs *Din* of the sorter. As a result, stuck-at faults on the control lines and the data paths of the block sorter are tested including the head and the tail of the block sorter. The next step is to test internal gate stuck-at faults of the comparator inside each sorting cell.

In test mode, test patterns are sent to the block sorter by the scan-in block. When a new test has been scanned into the scan-in block it is applied to each sorting cell in parallel since all cells are identical. The test results are mixed by a 64-input XOR gate and observed on its output *MixCmp*. In the presence of a single stuck-at fault inside the comparator of any sorting cell the output *MixCmp* will be changed to one during the test. Output *MixCmp* remains zero all the time during the test for the fault free block sorter.

8.3 Procedure for changing the operation mode

As was mentioned earlier a handshake circuit is a network of handshaking components which communicates via its channels. Each channel consists of a set of at least two wires (a request and an acknowledge wire) along which a four-phase communication protocol is performed. If the component performs a particular data processing operation an additional wire or a set of wires can be included to pass the data in and out. A transmission of signals along a channel can be made only after the receiver has confirmed the receipt of the previous transmission.

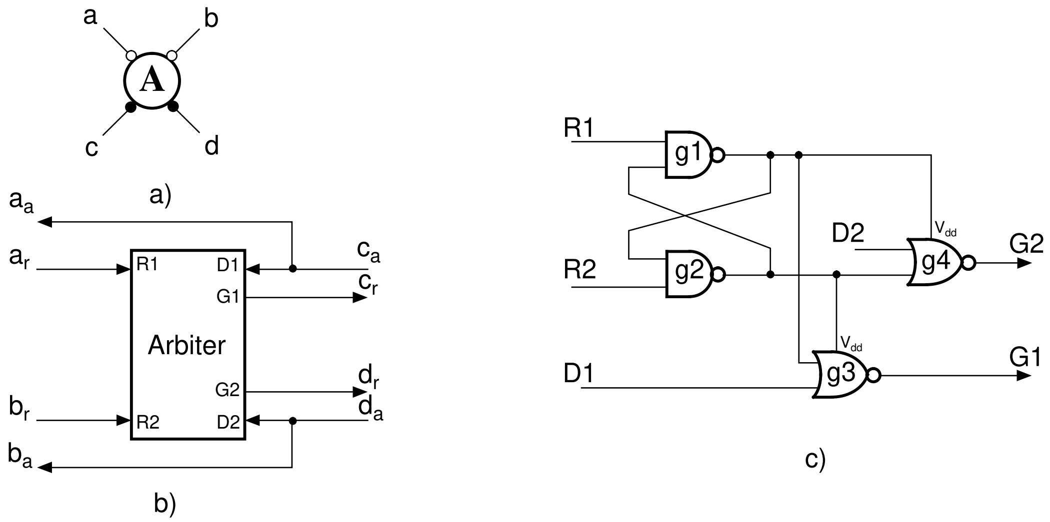


Figure 8.5: *Four-phase arbiter: a) symbol; b) high level; and c) gate level implementations*

States of asynchronous circuits are difficult to control since they operate autonomously. All the cells of the asynchronous block sorter are fully autonomous blocks which can be in any state during their operation. In order to set the sorter to test or normal operation mode one must be sure that the circuit cells are empty, i.e., the sorter has processed the current block of input data and ready to take a new one. The procedure for changing the operation mode of the circuit must be asynchronous since the cells of the sorter enter their empty states asynchronously and at different times.

An arbiter can be used to identify the situation when the circuit is empty. The arbiter takes a request for changing the operation mode of the circuit and waits until the circuit goes to the empty state. When it happens the arbiter serves the request and changes the operation mode of the circuit. A symbolic representation and four-phase high level design of the arbiter are shown in Figure 8.5a and 8.5b respectively. The arbiter has two passive (*a* and *b*) and two active (*c* and *d*) channels. If a request has arrived at the a_r input of the arbiter first the corresponding request signal is generated on its c_r output (see Figure 8.5b). After the completion of a handshake along channels *a* and *c* a new request generated on the b_r input can be served along channel *d*.

A gate level implementation of the four-phase arbiter is illustrated in Figure 8.5c. Request signals generated on the *R1* and *R2* inputs are stored in the RS flip-flop implemented using two NAND gates (*g1* and *g2*). The sources of the top *p*-type transistors in CMOS implementations of the two-input NOR gates *g3* and *g4* are connected to the outputs of gates *g2* and *g1* respectively (these connections are marked with V_{dd} on gates *g3* and *g4*). Initially, all the inputs of the arbiter are reset. As a result, the outputs of the RS flip-flop are high and the *G1* and *G2* outputs are low. If a rising event is generated first on the *R2* input of the arbiter the output of *g2* is reset preventing the output of NOR gate *g3* from being set to high (the *p*-stack of *g3* is disabled). The *G2* output is set to high. When the *D2* input has set to high the output of *g4* is reset and *G2* goes to low. Afterwards, the *R2* signal is returned to zero setting the outputs of the RS flip-flop to high. When the *D2* input is reset the arbiter is ready to serve another request. A request generated on the *R1* input is served in a similar way as a request on the *R2* request.

A fragment of the chain of the 64 sorting cells is shown in Figure 8.6. Each sorting cell contains an arbiter (A) which is placed after the repeater (*). The left communication channel of the arbiter serves requests when the cell is empty and ready to process a new data item. The right channel is used for changing the operation mode of the circuit.

In normal operation mode, the sorting cells enter their empty states sequentially starting from the first cell in the chain. Therefore, each cell which has changed its operation mode to test mode must generate a request to its right neighbour cell. The sorting cells are tested concurrently in test mode. Thus, a request to set the operation mode of the cells to normal mode must be applied to all the cells in parallel. This requires the use of a multiplexer (*Mx*) to steer the proper request signals to each sorting cell (see Figure 8.6).

Initially, the Boolean signal *ModeReq* is low and storage element *V*, which is the same as the one shown in Figure 1.8d but without the control signals in its output channel *b*, is reset. Operation mode signal *OpMode* is low and the case element (@) is switched to normal mode where its passive port is connected to its active port *NormCh2*. The sorting cells are activated along their activation channels marked with ⇨. Signal *ModeReq* is set to high in order to change the operation mode of the block sorter to test mode. As a result, a request signal is generated on input *GReq*. The arbiter of the first cell waits until

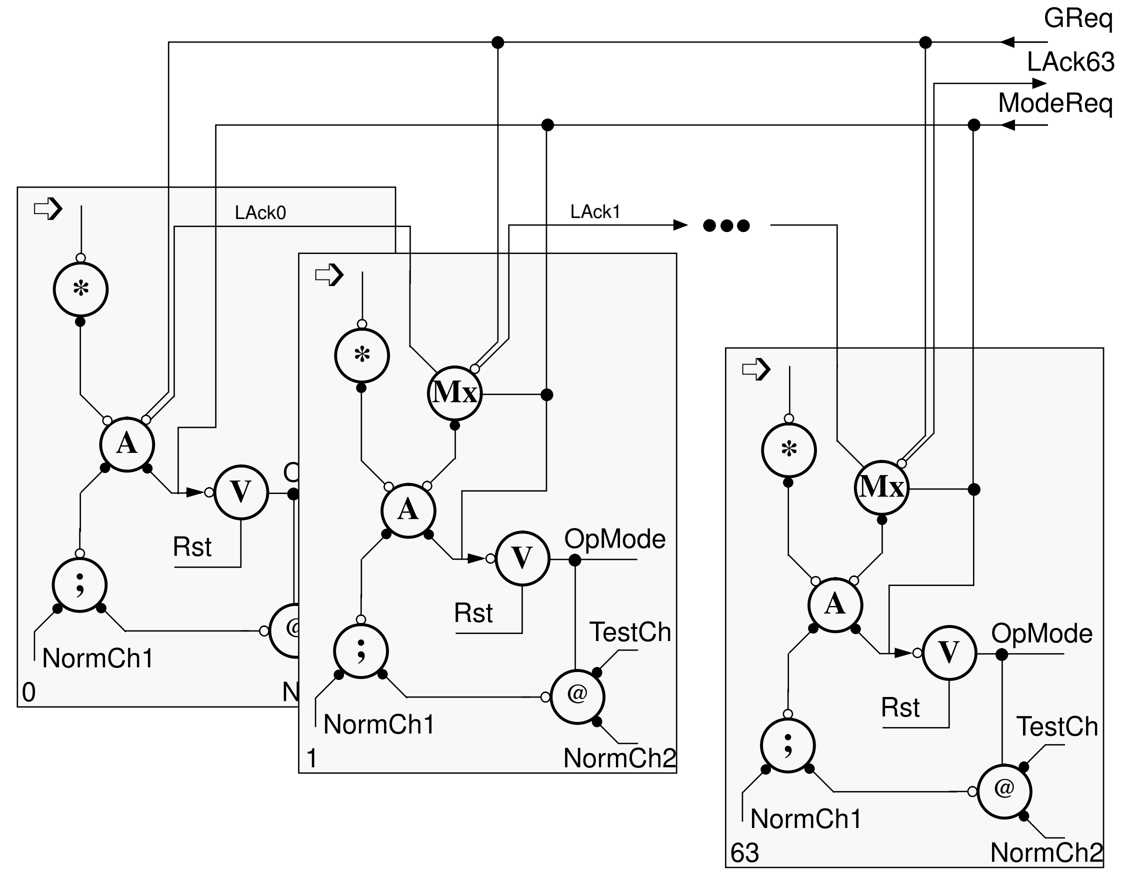


Figure 8.6: *Procedure for changing the operation mode of the sorting cells*

a handshake is completed on its left channel to activate its right channel. When the first cell is empty the output of the storage element *V* is set to one (*OpMode*=1*)*. The case element connects its passive port to its active port *TestCh*. An acknowledge signal is generated on output *LAck0* of the first cell which generates a request signal for the multiplexer of the second cell. When the second cell is empty the arbiter activates its right channel in order to change the operation mode of the cell. Once the second cell is set to test mode it produces a request on its output *LAck1* for the third cell. The procedure for changing the operation mode for the rest of the cells is repeated until the last cell is set to test mode. An acknowledge signal generated on output *LAck63* of the last cell indicates that signal *GReq* can be returned to zero. When signal *LAck63* is returned to zero all the sorting cells can be tested.

The design of the multiplexer of the *i*-th sorting cell is illustrated in Figure 8.7. When *ModeReq*=1 the multiplexer connects the *LAck* signal from the (*i*-1) cell to the c_r output.

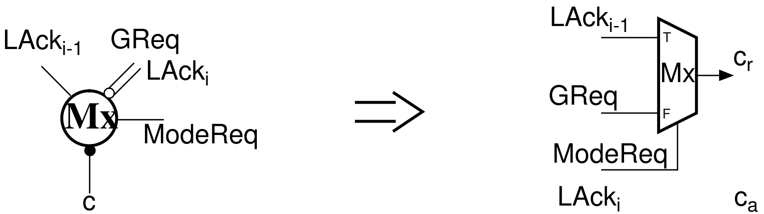


Figure 8.7: *Implementation of the multiplexer*

If the signal *ModeReq* is reset a request from the *GReq* input is passed to the c_r output of the multiplexer. Note that acknowledge signals are passed from the c_a input of the multiplexer to its $LAck_i$ output.

The block sorter is set to normal operation mode when boolean signal *ModeReq*=0. As a result, the multiplexers of the sorting cells are switched to pass a request signal from input *GReq* to the right channels of their arbiters in parallel. When the cells are empty the Boolean signal *OpMode* is set to low and *GReq* goes to high. Once acknowledged by a rising event on output *LAck63* of the last cell signal *GReq* can be returned to zero. When *GReq*=0 and the *LAck* signals of all the sells are returned to zero the block sorter can operate in normal mode. Note that the use of acknowledge signal *LAck63* ensures the correct fulfilment of the procedure for setting the sorting cells to normal operation mode since all the cells operate in parallel during the test.

In the presence of a stuck-at fault on input *GReq* or a stuck-at-1 fault on the *ModeReq* input of a multiplexer the corresponding cell will not be set to normal mode which eventually causes the block sorter to halt. A stuck-at fault on the *LAck* input or a stuck-at-0 fault on the *ModeReq* input of a multiplexer prevent the corresponding sorting cell from being set to test mode. These faults can be easily identified by causing the block sorter to halt.

The Boolean *ModeReq* is a two-phase signal which is high in test mode and low in normal operation mode. Since the whole circuit operates using four-phase signalling a mechanism for converting the two-phase signal into a four-phase one is required (see Figure 8.8). In the initial state the toggle element is reset and the inputs of the circuit are

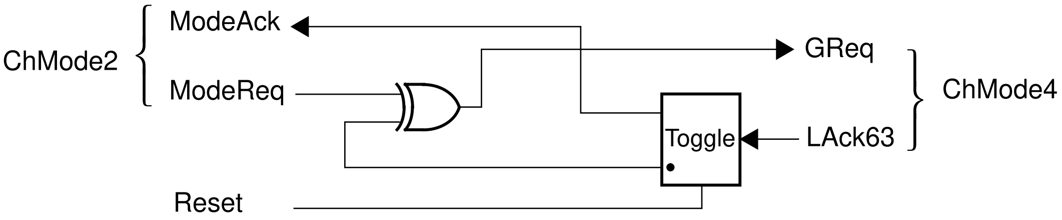


Figure 8.8: *Mechanism for converting a two-phase signalling along channel ChMode2 into a four-phase signalling along channel ChMode4*

low. A rising event on the *ModeReq* input is transmitted to the *GReq* output which is a request for changing the operation mode of the circuit. An acknowledge signal on the *LAck63* input changes the dotted output of the toggle element to one. As a result, the output of the XOR gate goes low and the *GReq* signal is returned to zero. When the acknowledge signal *LAck63* has returned to zero the toggle element steers a rising event to its output *ModeAck*. Thus, the two-phase signalling protocol is completed along channel *ChMode2* after the completion of the four-phase signalling protocol along channel *ChMode4*.

