

## 13

## Field-Programmable Gate Arrays and Microprocessors

## 13.1 INTRODUCTION

This chapter is devoted to two widely used types of specialized circuits: field programmable gate arrays (FPGAs) and microprocessors. Two of the most attractive attributes of these circuits are their availability off the shelf as IC and embedded cores and their flexibility to customize to many applications. In addition, they come in different technologies and a wide range of architectures. For example, some FPGAs are reprogrammable and may serve in particular applications that need different reconfigurations on the fly, such as emulators, and one-time programmable FPGAs satisfy dedicated applications. Similarly, microprocessors may have general-purpose RISC or dedicated CISC architectures in order to serve a variety of data processing, from a simple controller to a sophisticated image-processing engine. FPGAs and microprocessors nowadays are often used as embedded cores in a system on chips. In the beginning, FPGAs have been used as prototyping devices instead of wire wrapping small and medium-sized ICs (SSI and MSI). In certain applications, designs are first implemented in FPGAs. Then, as they mature, they are migrated to mask-programmable gate arrays (MPGAs).

The architecture of FPGAs is quite regular. The testing of FPGAs depends on their type of programming. For RAM-based FPGAs, it is possible to reprogram them in different configurations and test the blocks as well as the interconnects. One-time programmable FPGAs are not as easily tested for any possible application. In this chapter we concentrate on reprogrammable FPGAs and, in particular, RAM-based FPGAs. Testing these devices requires configuring them in specific designs and then applying test patterns. Selecting the appropriate test configurations and the test sequence has to be optimized for minimal testing time and low volume of testing data.

The microprocessor, on the other hand, has a more complex and less regular architecture. It consists of only a few components that are quite diverse: registers and functional units (the datapath), and an FSM (the controller). In addition, they have complex associated RAMs and data architectures. Present technology processors are characterized by high integration and high speed. They are produced in high volume and are cost-sensitive ICs. Testing microprocessors does not require them to be configured. Several models have been formulated to facilitate

their testing. Descriptions of FPGAs and their testing are covered in Sections 13.2 to 13.4; microprocessors and their testing are the topics of Sections 13.5 to 13.7.

## 13.2 Field-Programmable Gate Arrays

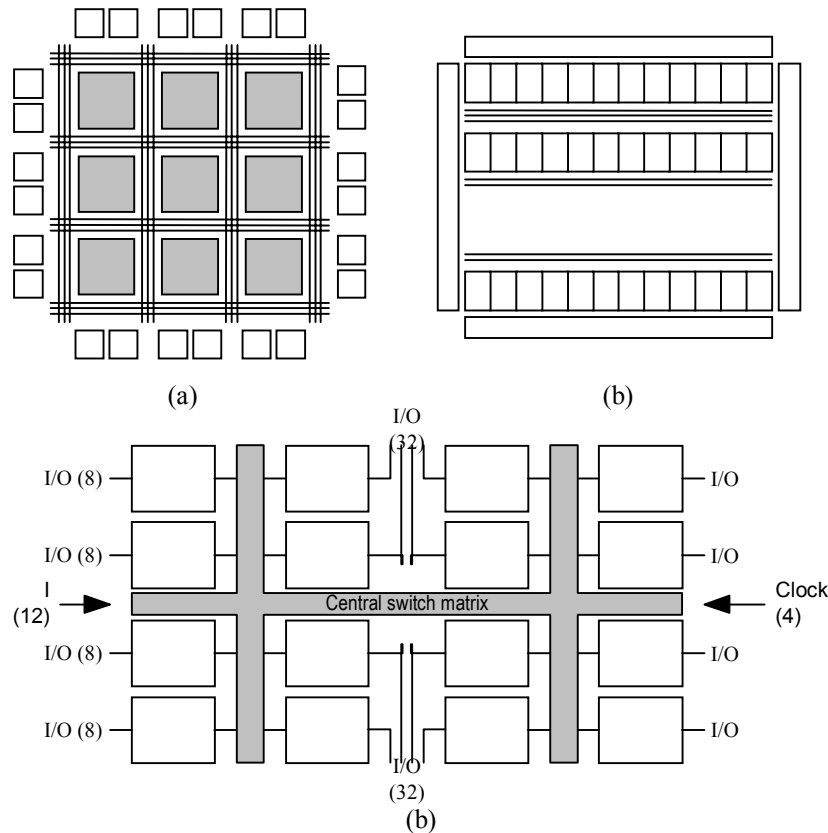


Figure 13.1 FPGAs Architectures a) Matrix , b) Gatearray, c) AND/OR Blocks

An FPGA consists of several uncommitted logic blocks in which the design is to be encoded. Each logic block consists of several universal gates. A large portion of die area is for programmable routing. The connectivity between any two blocks is programmed via different types of devices such as SRAM, EEPROM, or antifuse [Trimberger 1994]. The architecture of the FPGAs depends on the fashion by which the blocks are connected. They may be in one of the forms shown in Fig. 13.1 and listed below:

- Islands in a matrix with horizontal and vertical channels (Lucent, Quicklogic, and Xilinx devices)
- Rows separated by routing channels such as these in MPGAs (Actel devices)
- Blocks of AND/OR logic arrays (Altera devices)

The advantages of FPGAs are (1) rapid turnaround, (2) availability of parts off the shelf, (3) larger gate counts and more design flexibility than in SSI and MSI chips, (4) low risk, and (5) reprogrammability (for some FPGAs). Their limitations include (1) circuit delay depends on the performance of tools used in the design implementation, (2) the delay parameters can be extracted only after placement and routing, typically a time consuming process, (3) mapping the design on FPGA architecture requires sophisticated tools, and (4) FPGAs are less dense and slower than traditional gate arrays.

### 13.2.1 Architecture

The principal components of an FPGA are (1) an array of uncommitted logic blocks (LBs) in which the design is to be encoded, (2) an interconnect network, and (3) programmable elements. A logic block comprises a number of universal blocks or a group of gates; these are gates that can be used to implement any combinational function. Examples of these gates are NAND or NOR gates, combinations of AND/OR/NOT gates or of XOR/AND gates, multiplexers, and RAMs.

The logic blocks may be organized in the form of a matrix, a gate array, or a configuration of AND/OR blocks. The three types are illustrated in Fig. 13.1. Examples of matrix architectures are XACT 3000 and 4000 by [Xilinx 1993] and Pasic by [Quicklogic 1993]. In the first example, the LB is a small RAM, also referred to as a lookup table (LUT). The second example is a mix of multiplexers and primitive gates. The two examples are illustrated in Fig. 13.2*a* and *b*, respectively. In the matrix architecture, the blocks are interconnected along horizontal and vertical channels. ACT1 and ACT2 are examples of the gate array type [Actel 1991]. Each block consists of a set of multiplexers arranged as shown in Fig. 13.2*c*. Interconnections between the block are in the channels separating the rows of LBs. The EPM5128 is a typical AND/OR block architecture [Altera 1991].

Figure 13.3 shows general interconnect structure and resources in an FPGA. The routing in the channels is along segmented wires that may be joined through the programmable interconnecting points (PIPs). The programming elements are described in the next section. The change of routing from a horizontal to a vertical channel, or vice versa, is through switch boxes.