8.4 Test application

8.4.1 Scan testing

Test vectors for the sorting cells are generated by the scan-in block, the handshake implementation of which is shown in Figure 8.9. The *scan-in* procedure is activated along the activation channel marked with ⇨. This signal is steered by the two sequencers and triggers the passive port of the modulo-8 counter (#*8*). The modulo-8 counter has been implemented according to the methodology described in [Kess94]. The counter activates the passive port of the bit-serial shifter (*BS*) 8 times.

A four-phase implementation of the asynchronous shifter is illustrated in Figure 8.10. When the test bit is stable on the *scan-in* input the *Rin* signal goes high. As a result, the test bit is shifted into the shift register *SReg*. After the completion of handshakes along the input and output control channels the bit-serial shifter is ready to shift a new data bit.

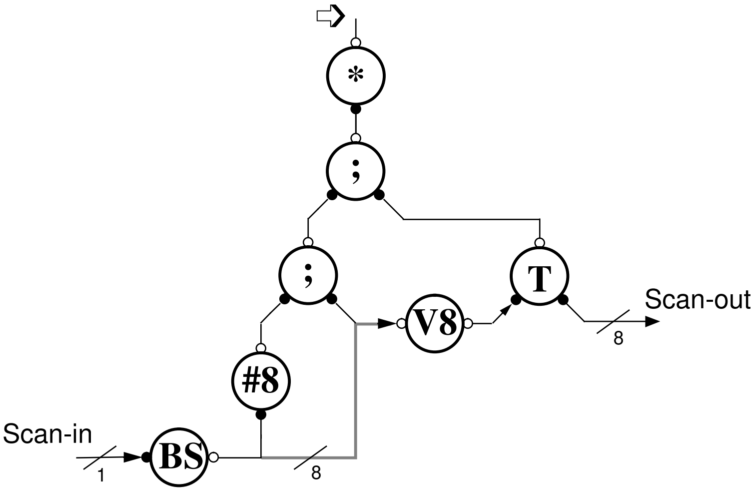


***Figure 8.9:** Handshake implementation of the scan-in block*

Once the test vector is shifted into the shifter *BS* the counter completes a handshake along the left channel of the bottom sequencer (see Figure 8.9). As a result, the contents of the shift register *SReg* are copied into the 8-bit storage element (*V8*). Thus, the top sequencer activates the transferrer which passes the contents of the storage element to the *Scan-out* channel. If the environment is ready to accept the data from the *Scan-out* channel the data from the outputs of the storage element is transmitted to the outputs of the scan-in block.

When a handshake has completed along the *Scan-out* channel the scan-in block activates its *Scan-in* channel and the procedure of scanning in a new test vector is repeated. The 8-bit storage element is used in the scan-in block to reduce the power dissipation of the block sorter during the scanning in procedure. In the absence of this buffer the fre-

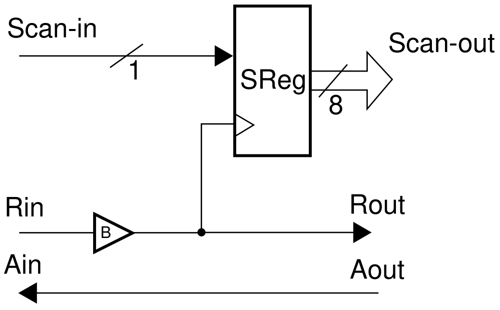


***Figure 8.10:** Four-phase bit-serial shift register*

```
ScanIn: proc(scan_in?(0 .. 1) & scan_out!(0 .. $2^8 - 1$ ))
begin
        testbit: var (0 .. 1)
        scantest, v8: var (0 .. $2^8 - 1$ )
|
        forever do
                for 8 do
                        scan_in?testbit;
                        scantest:=<<testbit.scantest.7. ... .scantest.1>>
                od
                v8:=scantest;
                scan_out!v8
        od
end
```

Figure 8.11: *Procedure ScanIn performed by the circuit shown in Figure 8.9*

quently changing outputs of shift register *SReg* would drive a large capacitance made by the test inputs of the 64 sorting cells during the shifting in operation.

Figure 8.11 shows an example of the *ScanIn* procedure written in the Tangram language. The procedure has the variable *testbit*, which is used to store a new data bit to be scanned in, and variables *scantest* and *v8* used to store the scan test vector. The *scan_in* input channel is activated 8 times to shift the scan data from the *testbit* variable into the *scantest* variable (*for 8 do* command). Once the shift operation is completed the content of the *scantest* variable is copied into the *v8* variable. As a result, the content of the *v8* variable is transmitted along the *scan_out* channel to the environment.

The test vectors for the comparator have been obtained using a set of automatic test generation program tools developed at Virginia Polytechnic Institute [LeeTR93]. A set of 18 pairs of 8-bit test vectors has been generated which detects all single stuck-at faults at the gate level representation of the comparator (see Figure 7.11).

8.4.2 Built-in self testing

In the design of the block sorter with BIST features the scan-in block of the scan testable block sorter (see Figure 8.4) is substituted by the pseudo-random pattern generator. The request-driven pseudo-random generator shown in Figure 6.3b was used in this BIST design. Note that the scan-in input used in the scan testable design is no longer needed in the implementation of the block sorter with BIST features.

An exact fault simulation analysis of the comparator illustrated in Figure 7.11 was carried out with the help of the *SIMIC* fault simulator. As a result, three hard-to-detect faults have been found (r=3) which have the minimal random pattern detection probability ($p_d = 1.95 \times 10^{-3}$). In order to calculate the upper bound of the random pattern test (L) formula (6.2) can be used. The calculation results showed that L=1741 and L=2920 for p_t=0.9 and p_t=0.99 respectively. The fault simulation results demonstrated that the application of 1000 pairs of 8-bit random test vectors is enough to detect all stuck-at faults inside the comparators of the sorting cells.

8.5 Simulation results and cost comparisons

The scan testable and BIST versions of the block sorter were designed using *Cadence* CAD tools and simulated using *SIMIC* design verification tools developed by Genashor Corporation [Sim94], [Ashki94].

8.5.1 Scan testable design

According to the scan test approach described above all test vectors are scanned into register *SReg* of the scan-in block. The outputs of the scan-in block are shared by all the sorting cells under test. The use of the system data paths of each cell in test mode reduces the cost of the scan procedure. Since the sorting cells are identical the test vectors can be applied to their inputs in parallel which makes the test procedure more effective. Note that the scan-in block is a fully asynchronous circuit which performs autonomously. This means that while the first test is sent to registers *Reg1* of the sorting cells (see Figure 8.3) the scan-in block can shift the second test in. As a result, the total test application time is reduced.

According to the simulation results shown in Table 8.1 the minimum application time of one pair of tests to each cell is 251ns. The area overhead for the scan testable design is 15.9% compared with the original version of the block sorter without testability features. In normal operation mode, the overall performance of the testable block sorter is degraded by 4% since extra components such as multiplexers and control handshake components are added to the original design. The average dynamic power consumption is equal to 28.4nJ per input vector in normal mode which is 7% more than that of the original version. In test mode the average dynamic power consumption of the testable sorter is equal to 27.8nJ per pair of tests.

A maximum of 8 extra pins are required to implement the scan testable sorter:

- 2 pins for the signals *ModeReq* and *ModeAck* of the two-phase channel *ChMode2*;
- 3 pins to implement the *Scan-in* channel and 3 pins for the channel along which the test results are observed.

The number of extra pins for the implementation of the *Scan-in* channel and the result observation channel can be reduced by sharing some of the system channels using select blocks and multiplexers. As a result, the number of extra pins is reduced to 2.

8.5.2 Built-in self test design

The scan-in block in the BIST version of the block sorter was replaced by the asynchronous pseudo-random pattern generator based on the 16-bit LFSR. The following derivation polynomial was used to design the LFSR:

$$f(X) = 1 + X^3 + X^4 + X^5 + X^{16}.$$

Although the generator produces a new test vector at the time when a new request arrives at its input the generation time of a random test vector is less than the time required to shift a test into the scan-in block which operates autonomously. The simulation results shows that the application time of a pair of random test vectors to each sorting cell is 92ns which is 2.73 times less than that of the scan testable version. The area overhead of the BIST design is 15.4% compared with the original version of the block sorter. In test mode, the average dynamic power consumption of the BIST sorter is equal to 30nJ per pair of tests.

The maximum number of extra pins required to implement the BIST design is 5 since there is no need for the *Scan-in* channel. Thus, the minimum number of extra pins is equal to 2. Simulation results of the BIST version of the block sorter can be found in Table 8.1.

Table 8.1: *Simulation results and cost comparisons*

Block Sorter	NT[a] k	T1[b] ns	PD[c] %	T2[d] ns	No. pairs of tests	SA[e] mm^2	AO[f] %	NEP[g]		PC[h] nJ	
								Max	Min	NM	TM
Original	109.1	66.7	n/a	n/a	n/a	14.50	-	-	-	26.5	-
Scan test	126.7	69.7	4	251	18	16.81	15.9	8	2	28.4	27.8
BIST	126.5	69.7	4	92	1000	16.77	15.7	5	2	28.4	30.0

a. NT is the number of transistors
b. T1 is the minimum application time of one input vector in normal operation mode
c. PD is the performance degradation
d. T2 is the minimum application time of a pair of test vectors in test mode
e. SA is the silicon area
f. AO is the area overhead
g. NEP is the number of extra pins
h. PC is the average power consumption per test in normal mode (NM) and test mode (TM)

8.6 Summary

It has been demonstrated how the asynchronous scan testing and BIST techniques can be applied to design testable asynchronous circuits with regular structures. A case study of an asynchronous block sorter has been presented. The sorter has been designed using handshake components and post-optimized to achieve a minimal silicon area after its compilation. The structural regularity of the block sorter makes it possible to test all of its sorting cells in parallel sharing a common source of test vectors. This increases the test performance of the circuit. The operation mode of the testable block sorter is changed asynchronously using a two-phase communication channel.

Single stuck-at output faults on the control paths and all single stuck-at faults on the data paths of the sorter have been considered. The block sorter is tested for faults on its control lines during normal operation since these faults cause the whole circuit to halt. Stuck-at faults on the control wires, which are not used in test mode, and on the data paths along which the data is transferred are detectable in normal mode by applying two blocks of test vectors. The test mode of the block sorter has been designed to test stuck-at faults inside the comparators of the sorting cells and to reduce the test application time.

Two different test circuits have been described. Each testable implementation of the block sorter imposes different penalties. Compared to the scan testable block sorter, the BIST version demonstrates approximately the same area overhead and average power consumption per test, fewer extra pins and a lower application time for a pair of tests. However, the number of random tests which must be applied to the test inputs of each sorting cell is much higher than the number required for scan testing. As a consequence, the total test application time is higher when random testing is used.

Fault simulation results reveal 100% testability of both single stuck-at output faults at the high level representation of the block sorter and all stuck-at faults inside data processing blocks of its sorting cells.

Chapter 9 : Conclusions and Future Work

9.1 Conclusions

A resurgence of interest in asynchronous VLSI circuits has been engendered by their potential ability to eliminate the clock skew problem, achieve average case performance, provide component modularity and to reduce power consumption.

Two asynchronous design methodologies have been considered in this thesis: micropipelines and handshake circuits. The micropipeline approach creates a powerful design framework for implementing complex asynchronous circuits such as microprocessors. Handshake circuits can be designed easily from their high level specifications written in the Tangram language.

Major silicon producers such as *Intel, Philips, Sun Microsystems*, etc. have already started to use asynchronous design methodologies in order to build chips with new properties which could compete in a market traditionally dominated by synchronous circuits [Berk94, SproTR94]. However, the testability issues of asynchronous VLSI circuits must be addressed before their commercial potential can be realized. Traditional fault models such as stuck-at and stuck-open models can be used to describe the fault behaviour of an asynchronous circuit.

The testing of asynchronous circuits is aggravated by the following main factors:

- the presence of a large number of state holding elements in asynchronous circuits makes test generation harder;

- the absence of any synchronization clock decreases the controllability of the states of the asynchronous circuit;
- logic redundancy introduced into the asynchronous design in order to ensure its hazard-free behaviour makes the detection of some faults difficult or even impossible.

Asynchronous DFT techniques are required to reduce the otherwise high costs of test generation and test application and to increase the coverage of faults in asynchronous VLSI circuits. Before choosing a particular DFT method one must estimate:

- the impact on the original design (increase in silicon area, effect on performing calculations, testability of extra logic, etc.);
- the complexity of the DFT implementation;
- the effect on test pattern generation (reduction in CPU time, improved fault coverage, etc.).

One promising application area for asynchronous VLSI circuits is in portable designs since they have a potential for low power consumption. As was shown in this thesis the DFT methodology and design for low power are in direct conflict, i.e., more testable circuits dissipate more power. The resolution of this lies in the separation of the circuit's normal operation and test modes. However, the presence of extra logic elements incorporated into the testable circuit design increases its overall power dissipation in normal operation mode. As a result, the dynamic power dissipation of the testable design must be taken into account when the circuit operates in its normal operation mode.

Testable CMOS designs for symmetric and asymmetric C-elements have been considered in this thesis. The proposed structures for C-elements provide for the detection of line stuck-at faults and transistor stuck-open faults. The testable CMOS implementations of the C-elements allow them to operate as:

- an AND or OR gate for the symmetric C-element;

Table 9.1: *Testability of micropipeline structures*

Micropipeline design	Control paths		Data paths
	DFT is not required for:	DFT is required for:	DFT is required for:
Two-phase	stuck-at input and output faults	-	1. Stuck-at input and output faults. 2. Delay faults.
Four-phase (simple latch control)	stuck-at input and output faults	-	1. Stuck-at input and output faults. 2. Delay faults.
Four-phase (semi-decoupled latch control)	stuck-at output faults	stuck-at input faults	1. Stuck-at input and output faults. 2. Delay faults.

- an AND gate or repeater for the OR-AND type of asymmetric C-element;
- an OR gate or repeater for the AND-OR type of asymmetric C-element.

Thus, the use of these C-elements reduces the test complexity of the asynchronous circuit by changing the sequential functions of the C-elements into combinational ones during the test. A scan testable CMOS implementation of the symmetric C-element has also been discussed.

The scan test technique is widely used to design testable synchronous circuits. In this thesis, the scan test methodology has been used to design micropipelines for testability. The stuck-at fault and delay fault models were considered. Table 9.1 shows comparison results for the testability of different micropipeline structures. Testing for stuck-at output faults in the control parts of micropipelines is trivial since they cause the faulty micropipeline to halt while it performs in normal operation mode. Stuck-at input faults in the control circuits of either two-phase micropipeline or four phase micropipeline with simple latch control can be detected in normal operation mode. However, the testing for stuck-at input faults in the control part of the four-phase micropipeline with semi-decoupled latch control requires the use of a special DFT technique presented in this thesis.

Some stuck-at faults on the inputs of asymmetric C-elements used in four-phase designs cause premature firings on their outputs which are difficult to detect. These faults can be identified by

- checking the order of events on the control lines of the circuit under test;
- applying a special test sequence of events to the control inputs;
- converting the asymmetric C-element into a symmetric one during the test.

Test vectors for detecting stuck-at faults in the combinational circuits of the micropipelines can be derived using any known test generation technique. Testing for delay faults in the data paths is important since they violate the bundled-data interface between the stages of the micropipeline.

Asynchronous designs for a pseudo-random pattern generator and signature analyser which operate according to either two-phase or four-phase transition signalling have been proposed.

Two structural (parallel and bit-serial) designs for random pattern testability of asynchronous sequential circuits have been described. The single stuck-at fault model has been considered. The testable sequential circuit designs have been implemented as two-phase and four-phase circuits. In addition, handshake implementations of testable sequential circuits have been discussed. The test complexity of sequential circuits designed using these DFT techniques is reduced to the complexity of testing just their combinational parts. Stuck-at faults in the control part of the testable two-phase sequential circuit cause the faulty circuit to halt in normal operation or test mode which is easy to detect. A technique for testing stuck-at faults in the control of the four-phase sequential circuit has been presented.

An asynchronous implementation of the BILBO technique has been demonstrated on a micropipeline design with BIST features. The micropipeline can operate following either two-phase or four-phase transition signalling. The proposed BIST micropipeline allows single stuck-at faults and delay faults to be detected in its combinational logic blocks during the test.

A testability analysis have been carried out on designs of an asynchronous adder implemented using single-rail, dual-rail and combined single-rail and dual-rail (hybrid) data encoding techniques. Each adder design brings different trade-offs between testability for stuck-at faults, performance and area overhead.

It has been observed that stuck-at faults in the control part of the single-rail adder can cause the adder to halt or cause premature and delayed firings. The detection of premature and delayed firings requires the adder to be set into a special test mode.

The hybrid implementation of an asynchronous adder does not require a special test mode since all its stuck-at faults can be detected in normal operation mode and it demonstrates a reasonable area overhead and performance degradation.

The scan test and BIST versions of an asynchronous block sorter designed as a handshake circuit have been investigated. The testable implementations of the block sorter require a low area overhead and exhibit 100% testability for stuck-at faults in its high level representation. A novel technique for changing the state of the sorter has been described.

9.2 Future work

The results presented in this thesis have revealed difficulties in the testing of asynchronous circuits which create the ground for new research in this area.

9.2.1 Testing control circuits

Automatic test generation for stuck-at input faults in quasi-delay insensitive and speed-independent control circuits of high complexity is still a problem.

Also, there is the problem of identifying stuck-at input faults which cause premature firings in a circuit. Creating software which, when applied to the circuit during its design, could identify this kind of fault seems to be an important task. As a result of using this software the regions in the circuit which have these faults could be identified at an early stage of the design process. Thus, the identified parts of the circuit could be redesigned in order to eliminate the hard-to-detect faults.

In addition, an investigation of possible DFT techniques which could facilitate the testing for bridging and transistor stuck-open faults in order to increase the fault coverage for fabrication faults in asynchronous control circuits seems to be an important field of research. Testable CMOS designs of C-elements described in this thesis can be used to detect these faults.

Testing asynchronous control circuits specified at high level using languages such as *communicating sequential processes* (CSP) [Mart90] or graphs such as *signal transition graphs* (STG) [Chu87] is another interesting field of future research.

9.2.2 Testing microprocessors

Two generations of asynchronous RISC processor (AMULET1 and AMULET2) have been designed by the AMULET research group. Problems related to the testing of asynchronous circuits have been largely ignored. As a consequence, the commercial usefulness of the AMULET designs, which requires effective test procedures to be applied to chips manufactured in high volume, remains low.

I believe that the design of a testable asynchronous microprocessor will bring the next generation of AMULET design to the stage where industrial potential together with academic research can make a breakthrough in the low power microprocessor market.

Appendix A : Testing of Synchronous VLSI Circuits

A.1 Test generation methods

A test procedure includes three basic steps: test generation, test application and response evaluation. During test generation input patterns are derived to sensitize faults and propagate fault effects to the observable outputs of the circuit under test. Test generation techniques can be divided into two major groups: algorithmic test generation methods and random (pseudo-random) testing.

A.1.1 Algorithmic test generation

Most of the reported test generation methods produce test sequences by means of analysing the topological structure of the circuit under test. Such well-known path sensitization algorithms as the D-algorithm [Lala85, Feug88], PODEM [Ben84] and FAN [Chen89] have successfully been used for automatic test generation for VLSI circuits.

The concept of the path sensitization technique is illustrated in Figure A.1. The test derivation algorithm for a stuck-at-1 fault on line *d* includes three sequential steps marked by circled numbers. During the first step line *d* is set to low. On the second step, a low level signal is applied to input *a* to justify the previous step. On the last step, the fault effect of the stuck-at-1 fault on line *d* is propagated to output *e* of the circuit by setting line *b* to low. As a result, test pattern 00 is derived to detect the stuck-at-1 fault on line *d*. Note that in Figure A.1 the logic value before the slash is the correct value for the fault-free circuit; the value after the slash is the fault response of the circuit in the presence of the stuck-at fault.

As shown above, the process of path sensitization consists of three basic operations: justification, implication and propagation [Chen89]. Step 2 of the above example is justification for generating a logic 0 on node *d*. In general, when a value is assigned to a certain node, it may imply other signals for some lines of the circuit. The aim of the implication procedure is to cause forward propagation of the result of the justification step. For example, a low signal can be set on line *d* by setting a one on line *b* (see Figure A.1). In this case the effect of the fault on line *d* cannot be propagated through the NOR gate to output *e*. Therefore, the result of the justification step can be propagated only if a low signal is applied to input *a.* The effect of the propagation process (step 3) is to move the fault effect through a sensitized path to an output of the circuit.

The D-algorithm is one of the classical test generation methods which derives tests for detecting stuck-at faults at the gate level circuit representation [Lala85, Chen89]. The set of five elements $\{0, 1, X, D, \overline{D}\}$ for describing signals is used to facilitate the path sensitization process: X means unknown, D represents a signal which is one in the fault-free circuit and zero in a faulty circuit; $\overline{D}$ is the complement of D. The D-algorithm consists of three parts: fault excitation and forward implication, D-propagation, backward justification. On the first step the minimal input conditions are selected in order to produce an error signal ($\overline{D}$ or D) on the output (faulty node) of the logic element. The forward implication process is performed in order to determine the outputs of those gates whose inputs are specified. The goal of the D-propagation step is to propagate the fault effect to primary outputs by means of assigning logical values to corre-

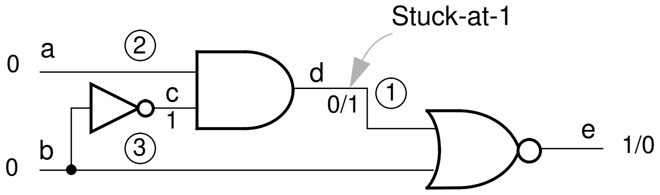


Figure A.1: Path sensitization technique sponding internal lines and primary inputs. In backward justification, node values are justified from primary inputs. If there is a conflict in one of the nodes the backwards consideration from the conflict node to the primary inputs is reiterated until the fault effect ($\overline{D}$ or D) reaches at least one of the primary outputs.

Not all the stuck-at faults inside the circuit can be detected by path sensitization algorithms. Logic redundancy is the reason why these faults cannot be detected. For example, a stuck-at-1 fault on node *c* of the circuit shown in Figure A.1 is undetectable since there is no sensitization path from the fault site to output *e*. As a result, in the presence of the redundant stuck-at-1 fault on node *c* the faulty circuit produces correct results. Clearly, the combinational circuit shown in Figure A.1 performs according to the following Boolean function: $\overline{a \cdot \bar{b} + b}$. This function is redundant and equivalent to $\overline{a + b}$ which is not redundant.

Thus, algorithmic test generation techniques allow both the generation of test vectors for the detection of stuck-at faults in the gate-level representation of VLSI circuits and the identification of redundant faults which are not detected by logic testing.

A.1.2 Random pattern testing

In random testing methods input patterns can be generated by a linear feedback shift register (LFSR), a shift register with connections from some of the stages to the input of the first element through an XOR gate [Need91]. A LFSR allows the generation of long repetitive pseudo-random sequences, segments of which have properties similar to random patterns [Davi81, Wag87]. Pseudo-random sequences are more suitable for testing digital circuits than truly random ones due to the possibility of repeating them for simulation purposes.

LFSRs which generate pseudo-random sequences of maximal length can be described by their primitive polynomials. If n is the number of stages of the maximum-length LFSR then the length of its output sequence is equal to $M = 2^n - 1$. Tables of primitive polynomials can be found elsewhere [MClus86, Russ89].

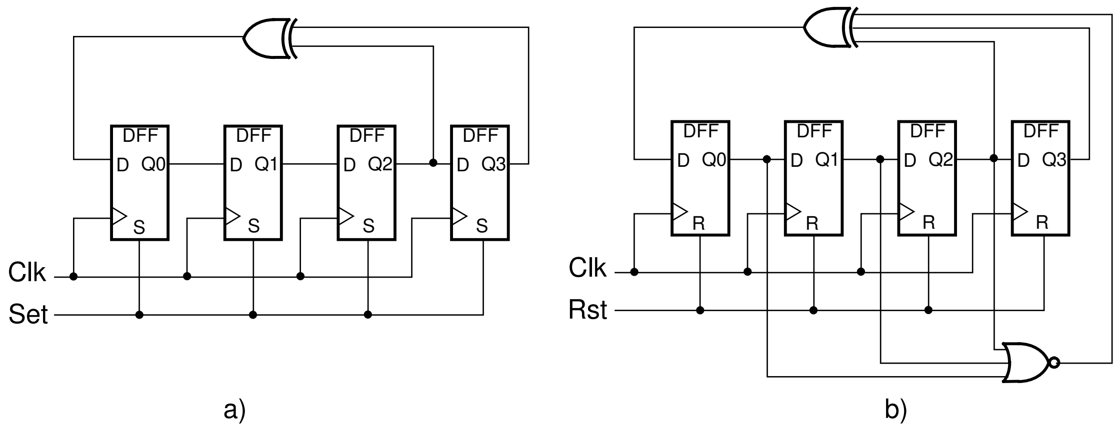


Figure A.2: *Four-bit pseudo-random pattern generator*

Figure A.2a shows a four-bit LFSR which 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. The initial state of the LFSR is $(Q_0, Q_1, Q_2, Q_3) = (1, 1, 1, 1)$. Note that the state of 'all zeros' is illegal for the LFSR shown in Figure A.2a since it stays in this state forever.

It is well known that for 100% testing of combinational circuits all binary input combinations must be applied to their inputs. This approach is called exhaustive testing. Figure A.2b illustrates a modified version of the four-bit LFSR which cycles through all possible 16 states including the 'all zeros' state. The use of the extra NOR gate forces the LFSR to go through all its original states plus the 'all zeros' state which is produced by the LFSR after passing the state 0001. The output sequence generated by such an LFSR is shown in Table A.1. A modified version of a maximal-length LFSR can be used for exhaustive testing [MClus86].

A.2 Response evaluation techniques

The main goal of response evaluation is to detect any wrong output of the circuit under test. There are two basic approaches for achieving this goal: a good response generator and compact testing.