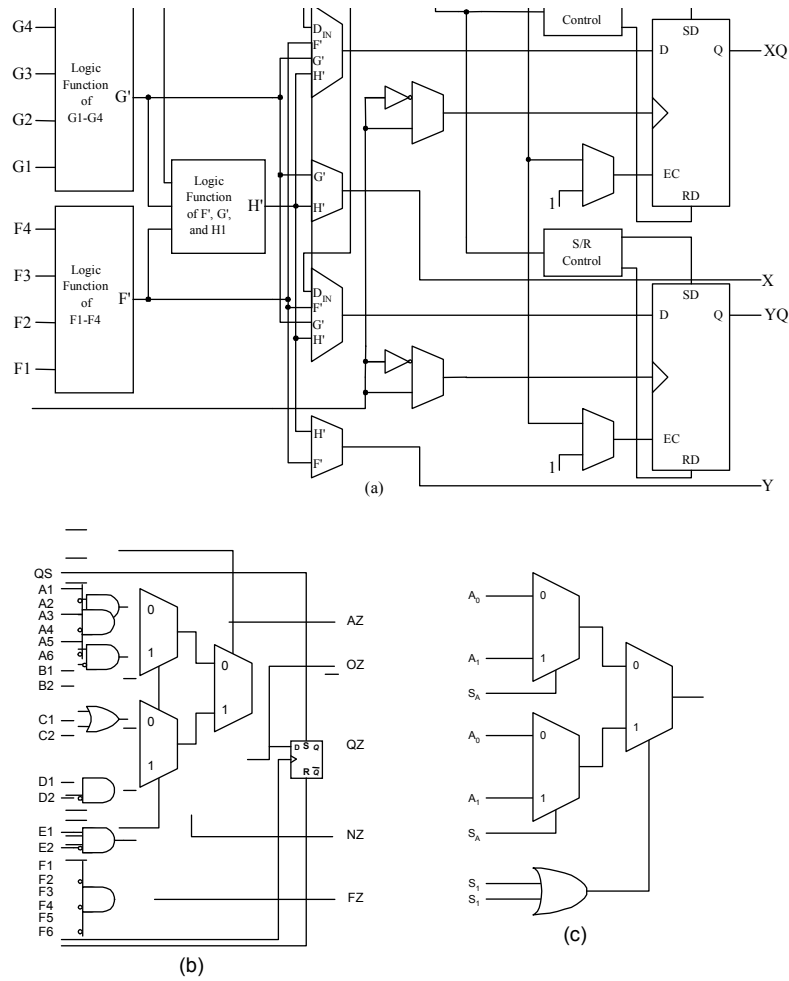


Figure 13.2 Logic Blocks a) Xilinx 4000, b)Quicklogic, c) Actel

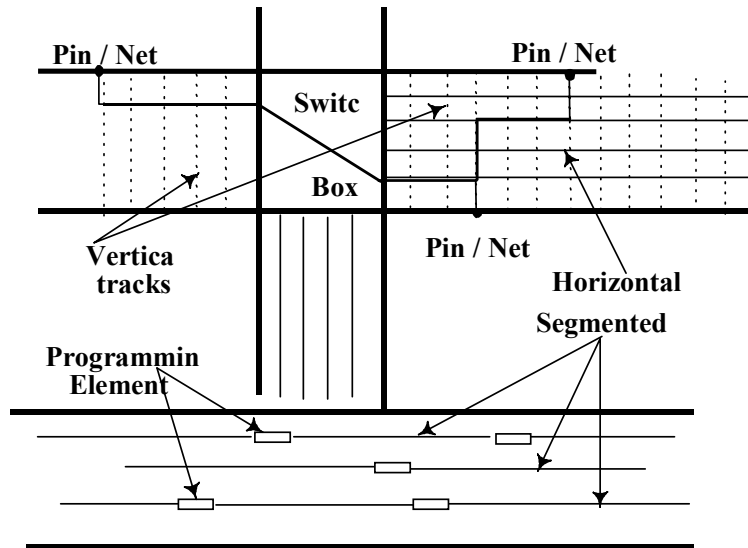


Figure 13.3 Interconnect Resources

### 13.2.2 Programmability

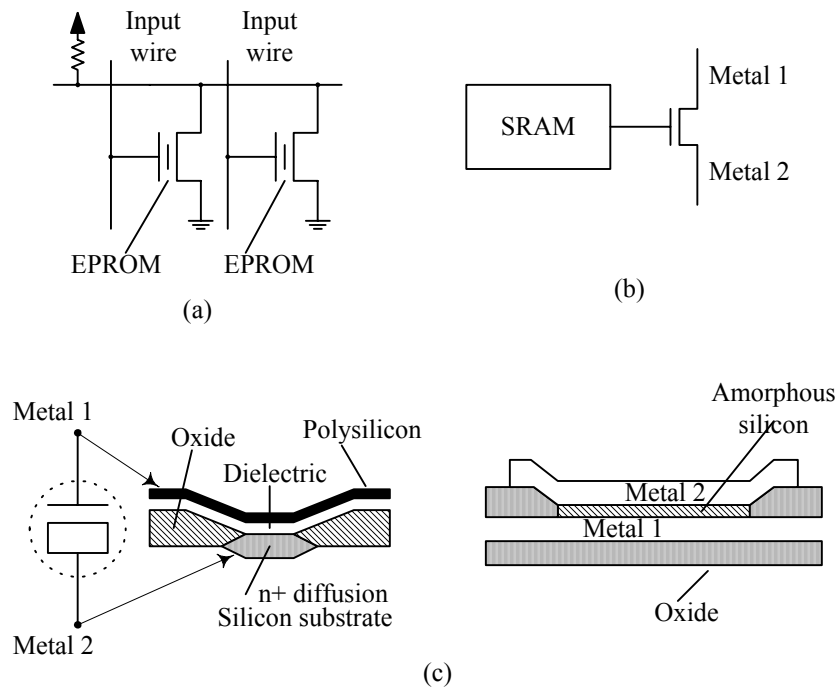


Figure 13.4 Programming Elements, a) EEPROM b) SRAM, c) Antifuse: Plice and ViaLink

The programming device is an important part of the FPGAs. The main types used are EEPROM, SRAM, and antifuse. All devices are illustrated in Fig. 13.4. Lookup table – based FPGAs use programmable devices in both the logic block and the interconnect structure. For other types of LBs, the programmable device is used only for the interconnections. Finally, for AND/OR blocks, the programmable device which is mostly an EEPROM type is used in a fashion similar to its use in PALs and PLDs, as shown in Fig. 13.4a.

The metal segments in SRAM FPGAs are connected via pass transistors that is on or off depending on the state of the SRAM cell attached to its gate. This is illustrated in Fig. 13.4b. The antifuse is used only for programming the interconnect structure. An antifuse is a nonvolatile, two-terminal element. It occupies the area of a via and connects two metal segments when programmed. Two types of antifuse are shown in Fig. 13.4c. The Plice, Actel's antifuse, consists of a dielectric layer between two conducting materials, polysilicon and diffusion. When the appropriate voltage is applied across the dielectric, the two metal segments are connected through a very low resistance, about 500  $\Omega$ . The off resistance of the antifuse is higher than 100 k $\Omega$  [Hamdy 1988]. Quicklogic's ViaLink is a metal-to-metal antifuse separated by a thin layer of amorphous silicon. It has a smaller resistance than the Plice, on the order of 50  $\Omega$ .

Table 13.1 Comparing Programming Elements

FEATURE	SRAM	EPROM	ANTIFUSE
Programming method	Shift register	UV erasure	Break down
Area	Very large	Large	Small
Resistance	$\approx 2K\Omega$	$\approx 1.8K\Omega$	$\approx 50$ or $500\Omega$
Capacitance	$\approx 50$ fF	$\approx 15$ fF	$\approx 5$ fF

The size of the RAM cell is large compared to the antifuse. In addition to size, the other two most important characteristics of programmable devices are the volatility and the electrical characteristics: resistance and capacitance. Table 13.1 compares among the three important elements.

### 13.3 Testability of FPGAs

Similar to any other IC, FPGAs are tested upon manufacturing. However, testing FPGAs is completely different from that of other ICs or embedded cores. Testing traditional VLSI requires test pattern generation that is applicable only to a specific IC. In contrast, testing FPGAs of a certain type is the same regardless of their use. Therefore, it is reasonable to invest in testing FPGAs since, in addition to increasing yield, testing cost is quickly amortized over the large production volume.

#### 13.3.1 Defects and Faults

In addition to failures common to conventional ICs, FPGAs are subject to unique failure modes that are identified as programmability failures [Chan 1994]. These types of failures may be due to a defective chip or malfunction of the device programmer. A noncalibrated programmer may apply too little or too much voltage for incorrect periods of time and cause an antifuse to remain intact or connected with higher resistance. This might cause missing and extra connections between the segments of the routing wires. In addition, they may introduce longer delays in the interconnect wires.

Each FPGA is also prone to specific physical defects, depending on the underlying architecture and programming technology. For instance, in EPROM-based technology, erasure of programmed connections can occur with exposure to light. The logic cell array using volatile static memory makes it prone to problems caused by noise, radiation, and power dropouts. Independent of the technology, failure modes in the interconnects between metal wire segments may result in extra or missing connections. Depending on the position of the extra or missing

connection, the failure results in different possible fault models. Examples of mapping these types of failures into known fault models are listed in Table 4.2. Other causes for interconnect failures may result from incorrect or corrupted programming data files or the use of the wrong device type. These types of failures can result in logic block functional failures as well as interconnect failures.

### 13.3.2 Approaches to Testing FPGAs

The approach to testing FPGAs varies with programmability type. One-time programmable devices usually include testing circuitry that is used for manufacturing testing. The vendor provides both hardware and software testing resources for the user to verify post-programming functionality [Actel 1991]. In the case of reprogrammable FPGAs, it is possible to reconfigure the device such that all programmable points are tested. However, reconfiguration is time consuming and contributes to the complexity of testing. Furthermore, these FPGAs have an extremely large number of PIPs and relatively limited I/Os. This makes testing of these devices complex and challenging. In the next section we concentrate on testing RAM-based, reprogrammable FPGAs. We refer to them simply as FPGAs without identifying them as RAM-based.

## 13.4 Testing RAM-based FPGAs

Although RAM-based FPGAs came on the market in 1989 [Xilinx 1989], it is only recently that interest in their testing has peaked. Testing FPGAs consists of testing (1) the LUT, the RAM of the LB; (2) the associated logic, mostly multiplexers, and flip-flops; and (3) interconnect structure and resources. In the remainder of this section we will take a functional approach to testing the above-mentioned components and the requirements for testing them. In addition, we describe  $I_{DDQ}$  testing, BIST application, and diagnosis testing. As we will see, the regular structure of the FPGA array will be exploited to expedite testing. The array is configured into a one- and two-dimensional ILAs that we discussed in Chapter 8.

### 13.4.1 Functional Testing

Testing an FPGA requires programming in several configurations, called *testing configurations* (TCs). Changing configurations implies reprogramming costs. This imposes a requirement to determine the minimum number of test configurations that will cover all the faults of the structural fault model. Associated with each TC is a sequence of *test patterns* (TS). The objective of the TS is the detection of a set of faults. The *test procedure* TP, is one or a

succession of (TC, TS):  $TP = \{(TC,TS)1, (TC,TS)2, \dots\}$  [Inoue 1997]. A fault that is redundant in one configuration may be detected by another. If a fault is redundant for all TCs, it is undetectable. Thus the fault coverage depends on both the configurations and the test sequences applied [Renovell 1997b].

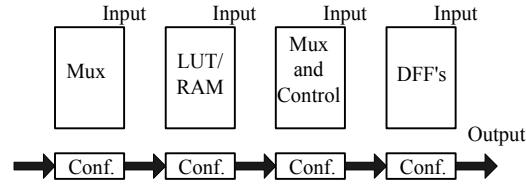


Figure 13-5. Testing the Logic Block

We consider here the three components of the LB: the RAM, the multiplexers, and the flip-flops. Figure 13.5 is a model for the LB that was detailed in Fig. 13.2a. The model distinguishes between the MUXs, the LUT/RAM, the control, and the flip-flops [Huang 1997]. The configuration registers, designated as “Conf.” in the figure, are connected in a long stream of serially entered bits.

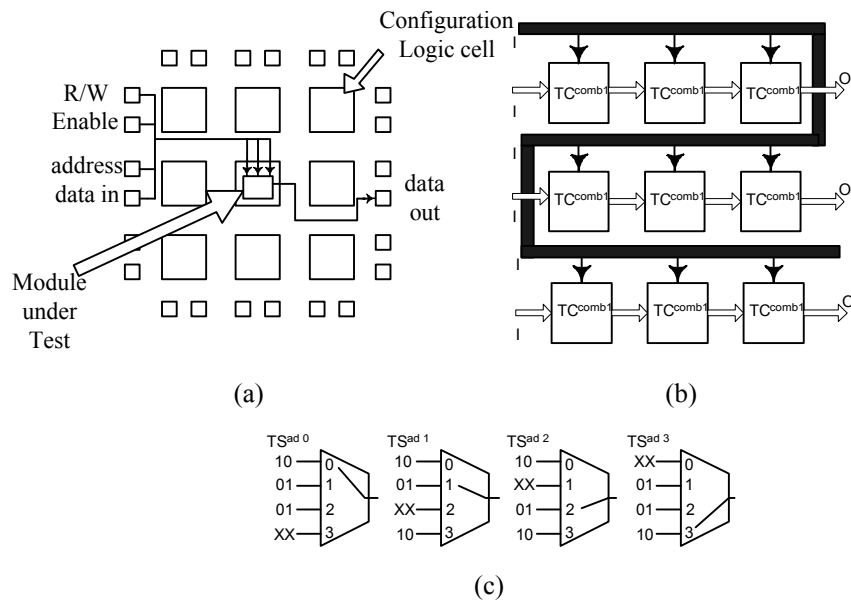


Figure 13.6 Testing the LB a) Accessing the LUT, b) Test Configuration, c) Mux Configurations

**13.4.1.1 Testing the LUT/RAM.**

We describe the approach followed by [Renovell 1999]. Accessing the RAM of an LB requires a number of input pins and an output pin as illustrated in Fig. 13.6a. The test applied to this RAM may be any of the algorithms used for memory testing, and a wealth of these given in Chapter 12. To test the RAM of an LB, only one TC is needed and the TS is short, as we show next. The XACT 4000 LB has two 16-bit RAMs.

The marching I/O test consists of  $14N$ , where  $N$  is the number of bits in the RAM. The test length is thus 224 patterns. This test will detect SAF, TF, and addressing faults (see Chapter 12). Testing one module at a time requires a long testing period. Also, the limited number of I/Os makes it impossible to test them concurrently. It is possible, however, to organize the RAMs of all modules in a single one-dimensional array as illustrated in Fig. 13.6b. The output of one module is connected to the input of the following one. This organization overcomes the I/O limitation, but it does make the controllability and observability of the modules more difficult unless a TC is used for each LB [Huang 1997]. It is also possible to have the output of one module connected to its flip-flop [Renovell 1999]. The read/write of the RAM and the clock of the flip-flops are coordinated such that shifting a value from the primary input to the primary output is through the flip-flops and the modules. This approach is referred to as a *pseudo shift register*.