Table A.1: *State sequence of the four-bit LFSR shown in Figure A.2b*

State	Q_0	Q_1	Q_2	Q_3	State	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0	9	1	0	1	0
1	1	0	0	0	10	1	1	0	1
2	0	1	0	0	11	1	1	1	0
3	0	0	1	0	12	1	1	1	1
4	1	0	0	1	13	0	1	1	1
5	1	1	0	0	14	0	0	1	1
6	0	1	1	0	15	0	0	0	1
7	1	0	1	1	16	0	0	0	0
8	0	1	0	1	17	1	0	0	0

A.2.1 Good response generation

Any faulty response can be detected by comparing good responses with output signals produced by the test object. For instance, all good responses can be stored in a ROM. After applying each test pattern to the circuit under test its output is compared with the good one. If they are different the comparator will activate an error signal. Good responses can easily be obtained by means of software-simulation of the VLSI circuit as a part of the design verification stage [Russ89]. In comparison test technique, test patterns are applied to the inputs of the circuit under test and a golden unit simultaneously and the responses of both units are compared by the comparator.

A.2.2 Signature analysis

The main drawback of the response evaluation technique is the necessity to operate with a large amount of output data during the testing of a VLSI circuit. In compact testing, the output data is compressed into a compact form during the test. After the test is complete the response of the circuit under test is compared with the compressed response of the golden unit.

The most widely used compact testing method is signature analysis [Lala85, Russ89]. In the signature analysis technique, the output data are compressed using a signature analyser built using a LFSR. The first practical realization of the signature analysis tech-

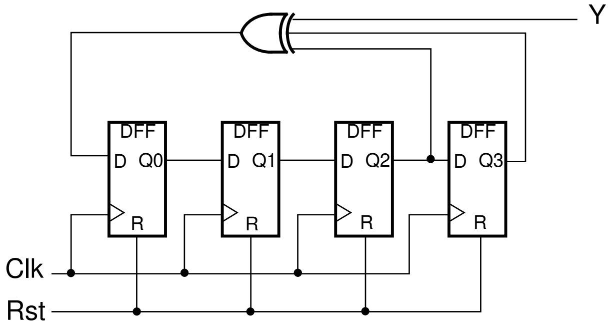


Figure A.3: *Four-stage serial signature analyser*

nique as a method of detecting errors in output data steams produced by digital circuits was pioneered by Hewlett Packard Ltd. [Hew77]. The compact forms produced by signature analysis are called signatures.

In compact testing, two kinds of signature analysers are used: serial and parallel signature analysers [MClus86]. A four-stage serial signature analyser and a four-stage parallel signature analyser are shown in Figure A.3 and Figure A.4 respectively. A serial signature analyser treats only one response (*Y*) bit at every clock whereas a parallel signature analyser compacts responses from multiple output streams (*Y0*, *Y1*, *Y2*, *Y3*).

The probability of a signature analyser failing to detect an error can be evaluated as $P \approx 1/2^n$, where n is the number of flip-flops of the LFSR [Hew77, MClus86]. For the 16-bit signature analyser used by Hewlett Packard $P = 1.5\times10^{-5}$. This confirms the high quality of the signature analysis technique.

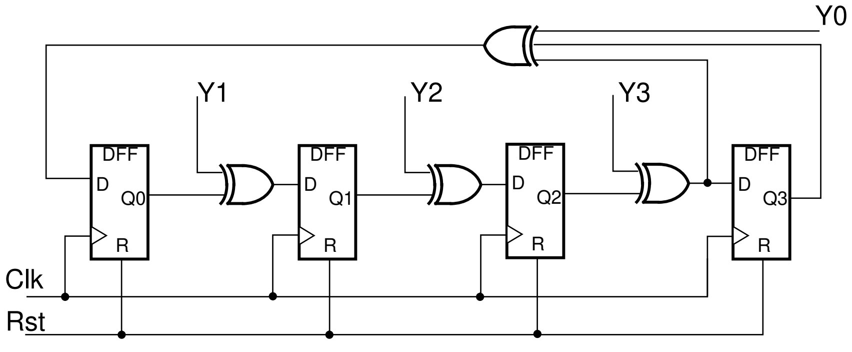


Figure A.4: *Four-stage parallel signature analyser*

Appendix B : Design for Testability of Synchronous VLSI Circuits

B.1 What is design for testability?

Design for testability (DFT) can be defined as a design philosophy that leads to decreasing the cost of testing digital circuits and to increasing the fault coverage or fault isolation.

There are two key concepts in DFT techniques: controllability and observability [Lala85, MClus86, Cort87, Russ89, Turin90]. Controllability refers to the ease of producing certain logic values on internal nodes of the circuit via its primary inputs. Observability refers to the ease with which the values of the circuit nodes can be determined via its primary outputs. The degree of controllability of the circuit can be increased by means of incorporating in it some additional logic elements and control terminals. The easiest way to increase observability is to add some extra output terminals. DFT techniques can be divided into three major groups: ad hoc strategies, structured approaches and built-in self-test techniques [Russ89].

B.2 Ad-hoc techniques

The ad-hoc strategy is used to help designers of VLSI circuits to alleviate testing problems. Test engineers, using their experience, have developed a number of recommendations for enhancing the testability of VLSI circuits. Practical guidelines for designing testable circuits can be found elsewhere [Ben84, Turin90]. All these recommendations

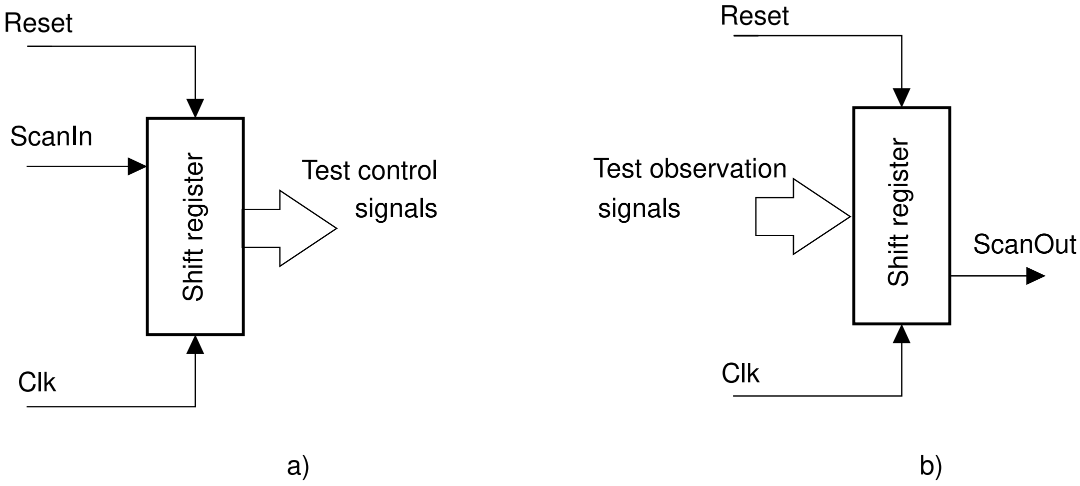


Figure B.1: *Using shift registers for improving (a) control access; (b) observation access*

can be divided into two groups: guidelines which 1) make test pattern generation easier; 2) simplify test application and fault isolation.

It is known that primary access to subcircuits of a VLSI design is extremely limited. In this case the use of multiplexers and demultiplexers can improve the controllability and observability characteristics of the VLSI circuit. Demultiplexers and multiplexers incorporated into the VLSI circuit allow the test engineer to manipulate the data streams inside the circuit using a special mode of operation (test mode).

Shift registers can be used to make internal nodes in the VLSI circuit more accessible either for controllability or observability as shown in Figure B.1. A serial-in, parallel-out shift register is used to set the circuit into a predefined state (see Figure B.1a). Figure B.1b shows a parallel-in, serial-out shift register which is used to store test information from internal nodes and to scan it out to a primary output of the VLSI circuit.

The addition of extra gates to block signal paths can be used to partition a VLSI design into smaller subcircuits, provide facilities to break feedback paths, break up long counter chains and provide initialisation of stored-state devices for simplifying test generation. This technique has been described in detail [Ben84].

B.3 Structural DFT approaches

Structural DFT techniques ensure the desired degree of controllability and observability at the structural level of a VLSI circuit. Most of these structured approaches rely on the concept that, if the latch variables can be controllable and observable within a sequential circuit, then the test generation problem can be reduced to the testing of just the combinational logic. There are four main formal structural DFT methods: the scan path [MClus86, Russ89], the level-sensitive scan design (LSSD) [Rice82, Lala85, Russ89], the scan/set method [Russ89] and the random access scan technique [Ben84, MClus86]. In this Section the scan path and the LSSD design styles will be considered.

B.3.1 Scan path

The scan path approach assumes that during the test all the memory elements of the sequential circuit are configured into a long shift register called the scan path. All the memory elements of the circuit can be controlled and observed by means of shifting in and shifting out test data along the path. The selection of the input source for the storage elements can be achieved using multiplexed data flip-flops [MClus86] or two-port flip-flops with two data inputs and two clocks [Russ89].

A scan path technique can be used to partition a VLSI structure into a number of less complex subcircuits by organizing the scan path to pass through a number of combinational networks. The sequential depth of such a circuit is much less than the depth of the original one which alleviates the test problem considerably. To test the scan path itself, flush and shift tests are applied. The flush test consists of all zeros and all ones. The shift test exercises the memory elements of the scan path through all of their possible combinations of initial and next states.

B.3.2 Level-sensitive scan design

The LSSD design approach is based on two main concepts: level sensitivity, and a scan path. The first assumes that: 1) all changes in the circuit are controlled by the level of a

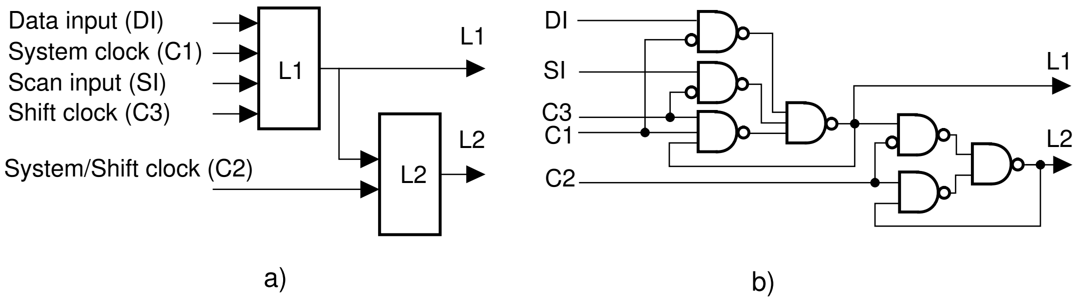


Figure B.2: *Polarity hold latch a) symbolic representation; b) implementation in NAND gates*

clock signal; 2) the steady state response of the sequential circuit to an input state change is independent of the rise and fall times and propagation delays of signals within the circuit. The second concept of the LSSD technique assumes that the circuit must incorporate a scan path.

Shift register latches (SRL) are used to implement all memory elements in LSSD circuits. Figure B.2 shows the symbolic representation of an SRL and its implementation in NAND gates. In the normal mode of operation clock *C3* is not activated and clock *C1* is used to write data to latch *L1*. Output data can be taken from *L1* or, if clock *C2* is used, from *L2*. In test mode non-overlapping clocks *C3* and *C2* are used to shift data from output *L2* of the previous SRL into latch *L1* (clock *C3*) with consequent copying of the data from output *L1* into latch *L2* (clock *C2*).

The basic LSSD configuration is illustrated in Figure B.3. In this structure the pair of non-overlapping clocks *C1* and *C2* is used to store the system data from the combinational logic (CL) in the SRLs (normal operation mode). In test mode two sequences of clocks *C3* and *C2* are applied to control and observe the states of all the SRLs by means of transmitting the test data through the scan path (dotted line). Note that both the *L1* and *L2* latches participate in the system function and during the test.

The basic algorithm for testing with the LSSD structure shown in Figure B.3 can be written as follows:

1) Verify the operation of all the SRLs by applying flush and shift tests.

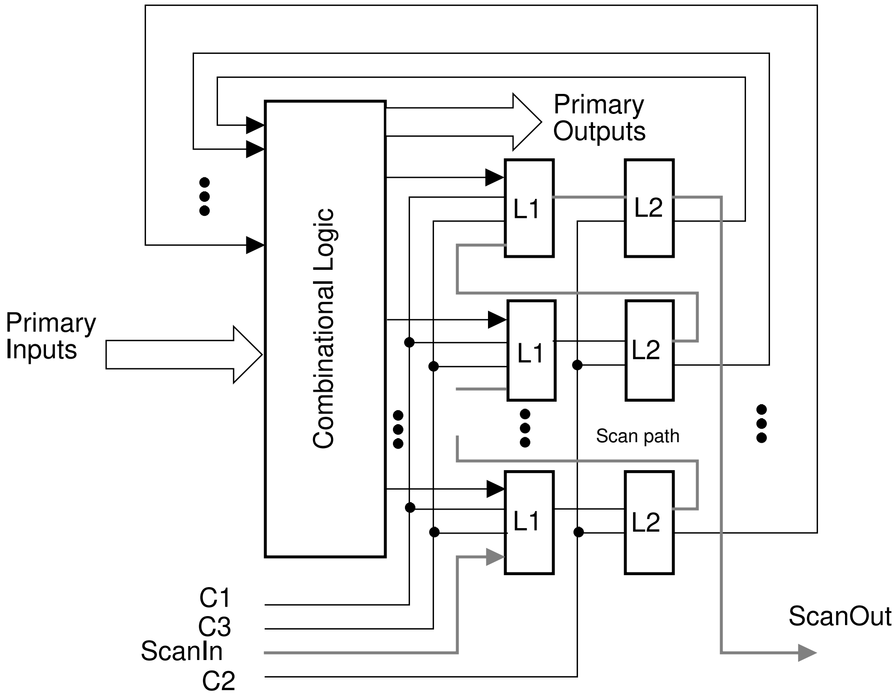


Figure B.3: *LSSD structure*

2) Load a test into the SRLs. The test is loaded from the scan-in port of the circuit and shifted in serially by means of clocks *C3* and *C2* alternatively activated.