**13.4.1.2 Testing the Logic Block.** To test the multiplexers functionally, each data line is selected and verified to pass a 0 or a 1. Selecting one data line is a TC. For a 4-to-1 MUX, four configurations are needed, and the test sequence consists of two vectors, as illustrated in Fig. 13.6c. Since the logic block consists of the RAMs and several multiplexers, similar test configurations of the set of MUXs can be arranged while applying the appropriate test sequences. For the XACT 4000 series, it has been found that eight TCs are sufficient to test the LB fully [Renovell 1999].

As in the case of the RAM above, the concurrency in testing the LBs is restricted by the number of I/Os. It is appropriate therefore, to join all the LBs in one array, as illustrated in Fig. 13.6b. All the blocks are first configured with the same TC, and then the test sequence is applied. To expedite the test application, each row of  $N$  blocks is receiving the TS and the observation is through the last block in the row. In such an arrangement, the TS to each block is received from the block to its left and the output is observed through the block to its right. For this testing, we then have to guarantee that the test sequence is actually passed to each embedded LB, and the response to the TS is eventually observed at the primary outputs. The test configurations include also testing the flip-flops that are connected as one shift register. For an LB having two flip-flops, FF1 and FF2, such as the XACT devices, all FF1's form one shift register and all FF2's form another shift register.

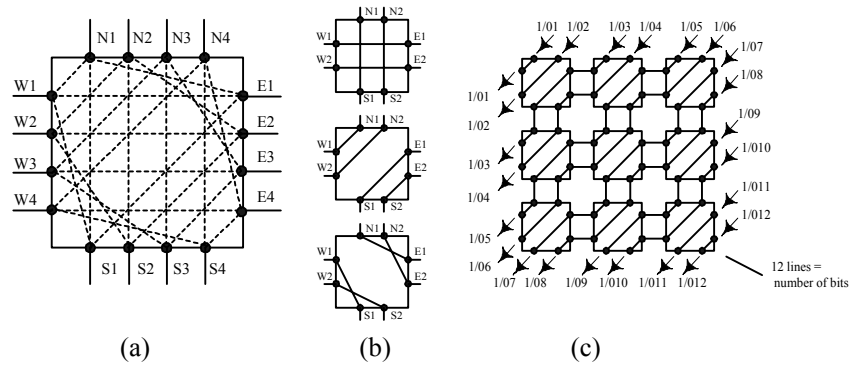


Figure 13.7 Testing the Interconnects [Renovell 1998]

**13.4.1.3 Testing the Interconnect Structure.** The interconnect structure consists of vertical and horizontal tracks and switchboxes that allow connectivities between the two types of channels. The one used in XACT 3000, shown in Fig. 13.7a, allows 20 different configurations. A switch matrix is a programmable connecting element receiving  $n$  lines on each side. Some pins in the matrix cannot be connected. These are NC pins. Pins that can be connected are C pins. Connection of C pins is governed by the programmed configuration. The failure modes of the interconnecting lines are short and open. The short may be with ground or  $V_{dd}$ . Therefore, stuck-at faults are also included. A general fault model is derived. Permanent connection (PC) is a short on any pair of NC pins or a bridge of any pair of C pins. Permanent disconnection (PD) is an open on any pair of C pins. To test these faults, a lower bound of three configurations was found sufficient for 100% test configuration coverage [Renovell 1997a]. The three TCs are shown in Fig. 13.7b for a simplified model of the switchbox. They are independent of the array size. Application of the first configuration to the entire array of switchboxes is shown in Fig. 13.7c. The number of vectors required to fully test any one of these three configurations is  $\log_2(2km + 2)$ , assuming that the switchbox has  $k$  pins and that there are  $m$  boxes along the diagonal. This formula is based on previous results for an  $n$ -bit bus [Kautz 1974, Feng 1995]. The TS has to be repeated for each TC [Renovell 1998b]. Routing resource testing was also investigated by [Huang 1996b].

### 13.4.2 $I_{DDQ}$ Testing

In Chapter 7 it was established that  $I_{DDQ}$  testing is an important supplement to voltage testing. In addition to detecting traditional fault models, it has the advantage of revealing defects that are not represented by these faults. For FPGAs, besides defects that occur in traditional ICs, there are programmability failures that result in open and bridging faults.  $I_{DDQ}$  testing can help uncover these faults. Since observability is guaranteed in  $I_{DDQ}$  testing, focus is

on controllability. Thus, configuring the FPGAs for testing can be less demanding. On the other hand, since current measurement is usually a slow process, it is important to strive for as few TCs as possible. This issue is particularly important because it usually takes more time to configure an FPGA than to measure the current.

$I_{DDQ}$  can be used to test bridging faults (BFs) in all components of an FPGA, the logic block, the I/O blocks, and the routing resources. A hierarchical approach for  $I_{DDQ}$  testing was developed for FPGAs [Zhao 1998]. This is similar to that proposed by the leakage fault model test pattern generation described in Chapter 7 [Mao 1990]. A test library was built for each type of module in the LB, at the circuit or functional level. The test library consisted of different sets of TCs and TSs to test all internal BFs. External BFs are tested using the same test libraries with some additional vectors. The two-dimensional structure of the FPGA helps in applying the TCs and TSs to all the LBs simultaneously. All LBs in a row (or a column) receive the stimulus directly from the primary inputs and, of course, there is no need to propagate the results to a primary output.

### 13.4.3 BIST

Any FPGA includes in its resources a number of flip-flops that have been used effectively in testing the circuit, as we mentioned in Section 13.4.1. These flip-flops may also be used for BIST application at no extra hardware cost. Some of the flip-flops can be configured for test pattern generation (TPG) and others for output response verification (ORV). This approach is similar to that used in BILBO and circular BIST, as described in Chapter 11. In the case of FPGAs, in addition to configuring the TPG and ORV, the array itself has to be configured before executing the test. If the FPGA includes IEEE Standard 1149.1, a RUNBIST instruction can be used for testing (see Chapter 10 on Boundary-scan). BIST can be used to test the LBs, the I/O blocks, and the routing resources. The LB may be tested as a RAM using the RAMBIST approach, or it may be configured as a logic function.

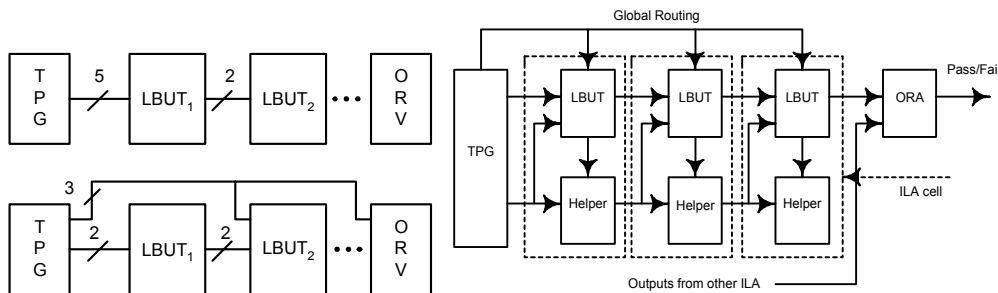


Figure 13.8 ILA Testing for BIST

Figure 13.9 ILA Testing for BIST Test Configuration [Stroud 1997]