3) Generate a test pattern on the primary inputs of the circuit and turn clock *C1* on and off. As a result, the response of the combinational network is stored in the *L1* latches.

4) Pulse the system clock *C2* to rewrite the contents of the *L1* latches into the *L2* latches.

5) Pulse two sequences of clocks *C3* and *C2* to scan out the contents of the SRLs. Meanwhile a new test pattern can be loaded into the SRLs.

The test procedure described above is continued until the combinational logic has been tested. The responses of the circuit are observed at the primary outputs and the scan-out port. Test generation can be fully automatic since the tests must be produced just for the combinational part of the LSSD circuit.

B.4 Built-in self-test

In built-in self-test (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 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.

Built-in logic block observer

A classical example of InSitu self-test is the built-in logic block observation (BILBO) technique [Lala85, MClus86]. 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 structure of a basic 4-bit BILBO element is shown in Figure B.4. The function of the BILBO element is controlled by lines *B1* and *B2*. The storage elements are *D*-type flip-flops. The inputs of the BILBO element are usually fed by the outputs of the preceding combinational circuit, the outputs are connected to the inputs of the following combinational network.

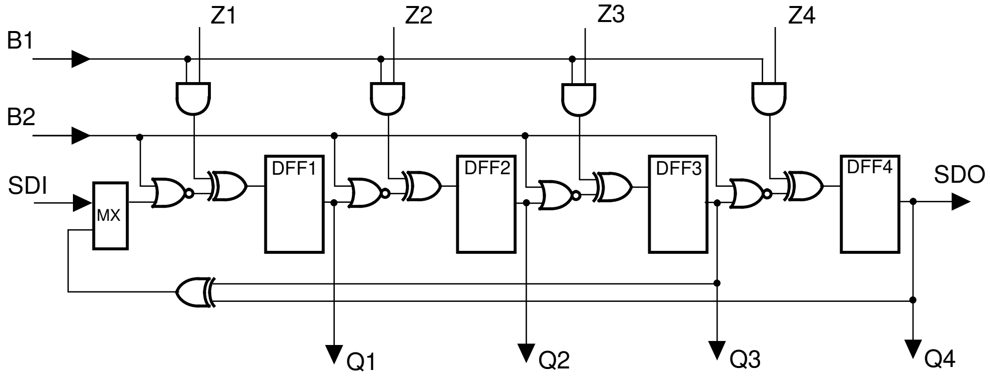


Figure B.4: *Basic BILBO element*

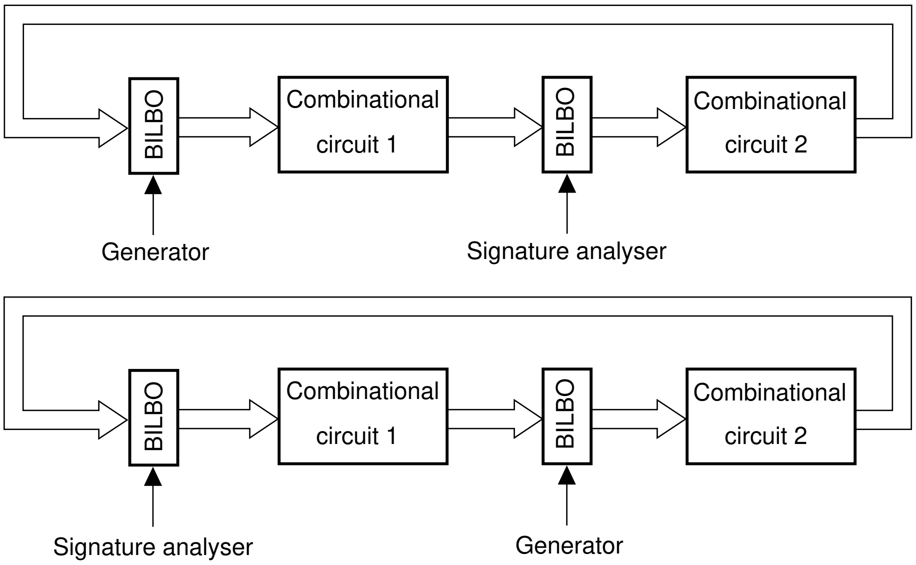


Figure B.5: *Self-testing structure with BILBOs*

The BILBO register can perform in four modes:

1. *B1*=*B2*=1, *System operation mode*. The BILBO is configured as a set of *D* flip-flops to store the system states of a VLSI circuit.

2. *B1*=*B2*=0, *Shift register mode*. The BILBO functions as a long shift register forming a scan path.

3. *B1*=1, *B2*=0, *LFSR with multiple inputs*. If all the inputs of the BILBO are fixed the BILBO element shown in Figure B.4 is configured into a 4-stage pseudo-random pattern generator (see Figure A.2). Otherwise the BILBO functions as the 4-bit parallel signature analyser as illustrated in Figure A.4.

4. *B1*=0, *B2*=1, *Reset mode*. The BILBO register is reset.

Figure B.5 shows how the BILBO technique can be used to test a VLSI circuit. Initially one BILBO register works as a generator to stimulate the combinational circuit to be tested. The second BILBO is used as a signature analyser to compress the responses of the circuit under test. After a certain number of clocks the BILBO register that contained the signature is reconfigured into a scan path register and the content is shifted out to compare with the signature of the golden unit. The roles of the BILBOs are reversed to test the next combinational circuit.

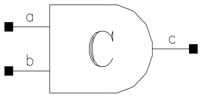
Appendix C : Testable Asynchronous Cells

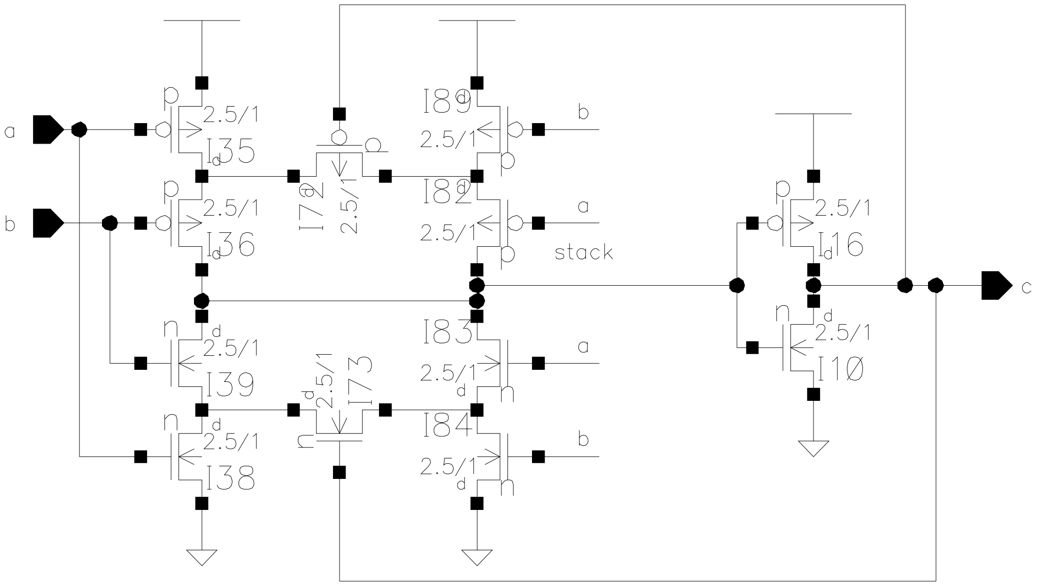
This Appendix contains symbols, schematics and layouts for some cells designed for a testable asynchronous library. The presented designs have been implemented in CMOS technology on a $1\mu m$ double layer metal process using *Cadence* design tools and simulated using *SPICE*. In addition, nodal capacitances for inputs and outputs of the testable cells calculated from their extracted layouts can be found in this Appendix.

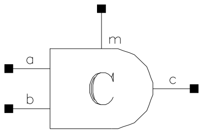
Table C.1 contains the names of the testable cells and their meanings with the corresponding figures where they can be found in this thesis.

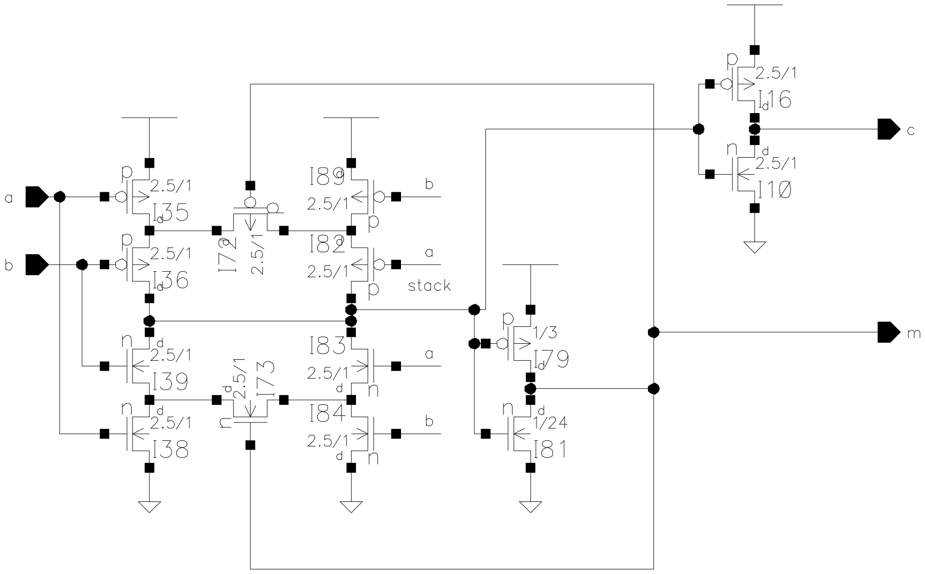
Table C.1: *Names of the testable cells and their meanings*

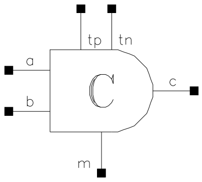
Cell name	Meaning	Figure
MullerC2_stat	Symmetric C-element	4.1b
tst_C2	Symmetric C-element testable for stuck-open faults	4.2
tst_safC2	Symmetric C-element testable for stuck-at faults	4.3
static_asy_mullc_nb	OR-AND type asymmetric C-element	4.4
static_asy_mullc_pb	AND-OR type asymmetric C-element	4.5
tst_stat_asyC2_nb	OR-AND type asymmetric C-element testable for stuck-open faults	4.6a
tst_stat_asyC2_pb	AND-OR type asymmetric C-element testable for stuck-open faults	4.6b
tst_saf_asyC2_nb	OR-AND type asymmetric C-element testable for stuck-at faults	4.7
tst_scanC2	Symmetric C-element with scan features	4.8
scan_latch	Scan latch	5.4

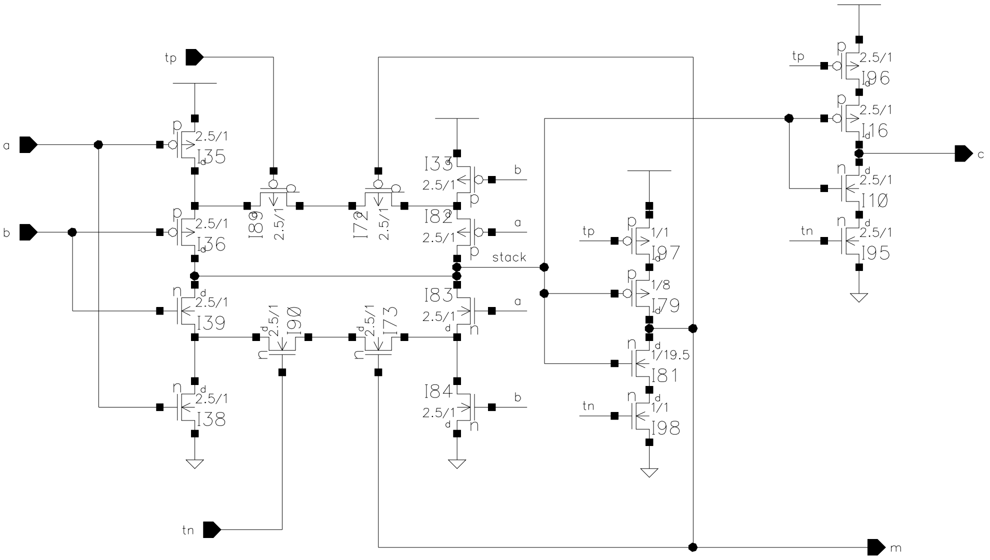

a
b
C
c

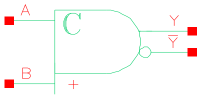

a
b
c
p
2.5/1
I35
I36
I39
I38
I72
I73
I89
I82
I83
I84
I16
I10
n
d
stack

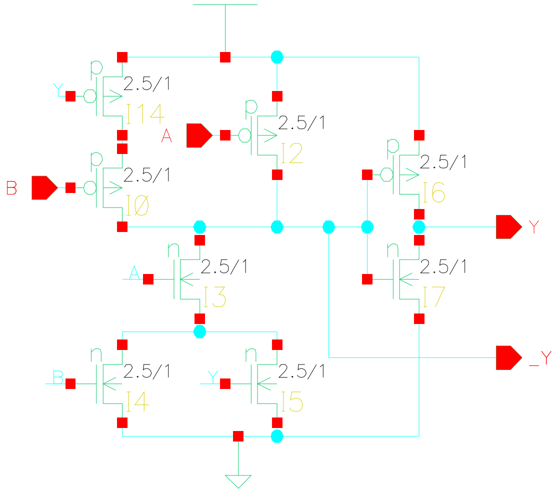

m
a
b
C
c

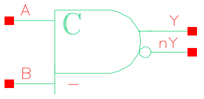

a
b
p
2.5/1
I35
I36
n
I39
I38
I72
I73
I89
I82
I83
I84
b
a
a
b
stack
1/3
I79
1/24
I81
I16
I10
c
m