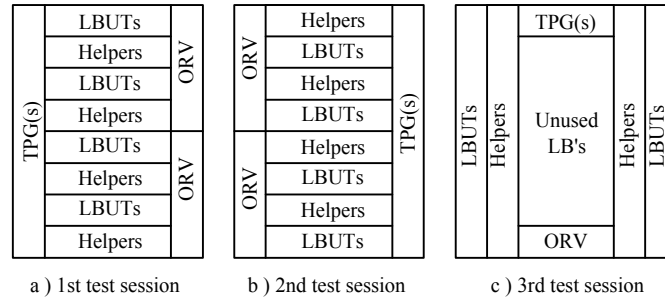


Figure 13.10 ILA Testing for BIST, Configuration for C-testability

There are different ways of configuring the FPGA for BIST. At one extreme it is possible to consider one LB at a time, configure it, and apply the test. The testing complexity is  $O(N^2)$  for an  $N \times N$  array. However, it is possible to take advantage of the regularity of the array and organize it in a one- or two-dimensional iterative array, as explained in Section 13.4.1. However, this cannot be done without special arrangement of the LBs because there are fewer outputs than inputs in each LB. Propagating the test from one LB to the next in the ILA cannot easily be accomplished. To overcome this limitation, it is possible to supply the missing signals from TPG directly to each reconfigured LB as shown in Fig. 13.8. However, this will cause routing congestion. Instead, it is possible to pair the LBs in such a way that one is placed under test (LBUT) and the other is a helper to provide the missing signals to the next LBUT in the array [Stroud 1996]. Figure 13.9 shows this arrangement. If the output of the LBUT and its helper are still fewer than the inputs of the LB, the missing signals are provided from the TPG directly to make the array C-testable. The test can be applied simultaneously to all paired arrays. The testing time is then  $O(N)$ . Once the test is completed, the LBUTs and the helpers exchange roles and the test is applied again. Finally, the LBs used for TPG and ORV are tested in the same fashion. The organization of the FPGA for the method described is illustrated in Fig. 13.10 [Stroud 1996]. The FPGA is, therefore, tested with three configurations. As any C-testable array, the technique is applicable to any size array. Each LB is tested exhaustively and the results verified by comparing the results of two consecutive rows. BIST was used effectively to test the interconnect resources [Stroud 1998] and for diagnosis [Stroud 1997].

#### 13.4.4 Diagnosis Testing

FPGA fault diagnosis is as important as testing. It applies to either programmed or unprogrammed FPGAs. Fault diagnosis of an unprogrammed FPGA leads to identification and isolation of a faulty part prior to programming, enabling designers to implement logic function using only fault-free parts. A universal fault diagnosis approach was

introduced by [Inoue 1998]. The fault model includes stuck-at, incorrect-access, non-access, and multiple-access faults of lookup tables in LBs, functional faults of multiplexers, and the D flip-flops in LBs [Inoue 1997]. The assumption is that several of these faults may occur simultaneously in a CLB, and the number of LBs including these faults in a FPGA is at most one [Inoue 1998]. The complexity of this diagnosis test is  $O(N^2 n \log n)$ .

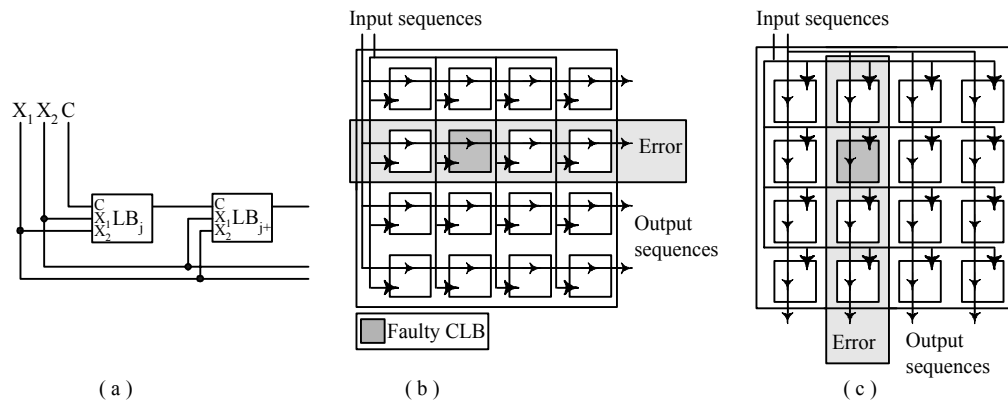


Figure 13.11 Diagnosis Testing: a) Arrangement of the LBs b) ILA Configuration in One Direction, c) ILA in the Cross Direction [Inoue 1998]

The approach of this testing is based on the concept of scalability testing discussed in Section 8.6.1. Consider a  $LB_j$  that is configured with TC. The response  $R_j$  to the application of test TS to this LB is used as part of the TS for the following  $LB_{j+1}$  under the same TC. The concept is illustrated in Fig. 13.11a. Based on these results, we can then test a sequence of LBs with a small number of I/Os. The problem, however, is that it is not possible to identify the erroneous LB. To identify one LB and expedite the test application, the TS is applied simultaneously to all rows (or columns) of LBs. In this arrangement of the LBs, which is shown in Fig. 13.11b, it is possible to identify the faulty block as the coincidence of a row and a column. A C-diagnosable approach, which is independent of the size of the array dimension  $N$ , was also devised and its complexity is  $O(n \log n)$ , where  $n$  is the size of the LUT. FPGAs diagnosis was also investigated by [Feng 1995] and [Liu 1995].

## 13.5 Microprocessors

It is superfluous to emphasize the indispensable role microprocessors have in present society. Their reliability is crucial to proper functioning of almost every aspect of this society. Since the introduction of the Intel 4004, microprocessors have proliferated in size, performance, and complexity. In addition, in some applications, instead of just customizing a generic microprocessor to the applications, special-purpose processors have been developed, such as digital signal processing (DSP). Although one can consider them as another electronic product and as such

can test them with techniques already developed, microprocessors have a whole host of unique problems. For example, microprocessor systems differ from other digital circuits in that they include a software program that can be a source of errors in both the actual program and the memory where the program is stored. Thus, for microprocessors, one needs to pay attention not only to the associated logic, glue logic, but also to the problems associated with the execution of instructions. In this case, as in the case of memory testing, functional faults are used rather than structural faults. That is, the functional faults will cover the effects of structural faults. For example, a March test detects stuck-on or stuck-open faults in the RAM cells and the decoding circuits conditions.

Testing modern microprocessors presents a real challenge for the following reasons [Needham 1998]:

- They have diverse and complex architectures.
- They have embedded memories, other storage devices, and FPGAs.
- They use deep-submicron devices, which bring an entirely new set of new problems.
- They run at very high speed, which requires accurate equipment.

In the remaining sections of this chapter we first examine the models for the generic structure of a microprocessor and then how to use the model for validating the design. Subsequently, using case studies, we review how DFT approaches are used to test actual microprocessors.

### 13.5.1 Microprocessor Models

Circuit models are useful in the development of test algorithms as well as for simulation. At its most abstract level, the microprocessor may be depicted as shown in Fig. 13.12. It consists of a processor and an attached memory system. In turn, the processor is decomposed into a data path and a control unit. The data path includes functional units and registers; that is, circuits that operate on the data, shifting, adding, or multiplying. The control unit is an FSM with its associated registers. It interprets the instructions to manipulate the datapath.