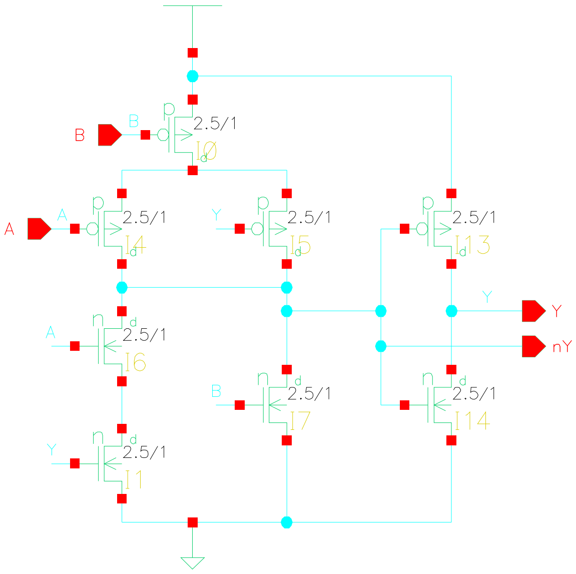

tp
tn
a
b
C
c
m

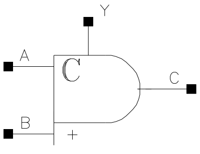

tp
a
b
tn
m
c
stack
I35
I36
I39
I38
I89
I72
I90
I73
I33
I82
I83
I84
I97
I79
I81
I98
I96
I16
I10
I95
2.5/1
1/1
1/8
1/19.5

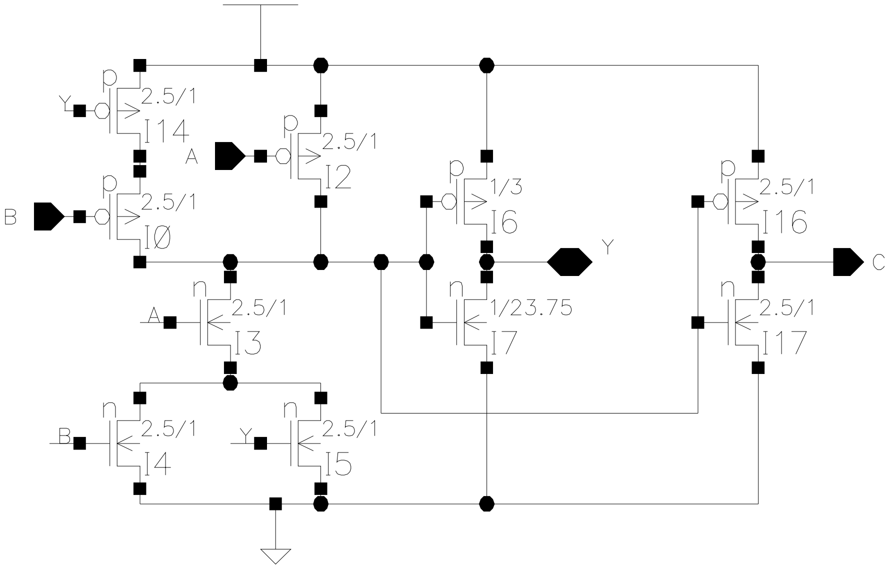

A
C
Y
$\overline{Y}$
B
+

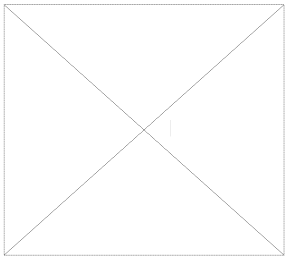

p
2.5/1
I14
Y
A
p
2.5/1
I2
B
p
2.5/1
I0
p
2.5/1
I6
Y
n
2.5/1
A
I3
n
2.5/1
I7
_Y
n
2.5/1
B
I4
n
2.5/1
Y
I5

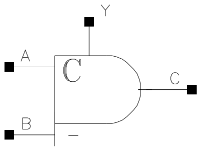

A
C
B
Y
nY

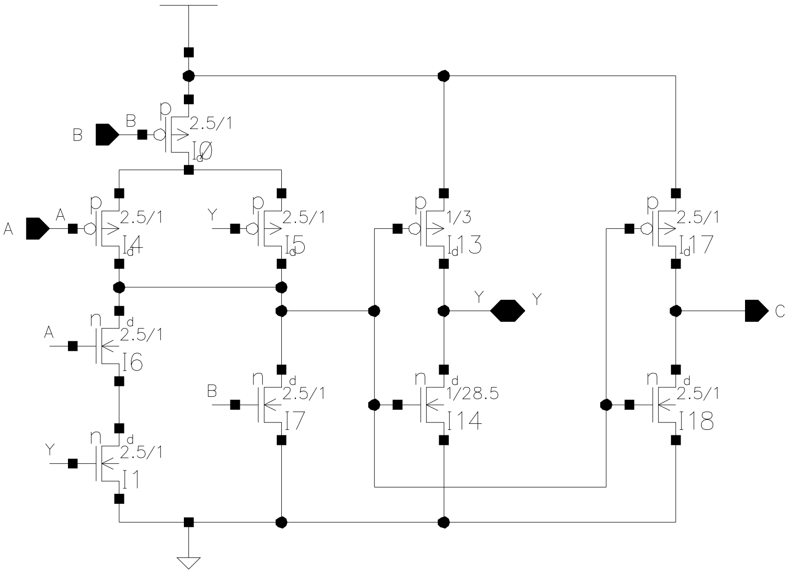

B
p
2.5/1
I0
A
I4
Y
I5
I13
n
I6
I7
I14
I1
d
Y
nY

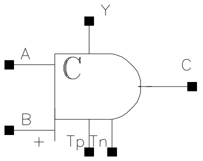

Y
A
C
C
B
+

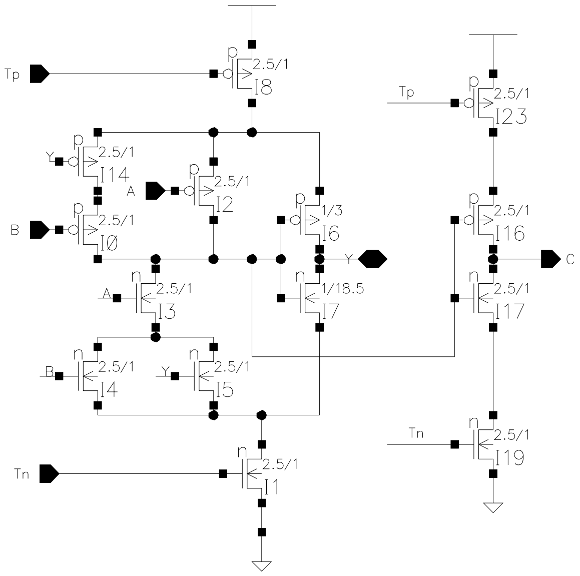

p
2.5/1
I14
Y
B
p
2.5/1
I0
A
p
2.5/1
I2
p
1/3
I6
Y
n
1/23.75
I7
p
2.5/1
I16
C
n
2.5/1
I17
n
2.5/1
A
I3
n
2.5/1
B
I4
n
2.5/1
Y
I5

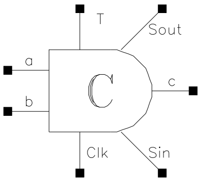

I

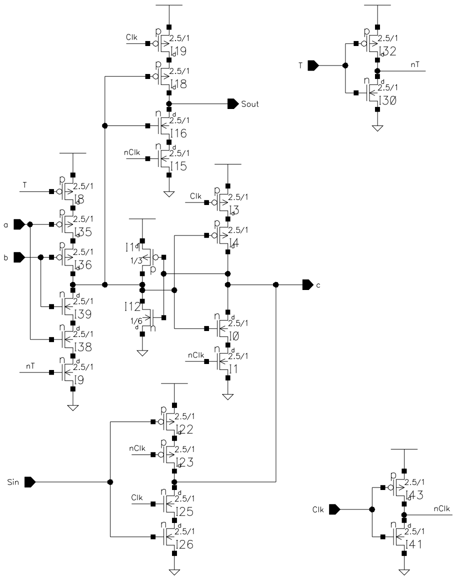

Y
A
C
C
B
−

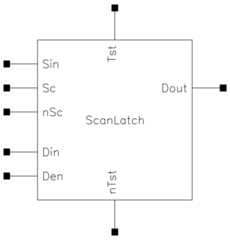

B
B
p
2.5/1
I0
A
A
p
2.5/1
I4
Y
p
2.5/1
I5
p
1/3
I13
p
2.5/1
I17
Y
Y
C
A
n
2.5/1
I6
B
n
2.5/1
I7
n
1/28.5
I14
n
2.5/1
I18
Y
n
2.5/1
I1

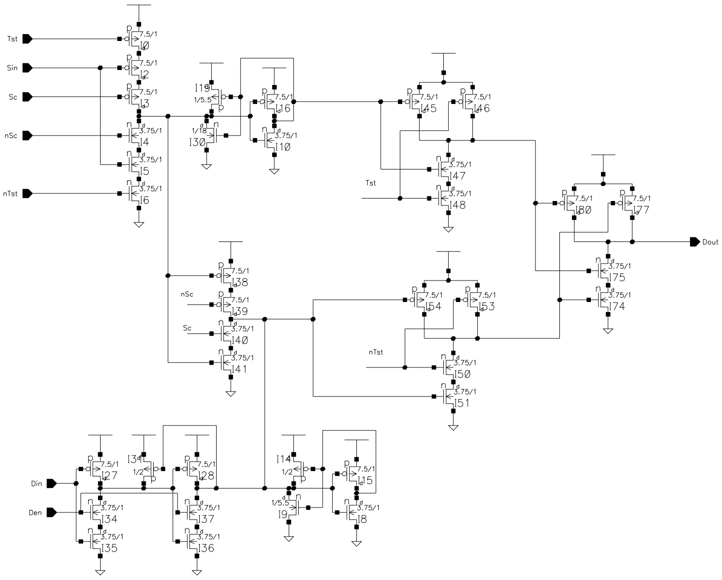

Y
A
C
C
B
+
Tp
Tn


Tp
p
2.5/1
I8
Y
p
2.5/1
I14
A
p
2.5/1
I2
B
p
2.5/1
I0
p
1/3
I6
Y
n
1/18.5
I7
A
n
2.5/1
I3
B
n
2.5/1
I4
Y
n
2.5/1
I5
Tn
n
2.5/1
I1
Tp
p
2.5/1
I23
p
2.5/1
I16
C
n
2.5/1
I17
Tn
n
2.5/1
I19


T
Sout
a
C
c
b
Clk
Sin


Clk
I19
I18
Sout
I16
nClk
I15
T
I8
a
I35
b
I36
I39
I38
nT
I9
I11
1/3
I12
1/6
Clk
I3
I4
c
I0
nClk
I1
I22
nClk
I23
Sin
Clk
I25
I26
T
I32
nT
I30
Clk
I43
nClk
I41
2.5/1


Tst
Sin
Sc
nSc
Din
Den
Dout
ScanLatch
nTst


Tst
Sin
Sc
nSc
nTst
Din
Den
Dout
I0
I2
I3
I4
I5
I6
I19
I30
I16
I10
I45
I46
I47
I48
I80
I77
I75
I74
I38
I39
I40
I41
I54
I53
I50
I51
I27
I31
I28
I34
I35
I37
I36
I14
I15
I9
I8

Appendix D : AMULET2e memory controller

Table D.1: *State table for the AMULET2e memory controller*

x_6	x_5	x_4	x_3	x_2	x_1	x_0	current state	f_3	f_2	f_1	f_0	next state
DRAM refresh												
-	-	-	-	1	1	-	000	0	0	0	0	001
-	-	-	-	1	1	-	001	0	0	0	0	010
-	-	-	-	1	1	-	010	0	0	1	0	011
-	-	-	-	1	1	-	011	1	0	1	0	100
-	-	-	-	1	1	-	100	1	0	1	0	101
-	-	-	-	1	1	-	101	1	0	0	0	110
-	-	-	-	1	1	-	110	1	0	0	0	111
-	-	-	-	1	1	-	111	0	0	0	0	000
DRAM access												
-	-	-	-	1	0	0	000	0	0	0	0	001
-	-	-	-	1	0	1	000	1	1	0	0	101
-	-	-	-	1	0	-	001	0	0	0	0	010
-	-	-	-	1	0	-	010	0	1	0	0	011
-	-	-	-	1	0	-	011	1	1	0	1	100
-	-	-	-	1	0	-	100	1	1	0	0	101
-	-	-	-	1	0	-	101	1	1	1	0	110
-	-	-	-	1	0	-	110	1	1	1	0	000
Tidy												
-	-	-	-	1	0	-	000	0	0	0	0	000
Static cycles												
0	0	0	0	0	0	-	000	1	1	0	0	000
0	1	0	0	0	0	-	000	1	1	0	0	011
1	-	0	0	0	0	-	000	1	0	0	0	001
1	-	1	-	0	0	-	000	1	0	0	0	010
1	-	-	1	0	0	-	000	1	0	0	0	010
0	-	1	-	0	0	-	000	1	1	0	0	100

Table D.1: *State table for the AMULET2e memory controller*

x_6	x_5	x_4	x_3	x_2	x_1	x_0	current state	f_3	f_2	f_1	f_0	next state
0	-	-	1	0	0	-	000	1	1	0	0	100
-	1	-	-	0	0	-	001	1	1	0	0	011
-	0	-	-	0	0	-	001	1	1	0	0	000
-	-	-	-	0	0	-	010	1	1	0	0	100
-	-	1	-	0	0	-	100	1	1	0	0	101
-	0	0	1	0	0	-	100	1	1	0	0	000
-	1	0	1	0	0	-	100	1	1	0	0	011
-	0	-	0	0	0	-	101	1	1	0	0	000
-	1	-	0	0	0	-	101	1	1	0	0	011
-	-	-	1	0	0	-	101	1	1	0	0	110
-	0	-	-	0	0	-	110	1	1	0	0	000
-	1	-	-	0	0	-	110	1	1	0	0	011
-	-	-	-	0	0	-	011	1	0	0	0	000

Inputs:

x_6 - setup signal;

x_5 - hold signal;

x_4 and x_3 - timing/DRAM size signals;

x_2 - DRAM memory signal;

x_1 - refresh cycle signal;

x_0 - sequential memory address signal.

Outputs:

f_3 - memory strobe;

f_2 - read/write strobe;

f_1 - column address strobe;

f_0 - control signal for the address multiplexer.

Appendix E : Asynchronous Block Sorter

E.1 Tangram program of the four-stage block sorter

```
byte = type [0..255]
& word = type <<byte, byte>>
& boolword = type <<bool, word>>
& stages = const 4
|
(ip?word & a!boolword & b!boolword & c!boolword & d!boolword & e!boolword &
op!word).

begin

        head = proc(ip?word & op!boolword).
        begin
                new: var word
        |
                forever do
                        for (stages - 1) do ip?new; op!<<true,new>> od
                        ;ip?new; op!<<false,new>>           {last value mark true}
                od
        end

{tail strips off the boolean and outputs a word as required}

        & tail = proc(ip?boolword & op!word).
        begin
                new: var boolword
        |
                forever do
                        ip?new; op!new.1  {removes the boolean by outputting only the word}
                od
        end

        & sortcell = proc(ip?boolword & op!boolword).
        begin
                new, value: var boolword
```

```
    & sort : proc().
    if (new.1).0>(value.1).0 then op!value; value:=new else op!<<true,new.1>> fi

    & ipnew : proc().
        ip?new
        |
        forever do
            ip?value; ipnew();
            do (new.0) then sort() ;ipnew() od;
            sort();
            op!<<false,value.1>>
        od

    end

{main program}
|
head(ip,a) || sortcell(a,b) || sortcell(b,c) || sortcell(c,d) || sortcell(d,e) || tail(e,op)


{the end}
end
```

E.2 Handshake implementations of the basic components of the block sorter

E.2.1 Handshake implementation of the head cell

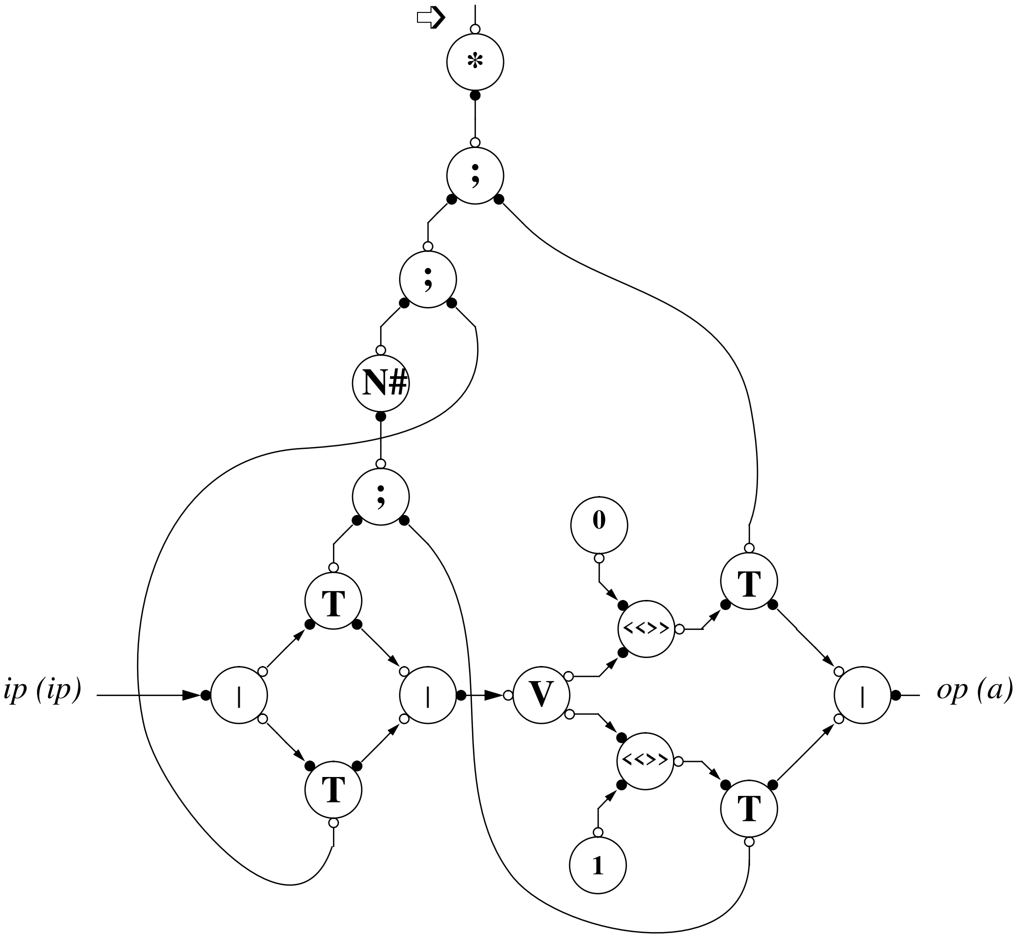


Figure E.1: *Head cell*

E.2.2 Handshake implementation of the sorting cell

(see next page)

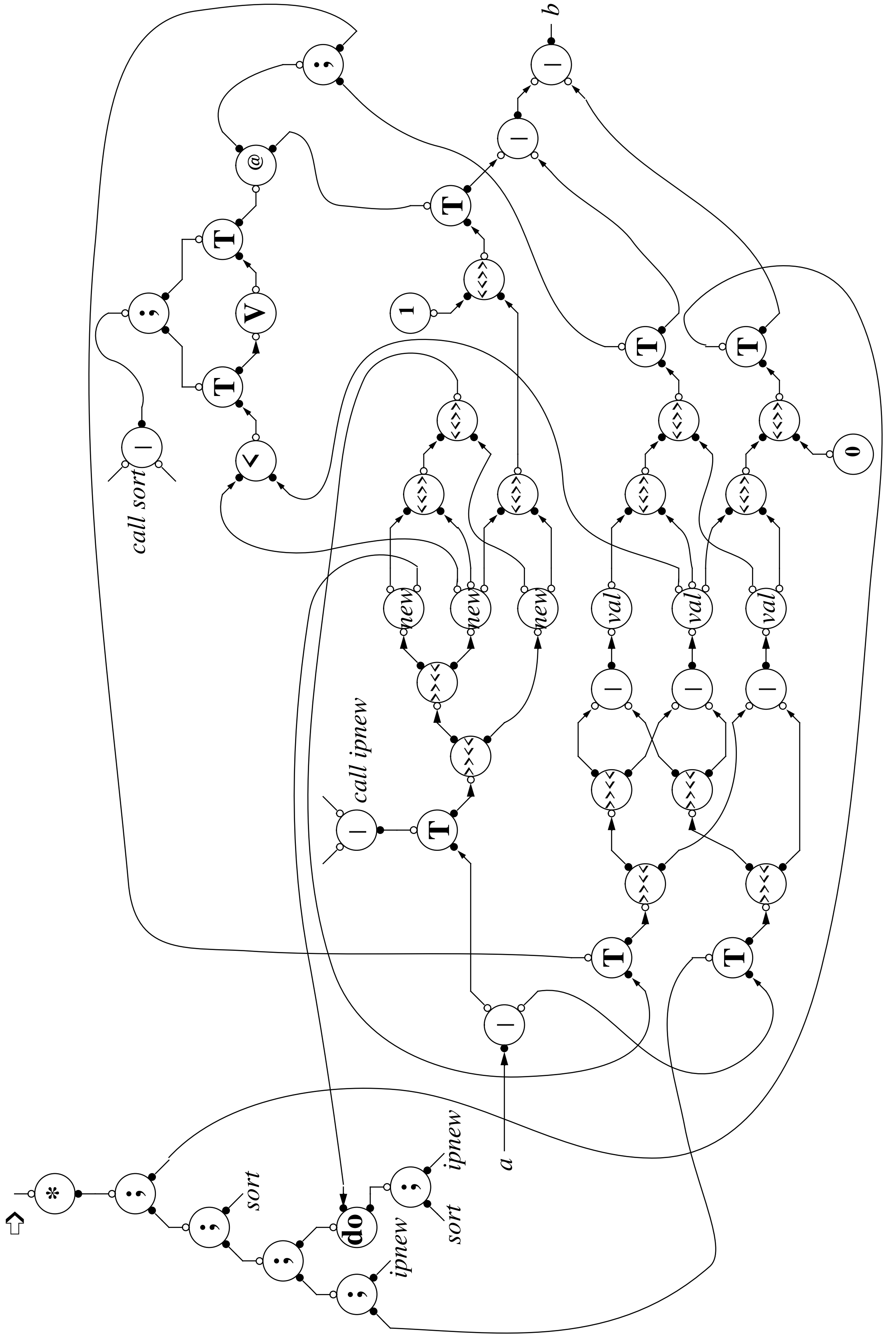


Figure E.2: *Sorting cell*

E.2.3 Handshake implementation of the tail cell

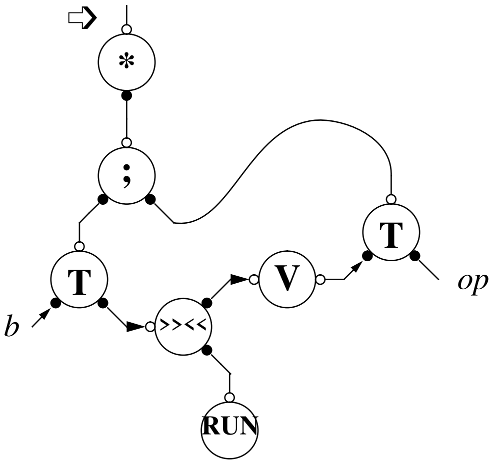


Figure E.3: *Tail cell*

References

[Abra90] M. Abramovici, M. Breuer, A. D. Friedman, “Digital Systems Testing and Testable Design”, Computer Science Press / Freeman, New York, 1990.

[Abrah86] J. A. Abraham, W. K. Fuchs, Fault and error models for VLSI”, Proceedings of the IEEE, vol.74, no.5, May, 1986, pp. 639-654.

[Agra76] P. Agrawal, V. D. Agrawal, “On Monte Carlo testing of logic tree networks”, IEEE Trans. on Computers, C-25(6), 1976, pp. 664-667.

[Agra81] V. D. Agrawal, “An information theoretic approach to digital fault testing”, IEEE Trans. Computers, Vol. C-30, No. 8, Aug., 1981, pp. 582-587.

[Agra92] P. Agrawal, V. D. Agrawal, S. C. Seth, “A new method for generating tests for delay faults in non-scan circuits”, Proc. the Fifth Int. Conf. on VLSI Design, Bangalore, India, January 1992, pp. 4-11.

[Ashki94] A. Ashkinazy, D. Edwards, C. Farnsworth, G. Gendel, S. Shikand, “Tools for validating asynchronous digital circuits”, Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (Async94), Nov. 1994, pp. 12-21.

[Beerel92] P. A. Beerel, T. H.-Y. Meng, “Semi-modularity and testability of speed-independent circuits”, Integration, The VLSI Journal, 13(3), Sept., 1992, pp. 301-322.

[Beerel93] P. A. Beerel, T. Meng, “Automatic gate level synthesis of speed-independent sircuits”, Proc. IEEE/ACM Int. Conf. on CAD, IEEE Computer Society Press, Nov. 1993, pp. 261-267.

[Ben84] R. G. Bennetts, “Design of testable logic circuits”, Addison-Wesley Publishers Limited, 1984.

[Berk88] C. H. Kees van Berkel, C. Niessen, M. Rem, R. W. J. J. Sages, “VLSI programming and silicon compilation”, Proc. ICCD'88, Rye Brook, New York, 1988, pp. 150-166.

[Berk91] C. H. Kees van Berkel, J. Kessels, M. Roncken, R. Saeijs, F. Schalij, “The VLSI programming language Tangram and its translation into handshake circuits”, Proc. European DAC, 1991, pp. 384-389.

[Berk93] C. H. Kees van Berkel, “Handshake circuits. An asynchronous architecture for VLSI programming”, Int. Series on Parallel Computation 5, Cambridge University Press, 1993.

[Berk94] C. H. Kees van Berkel, R. Burgess, J. Kessels, M. Roncken, F. Schalij, A. Peeters, “Asynchronous circuits for low power: A DCC error corrector”, IEEE Design & Test of Computers, Vol. 11, No. 2, 1994, pp. 22-32.