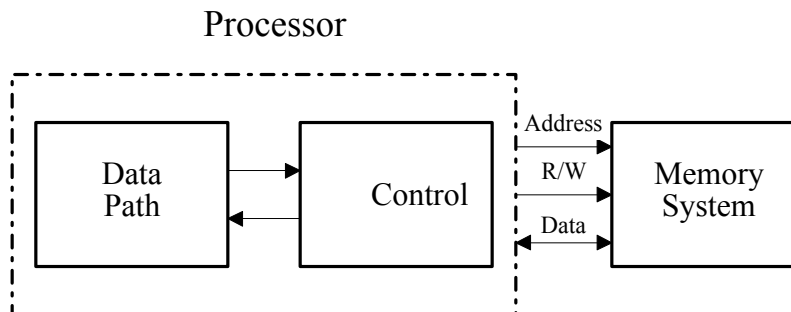


Figure 13.12 Generic Model of a Microprocessor

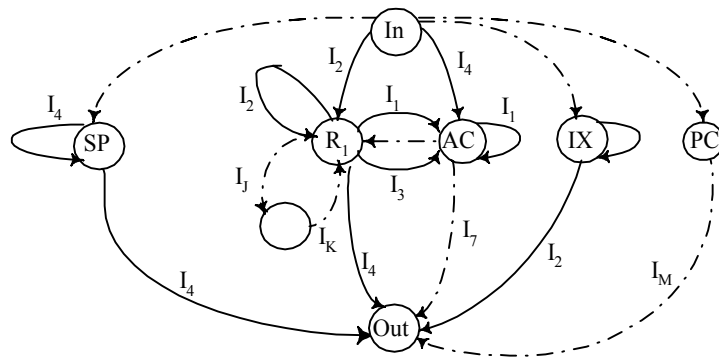


Figure 13.13 S-Graph Model for a Microprocessor

A microprocessor may also be represented by interaction between registers  $R = \{R_1, R_2, R_3, \dots\}$  according to an instruction set  $I = \{I_1, I_2, I_3, \dots\}$ . Both sets are used to construct a system graph, the S-graph  $G(R, I)$ , in which the vertices represent the registers and the edges represent the instructions [Thatte 1980]. An example of an S-graph is shown in Fig. 13.13. In addition to the registers, two of the nodes, *In* and *Out*, represent the outside world to the processor. They may be viewed as buses that connect the registers to memory locations and peripheral devices. Vertices may thus be connected by several edges since the registers may be used by several instructions.

Table 13.2 Example Instructions of S-Graph in Fig. 13.13

Instruction		Operation	Edges
$I_1$	ADD $A, R_1$	$A \leftarrow R_1$	$R_1 \rightarrow A$ $A \rightarrow A$
$I_2$	ADD $R_1, (IX)$	$R_1 \leftarrow R_1 + (IX)$	$IX \rightarrow OUT$ $IN \rightarrow R_1$ $R_1 \rightarrow R_1$
$I_3$	MOV $A, R_1$	$A \leftarrow R_1$	$R_1 \rightarrow A$
$I_4$	PUSH $R_1$	$SP \leftarrow R_1$ $SP \leftarrow SP + 1$	$SP \rightarrow OUT$ $R_1 \rightarrow OUT$ $SP \rightarrow SP$

Also, an instruction may result in more than one edge. Only a few registers are shown: the program counter (PC), the accumulator (AC), the index register (IX), the stack pointer (SP), and one data register (R1). The three instructions in Table 13.2 will be used to illustrate the description of the graph. An instruction may be represented by several edges. For example,  $I_1$  consists of an edge from R1 to A and from A to itself.  $I_2$  is represented by three edges.  $I_3$  is another instruction requiring the same transfer from R1 to A will be represented by another edge, from the first register to the second.

The more complex the set of instructions, the more complex is the graph. The graph might become too complex, thereby losing its effectiveness as a model. By drawing a relationship among the registers and among the instructions, it is possible to simplify the model [Brahme 1984]. The instructions are broken in to microcode. Several instructions might require the same microinstruction.

To contain the explosion of microprocessor states due to the increasing complexity of the instructions and of the datapath size, an abstraction is extracted directly from the behavioral level to model the processor. This model encapsulates the control flow as well as the effect of the datapath on the control in a form of a FSM. An example of this model is the extracted control flow machine model (ECFM) [Moundanos 1998].

Another model for a bit-slice microprocessor was also developed [Sridhar 1981]. A basic device,  $U$  (also called a *cell* or *slice*), performing a set of operations on  $n$ -bit operands is known to be *bit-sliced* if a system that performs the same set of operations on  $k$ -bit operands, where  $k = N \cdot n$ , can be built by interconnecting  $N$  copies of  $U$  in a normal way. In a *bit-sliced* microprocessor, the cell  $U$ , performs the functions of the ALU and the register file or scratchpad RAM of a computer. In past microprocessor designs, the use of bit slicing introduced structural simplicity and normality in the connections between IC chips at the printed circuit board level [AMD 1976]. Bit slicing has also been used within IC chips to achieve simplicity and compactness in many VLSI designs.

The models described in this section are used for testing microprocessors. However, before addressing this topic, we examine briefly the validation of these processors.

### 13.5.2 Microprocessor Validation

As pointed out in Chapters 1 and 5, simulation remains to be the most feasible verification means for VLSI design. However, with increased complexity, emulation can be on the order of six times faster than simulation, and this is why emulators are becoming more feasible for microprocessor verification. Emulators were used for Pentium microprocessors and Motorola 68060 microprocessors. The actual time required for verifying real-time operation of the latter processor was six weeks; simulation of the same system would have taken 13 years [Kumar 1995]. Emulators facilitate true concurrency since the traditional software simulation is run in sequential fashion. The emulator is constructed at the gate level obtained for the synthesis of an RTL model. Emulators built out of reprogrammable FPGAs are becoming common.

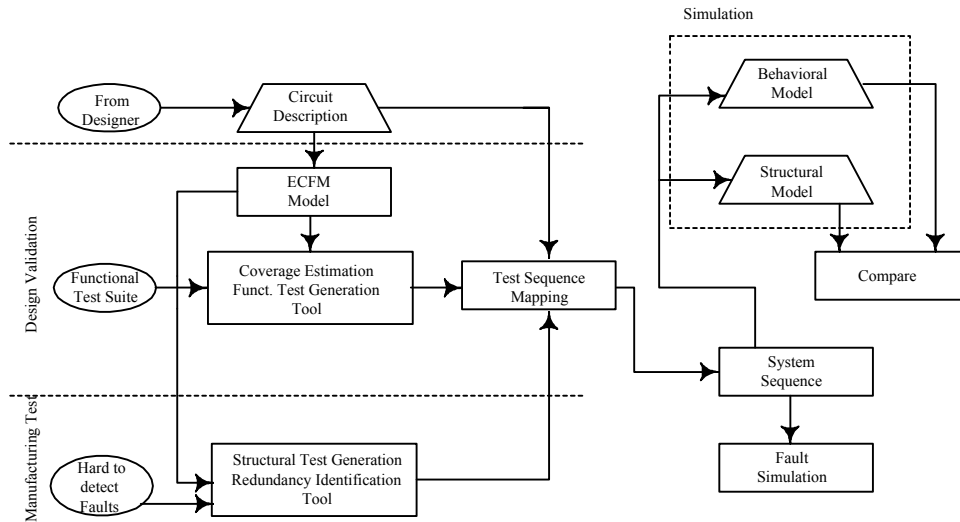


Figure 13.14 Unified Approach to Validation and Testing [Moundanos 1998]

As the complexity of microprocessors increases, it is more realistic to use a more abstract model such as the ECFM, which was described briefly, in the Section 13.5.1. The higher abstraction model has the advantage of being more general and can cover a wider range of processors. The benefit of the model is that it can also be linked to the structural level to guide in test pattern generation as described in the next section and illustrated in Fig. 13.14.

## 13.6 Testing Microprocessors

In the early years of microprocessors, ad hoc testing techniques were used. For example, they were tested by exercising some of their functionality. That is, every instruction is exercised for a selective set of operands. Thus the approach used for one microprocessor architecture could not be applied easily to another architecture. Also, no fault model was used, and hence it was difficult to assess the effectiveness of the test. On the other hand, testing microprocessors using traditional fault models requires extremely long test sets and is time consuming. Toward easing these difficulties, a functional fault model was developed and used successfully. One such approach proposed to test the CPU according to the following methodology [Robach 1976]. The system is divided into *macroblocks*, which in turn are subdivided into *microblocks*. Testing is carried out at the microlevel. The information is then passed to the macroblocks for fault detection and diagnosis. The emphasis in testing microprocessors has then shifted to instruction set verification.

### ***13.6.1 Instruction Set Verification***

The methodology aimed at testing the control unit under normal sequencing of the instructions was proposed by [Robach 1978]. The model used an FSM to represent the control commands and assumes that the results of these commands on the operands are directly observable. However, the operands are not directly available at the function units, and the results are not directly observable.

With the advent of the bit-sliced processor, a testing approach more suitable for this architecture has been developed based on the bit-slice model described above [Sridhar 1981]. The main idea was to develop the test for one slice, then arrange the slices in an iterative array, and develop a C-testable strategy for testing the array. The test patterns are generated by injecting faults in the datapath and the controller; for example, modifying the truth table of the combinatorial parts or the controller's state table of the controller. On the basis of this model, test patterns are generated to verify the different components of the datapath. The advantage of this method is that the efforts are concentrated on one slice and then generalized for the entire processor. However, the controller is not tested adequately except indirectly as the functional components are tested.

Toward a more generic approach to testing microprocessors, the S-graph model described above is used to verify the instruction set systematically [Thatte 1980]. The problem with the model was the increased complexity of the graph as mentioned in Section 13.5.1. An improvement over this approach was realized with a closer examination of the instructions. They can be broken into microcodes [Annaratone 1982]. Examples of these microcodes are listed in the last column of Table 13.2. Generation of the test patterns does not require the details of the instructions but operates on the microcode [Brahme 1984]. The consequences of executing the instructions determine which circuit components are to be tested by which instructions. Accordingly, the following microprocessor component can be tested: register decoding, instruction decoding and sequencing, data transfer functions and data storage, and other functional units such as the ALU. By relating a sequence of instructions with every one of these components, it is possible to generate the appropriate test patterns: the necessary data to decode the instructions and the data to place in the register, if necessary. For more details of test pattern generation using the model described above, refer to [Brahme 1984, Miczo 1986, Abramovici 1990, Rajsuman 1992].

Although this approach is a step in the right direction, as the instruction set proliferates, the model used becomes unmanageable. For this, a unified framework for multilevel and test generation was developed based on the ECFM mentioned for microprocessor modeling in Section 13.5.1 [Moundanos 1998]. Techniques used in formal

verification are used to generate test to exercise the control transitions. Test generation is performed on the ECFM model. This sequence may have to be augmented by traditional ATPG techniques. The framework is outlined in part in Fig. 13.14. A similar unified principle was cited as the key to the Pentium Pro's testing success [Carbine 1998].

**13.6.2 Testing the Datapath**

The datapath consists of a set of functional modules of different types: arithmetic functional modules (adders, multipliers, etc), and logic functional modules (logical operations, comparators, shifters). All modules interact among each other and with embedded memories using registers. To optimize on the use of the functional modules and the registers, a set of multiplexers are used to facilitate the flow. The different interactions are performed under the control unit. A model for the datapath as described here is shown in Fig. 13.15.

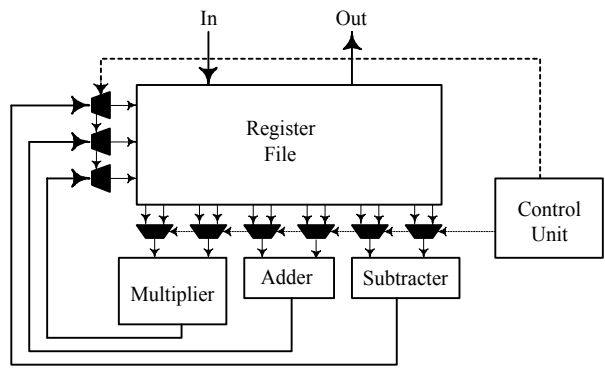


Figure 13.15 Datapath Model

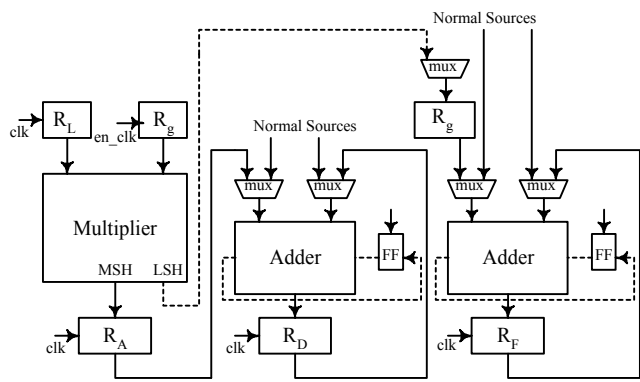


Figure 13.16 Testing the Datapath

Such tightly structured architecture causes several testability problems due to low controllability and observability of the embedded modules. There are known schemes to test the functional unit independent of each other. Some of the modules are modeled as iterative logic arrays (ILAs), some as one-dimensional (adders) and

others as two-dimensional (array multipliers). In Chapter 8 we addressed the testability of ILAs. As we have learned, an exhaustive test set is applied on the basic cell of the ILA. The test guarantees the detection of all combinational faults in the unit and the faults are then propagated through subsequent units in the array. This approach is illustrated in Fig. 8.13 for an  $n$ -bit adder consisting of a succession of full adder. The exhaustive test is applied to the first full adder (FA) and the response from the carry  $c$  is applied to the second FA. For maximum concurrency testing (MCT), the patterns on  $a_2$  and  $b_2$  are arranged to apply an exhaustive test on this second adder. As shown in the figure, the test set for the adder is independent of the number of bits, the datapath size. Another example, the array multiplier, was shown to be C-testable after a slight modification to the basic cell [Shen 1984]. Only 16 patterns were sufficient to test any word-size multiplier. The concept was also extended to the Booth multiplier.

However, the main problem with datapath testability is the inaccessibility of the embedded module. BIST has been shown to be effective in solving this problem since it reduces the interaction with the outside world and utilized existing registers for pattern generation and response compaction. An effective BIST scheme for datapath would be capable to performing testing at speed with high fault coverage and reasonable overhead. A generic BIST approach will result in a long test set whose effectiveness is to be assessed with a fault simulator. Also, it depends on the functional unit under test and the width of the datapath. A more efficient BIST would rely on the C-testability of the functional modules as described above. Some of the registers of the datapath are configured to generate the necessary test sets. Other registers can be configured as a SSA or a MISR. In addition, other types of compaction may be used, a count-based compaction [Ivanov 1992] or rotate carry addition [Rajski 1993]. The latter scheme was used successfully for BIST application to multipliers [Gizopoulos 1995].

The discussion above concentrated on the functional modules, however, the registers and multiplexers in the path of the tested modules are also tested fully. Those registers or multiplexers that were not exercised should be tested by extra patterns. BIST is also used for other circuitry such as the embedded ROMs [Zorian 1992] and the RAMs [Zorian 1994]. Use of BIST for datapath is very popular and has also been reported to be successful with special-purpose processors [Pichon 1996]. In this example, due to the large number of ROMs and RAMs, the test pattern for BIST application was software generated and stored in a ROM.