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

[Bost84] L. Bostock, S. Chandler, “Pure mathematics”, Stanley Thornes (Publishers) Ltd., 1984.

[Brun89] E. Brundvand, R. F. Sproull, “Translating concurrent programs into delay-insensitive circuits”, Int. Conf. on CAD, ICCAD-89, IEEE Computer Society Press, 1989, pp. 91-97.

[Brzo95a] J. A. Brzozowski, C-J. H. Seger, “Asynchronous circuits”, Springer-Verlag New York, Inc., 1995.

[Brzo95b] J. A. Brzozowski, K. Raahemifar, “Testing C-elements is not elementary”, Proc. 2nd Working Conf. on Asynchronous Design Methodologies, South Bank University, London, May 30-31, 1995, pp.150-159.

[Chan73] T. J. Chaney, C. E. Molnar, “Anomalous behavior of synchronizer and arbiter circuits”, IEEE Transactions on Computers, C-22(4), April, 1973, pp. 421-422.

[Chen84] H. H. Chen, R. G. Mathews, J. A. Newkirk, “Test generation for MOS circuits”, 1984 International Test Conference, October, 1984, pp. 70-79.

[Cheng89] K.-T. Cheng, V. D. Agrawal, “Unified methods for VLSI simulation and test generation”, Kluwer Academic Publishers, USA, 1989.

[ChinTR84] C. Chin, E. J. McCluskey, “Weighted pattern generation for built-in self-test”, Stanford University, Centre for Reliable Comput., Technical Report 84-249, Aug., 1984.

[Chu87] T.-A. Chu, “Synthesis of self-timed VLSI circuits from graph-theoretic specifications”, PhD thesis, Massachusetts Institute of Technology., 1987.

[Cort87] J. Max Cortner, “Digital test engineering”, John Wiley & Sons, Inc., USA, 1987.

[Dav90] I. David, R. Ginosar, M. Yoeli, “Self-timed is self-diagnostic”, TR-UT-84112, Department of Computer Science, University of Utah, Salt Lake City, UT, USA, 1990.

[Davi81] R. David, P. Thevenod-Fosse, “Random testing of integrated circuits”, IEEE Transactions on Instrumentation and Measurement, Vol.IM-30(1), March, 1981, pp. 20-25.

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

[Deng94] A.-C. Deng, “Power analysis for CMOS/BiCMOS circuits”, Proc. of Int. Workshop on Low Power Design, Napa, Calif., USA, Apr. 24-27, 1994, pp. 3-8.

[Dob93] D. W. Dobberpuhl and et al. “A 200-MHz 64-bit dual-issue CMOS microprocessor”, Digital Technical Journal, 4 (4), 1993, pp. 35-50.

[Eber87] J. C. Ebergen, “Translating programs into delay-insensitive circuits”, PhD thesis, Eindhoven University of Technology, 1987.

[Eber91] J. C. Ebergen, “A formal approach to designing delay-insensitive circuits”, Distributed Computing, 5(3), 1991, pp. 107-119.

[EdTR95] D. Edwards, C. Farnsworth, “Exploitation of asynchronous circuit technologies”, Tech. Report EXACT/MU/Nov/C4, Nov. 1995.

[Farn95] C. Farnsworth, D. A. Edwards, Jianwei Liu, S. S. Sikand, “A hybrid asynchronous system design environment”, Proc. 2nd Working Conf. on Asynchronous Design Methodologies, South Bank University, May 30-31, 1995, pp. 91-98.

[FarnTR96] C. Farnsworth, “Tangram optimizations”, Technical Report, Department of Computer Science, University of Mancheter, Manchester, UK, to appear in 1996.

[Feug88] R. J. Feugate, “Introduction to VLSI testing”, Prentice Hall, New Jersey, USA, 1988.

[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.

[Furb96] S. B. Furber, P. Day, “Four-phase micropipeline latch control circuits”, to be published in IEEE Transactions on VLSI Systems, June, 1996.

[Gars93] J. D. Garside, “A CMOS VLSI implementation of an asynchronous ALU”, IFIP WG 10.5 Working Conference on Asynchronous Design Methodologies, Editors S. Furber, D. Edwards, Manchester, 1993.

[Hauck95] S. Hauck, “Asynchronous design methodologies: An overview”, Proc. IEEE, Vol. 83, No. 1, Jan. 1995, pp. 69-93.

[Haz92] P. Hazewindus, “Testing delay-insensitive circuits”, Ph.D. thesis, Caltech-CS-TR-92-14, California Institute of Technology, 1992.

[Hew77] “A designer's guide to signature analysis”, Hewlett-Packard Ltd., Application Note 222, 1977.

[Huff54] D. A. Huffman, “The synthesis of sequential switching circuits”, J. Franklin Institute, 1954.

[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.

[Kess94] J. L. W. Kessels, “Calculational derivation of a counter with bounded response time and bounded power dissipation”, Distributed Computing, 8(3), 1994.

[Keut91] K. Keutzer , L. Lavagno, A. Sangiovanni-Vincentelli, “Synthesis for testability techniques for asynchronous circuits”, Int. Conf. on Computer-Aided Designs, 1991, pp. 326-329.

[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.

[Khoc95] A. Khoche, E. Brunvand, “A partial scan methodology for testing self-timed circuits”, Proc. 13th IEEE VLSI Test Symposium, Princeton, New Jersey, USA, May 1995, pp. 283-289.

[Lala85] P. K. Lala, “Fault tolerant and fault testable hardware design”, Prentice-Hall Int., Inc., London, UK, 1985.

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

[LeeTR93] H. K. Lee and D. S. Ha, “On the generation of test patterns for combinational circuits”, Technical Report No. 12_93, Dept. of Electrical Eng., Virginia Polytechnic Institute and State University, 1993.

[Li88] T. Li, “Design for VLSI asynchronous circuits for testability”, Int. J. Electronics, Vol. 64, No. 6, 1988, pp. 859-868.

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

[Mart89] A. J. Martin, “From communicating processes to delay-insensitive circuits”, Tech. Report, Department of Computer Science, California Institute of Technology, Caltech-CS-TR-89-1, 1989.

[Mart90] A. J. Martin, “Programming in VLSI: from communicating processes to delay-insensitive VLSI circuits”, In C. A. R. Hoare, editor, UT Year of Programming Institute on Concurrent Programming, Addison-Wesley, 1990.

[Mead80] C. Mead, L. Conway, “Introduction to VLSI systems”, Addison-Wesley Publishing Company, 1980.

[Meng89] T. H.-Y. Meng, R. W. Brodersen, D. G. Messershmitt, “Automatic synthesis of asynchronous circuits from high-level specifications”, IEEE Transactions on CAD, 8(11), Nov., 1989, pp. 1185-1205.

[Mol85] C. E. Molnar, T.-P. Fang, F. U. Rosenberger, “Synthesis of delay-insensitive modules”, In Henry Fuchs, editor, 1985 Chapel Hill Conf. on VLSI, Computer Science Press, Inc., 1985, pp. 67-86.

[Need91] W. M. Needham, “Designer's guide to testable ASIC devices”, Van Nostrand Reinhold, New York, USA, 1991.

[Nowick91] S. M. Nowick, D. L. Dill, “Automatic synthesis of locally-clocked asynchronous state machines”, In Proc. Int. Conf. on CAD, Nov. 1991.

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

[Panc92] A. Pancholy, J. Rajski, L. J. McNaughton, “Emperical failure analisys and validation of fault models in CMOS VLSI circuits”, IEEE Trans. Design and Test of Computers, vol. 9, no. 2, March, 1992, pp. 72-83.

[Park89] E. S. Park, M. R. Mercer, T. W. Williams, “A statistical model for delay-fault testing”, IEEE Design and Test of Computers, vol. 6, no. 1, February, 1989, pp. 45-55.

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

[PeTR95] O. A. Petlin, S. B. Furber, “Designing C-elements for testability”, Technical Report UMCS-95-10-2, Department of Computer Science, University of Mancheter, Manchester, UK, 1995.

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

[Pet95a] O. A. Petlin, S. B. Furber, A. M. Romankevich, V. V. Groll, “Designing asynchronous sequential circuits for random pattern testability”, IEE Proc.- Comput. Digit. Tech., Vol. 142, No. 4, July 1995.

[Pet95b] 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.

[Pet95c] 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.

[Pet95d] O. A. Petlin, S. B. Furber, “Power consumption and testability of CMOS VLSI circuits”, submitted to IEEE Transactions on CAD.

[Pet95e] O. A. Petlin, C. Farnsworth, S. B. Furber, “Design for testability of an asynchronous adder”, Proc. of IEE Colloquium on Design and Test of Asynchronous Systems, London, UK, 28 Feb., 1996, pp. 5/1- 5/9.

[Putz71] G. R. Putzolu, J. P. Roth, “A heuristic algorithm for the testing of asynchronous circuits”, IEEE Transactions on Computers, C-20(6), June 1971, pp. 639-647.

[Red86] M. K. Reddy, S. M. Reddy, “Detecting FET stuck-open faults in CMOS latches and flip-flops”, IEEE Design & Test of Computers, vol. 3, no. 5, Oct. 1986, pp. 17-26.

[Rice82] R. Rice, “Tutorial: VLSI Support Technologies-Computer-Aided Design, Testing and Packaging”, IEEE Computer Society, Los Alamitos, Calif., 1982, pp. 228-308.

[Rom89] A. M. Romankevich, V. V. Groll, “On random pattern testability of digital circuits”, Proc. 12th Int. Conf. on Diagnostic Support of Digital Systems, Praha, Czech Republic, Sept. 1989, pp. 217-218.

[Ron93] M. Roncken, R. Saeijs, “Linear test times for delay-insensitive circuits: a compilation strategy”, IFIP WG 10.5 Working Conference on Asynchronous Design Methodologies, Editors S. Furber, D. Edwards, Manchester, 1993, pp. 13-27.

[Ron94] M. Roncken, “Partial scan test for asynchronous circuits illustrated on a DCC error corrector”, Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (Async94), Nov. 1994, pp. 247-256.

[Rosie66] A. M. Rosie, “Information and communication theory”, Blackie & Son Ltd., 1966.

[Russ85] G. Russell, D. J. Kinniment, E. G. Chester, M. R. McLauchlan, “CAD for VLSI”, Van Nostrand Reinhold (International), 1985.

[Russ87] G. Russell, “Computer aided tools for VLSI system design”, Peter Peregrinus Ltd., 1987.

[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.

[Scha93] F. Schalij, “Tangram manual”, Technical report UR 008/93, Philips Research Laboratories Eindhoven, P.O. Box 80.000, 5600 JA Eindhoven, The Netherlands, 1993.

[Shan64] C. E. Shannon, W. Weaver, “The mathematical theory of communication”, The University of Illinois Press, Urbana, 1964.

[ShenG92] A. Shen, A. Ghosh, S. Devadas, K. Keutzer, “On average power dissipation and random pattern testability of CMOS combinational logic networks”, ICCAD-92: IEEE/ACM Int. Conf. on CAD, Santa Clara, CA, Nov., 1992, pp.402-407.

[ShenM85] J. Shen, W. Maly, F. Ferguson, “Inductive fault analysis of MOS integrated circuits”, IEEE Trans. Design and Test of Computers, vol. 2, no. 6, Dec., 1985, pp. 13-26.

[Sim94] “SIMIC: design verification tool”, User's Guide, Genashor Corporation, N.J., 1994.

[Souf95] M. Soufi, Y. Savaria, B. Kaminska, “On the design of at-speed testable VLSI circuits”, Proc. 13th IEEE VLSI Test Symposium, Princeton, New Jersey, USA, May 1995, pp. 290-295.

[SproTR94] R. F. Sproull, I. E. Sutherland, C. E. Molnar, “Counterflow pipeline processor architecture”, Technical Report SMLI TR-94-25, Sun Microsystems Labs, Inc., April, 1994.

[Suss84] A. K. Susskind, “A technique for making asynchronous sequential circuits readily testable”, 1984 Int. Test Conf., 1984, pp. 842-846.

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

[Turin90] J. L. Turino, “Design to test: a definitive guide for electronic design, manufacturing, and service”, Van Nostrand Reinhold, New York, USA, 1990.

[Unger69] S. H. Unger, “Asynchronous sequential switching circuits”, Wiley-Interscience, New York, NY, 1969.

[Veen84] H. J. M. Veendrick, “Short-circuit dissipation of static CMOS circuit and its impact on the design of buffer circuits”, IEEE J. Solid-State Circuits, Vol. 19, Aug., 1984, pp. 468-473.

[Wad78] R. L. Wadsack, “Fault modelling and logic simulation of CMOS and MOS circuits”, Bell System Tech. Journal, vol. 57, May-June 1978, pp. 1449-1474.

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

[Waic89] J. A. Waicukauski, E. Lindbloom, F. B. Eichelberger, “A method for generating weighted random test patterns”, IBM J. Res. and Dev., 1989, no. 2, pp. 149-161.

[Weste93] N. H. E. Weste, K. Eshraghian, “Principles of CMOS VLSI design: A systems perspective”, Addison-Wesley Publishing Co., 1993.

[Wey93] Chin-Long Wey, Ming-Der Shieh, D. Fisher, “ASCLScan: a scan design for asynchronous sequential logic circuits”, Proc. IEEE Int. Conf. on Computer-Aided Design, 1993, pp. 159-162.

[Will73] T. W. Williams, J. B. Angell, “Enhancing testability of large scale integrated circuits via test points and additional logic”, IEEE Trans. Computers, Vol. C-22, No. 1, Jan., 1973, pp. 46-60.

[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.