## 13.7 DFT Features in Modern Microprocessors

In the discussion above we have given the foundation of testing microprocessors based on generic architecture. The first known commercial processor that led in incorporating testability structures was the Motorola [Kuban 1984]. In the 386 processors, Intel introduced testing structure in their products [Gelsinger 1989]. At present, most DFT constructs and techniques are commonly utilized to test microprocessors. As a conclusion of this chapter, the testability features of some leading processor companies microprocessors are given next. This presentation is useful since it puts in perspective the different concepts learned so far in the book. In addition, it will give a sense of the real-life handling of testability issues for large and complex systems.

The information in remainder of this chapter is abstracted from a collection of papers on microprocessor testing that were presented at the International Test Conference in 1997 and 1998 [ITC 1997, 1998]. The highlights of these presentations were also published in two issues of *Design and Test of Computers* magazine [D&T 1997, 1998]. Testing covered the UltraSparc [Levitt 1997], Alpha 21164 [Bhavsar 1997, Stolicny 1998], Intel Pentium Pro [Carbine 1998], AMD-K6 [Fetherston 1998], IBM S/390 [Foote 1998], HP PA8500 [Brauch 1998], and M69060 [Kumar 1997].

### 13.7.1 Testing Sun Microsystems Processors

As a complex project, the design of Sun processors has numerous goals that are not compatible. The achievement of high performance coupled with reduced chip area conflicts with the need for a design that is easy to debug, to test, and to manufacture. The UltraSparc incorporates the following DFT constructs:

- Full-scan design for all units except one that features partial scan using one scan clock
- IEEE Standard 1149.1, which includes a special memory test mode
- All built-in devices accessible via the IEEE Standard 1149 test port
- Chips tested using standard testers

The approach to the design of the UltraSparc combined structural techniques that added 3.5% to the die area and a fully custom design in the precharged logic and memory arrays. These arrays used 44% of the total transistor count. Test pattern generation was accomplished through commercial tools and the percentage fault coverage was in the high 90s. Table 13.3 shows how the DFT structures were deployed in testing, debugging, and manufacturing.

Table 13.3 Taxonomy of (DFT) Uses [Levitt 1997]

Feature	Test	Debug	Manufacturing
Scan	√	√	√
IEEE 1149.1	√	√	√
SRAM test mode	√	√	√
I <sub>DDQ</sub>	√		√
Clock control	√	√	√
Observation bus	√	√	√
Process Control monitors	√		√

### 13.7.2 Testing Digital's Alpha 21164

The Alpha 21164 processor combines structured and ad hoc DFT solutions [Bhavsar 1997]. One of the features that complicated the design was the number of embedded RAMs that included extra cells to be used in memory repair. For this, a mix of hardware and software BIST was adopted. Attention was given, in particular, to the instruction cache that was designed with BIST and built-in self-repair (BISR). The RAM was organized as several columns of N x 1 arrays stacked side by side.

Table 13.4 Alpha 21164 Cost Parameter Summary [Bhavsar 1997]

Characteristic	%Cost
Die area	2.2
No. of transistors	
% total	0.5
% core logic	3
Number of chip pins	
% of all pins	2.6
% of signal pins	4.4
Design effort	5
Performance decrease	0

Another important feature was the observability LFSRs (OBL), which served as MISR and scan cells to observe during testing and at speed, a single snapshot of data for chip debugging. The chip has 27 OBLs that observe 550 internal nodes. They are organized in three scan chains to cover different geographic areas on the chip. The OBLs were configured to shorten the scan chain whenever needed. The cost of DFT is summarized in Table 13.4.

In addition, the chip also has a unique testability feature, the *test port*, which is provided solely for interfacing with the testability functions, including customer usable ones. Its 13 dedicated pins support three interface modes: normal, manufacturing, and debugging. Normal mode operation supports customer-usable test features. Two pins select the test mode. The boundary-scan port complies with IEEE Standard 1149.1 except for two deviations that actually enhance value to the end user:

- The optional reset pin has an internal pull down instead of pull up, as required by the standard.
- Two differential-oscillator input pins do not have any associated boundary-scan cells, which allows a meaningful test of the oscillator input pins.

### 13.7.3 Testing the Intel Pentium Pro

This high-performance Intel architecture microprocessor was designed for desktop, workstation and server applications [Carbine 1998]. Its unique business requirement of meeting very high production, performance, and test quality goals simultaneously strongly influenced its design-for-test (DFT) direction. A set of constraints limits their design teams' ability to use DFT and test generation techniques (full or partial scan and scan-based BIST). For example, an increase of the die by 15% would have cost Intel a new fab! Nevertheless, they are able to optimize the design for low die area, high performance, low power dissipation, high-quality test, and low test cost.

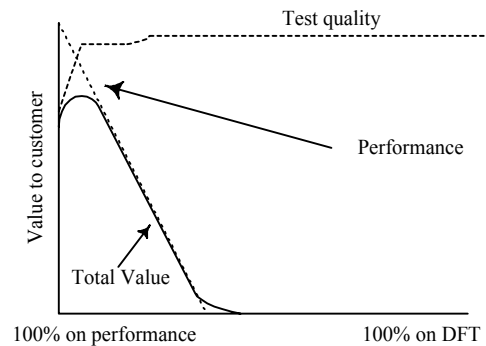


Figure 13.17 Intel: Performance versus Testability [Intel 1999]

Intel was guided by customer satisfaction for performance and testability goals that were conflicting in optimization of the design as illustrated in Fig 13.17. The intent was to follow the four important DFT principles:

1. Have zero performance impact
2. Have minimal die area impact

3. Have multiuse features wherever possible
4. Be designed in from the start; that is, coded and validated in the register-transfer-level (RTL) model

The last principle was the key to the Pentium Pro's success. Intel coded the DFT features into the RTL model, verified their schematics using schematic formal verification tools, and verified their functionality through RTL simulation.

Intel's processor includes the following DFT features:

- *JTAG test access port (TAP)*. The Pentium Pro processor TAP is a compliant implementation of IEEE Standard 1149.1, which includes seven public TAP instructions.
- *Scan-out*. Scan-out is a patented observation-only, scanlike technique that provides observability of internal nodes in the design. It works in two modes; a *snapshot mode* takes a single sample of the key processor state and shifts it out through the JTAG port while the processor is running, and a *signature mode* that enables continuous sampling and signature compression of the same key state elements in the design. Both modes can be used while the processor is running normally at full clock frequency without affecting its operation.
- *I<sub>DDQ</sub> mode*. This testing mode allows a private TAP instruction to be used to disable all devices that draw static current.

### 13.7.4 Testing AMD's K6

The heart of AMD's K6 processor is a reduced-instruction-set computer (RISC) core known as the enhanced RISC86 microarchitecture [Fetherston 1998]. From the onset, the designer team decided to make the AMD's K6 fully scannable, and therefore all internal storage elements are scannable and I/O cells are equipped with scan to support boundary scan and IEEE Standard 1149.1 compatibility. The internal storage element is a shared master, rising edge-triggered scan flip-flop. In normal operation, it operates as a simple flip-flop. In scan operation, the clock, which is embedded and controlled by a PLL, is held low and data are scanned using nonoverlapping shift clock pulses. AMD uses a commercial ATPG tool to create scan test patterns for stuck-at faults in their processor. By using a gate-level-fault simulation tool, the K6 contains just over 1 million simulator-primitive elements. The full-scan implementation makes combinational ATPG possible, thus providing a realistic opportunity to push the stuck-at-fault coverage goal for this large design to 100%.

The K6 also incorporates BIST into its DFT process. Each RAM array in the K6 processor has its own BIST controller. At power-on, the processor initializes the BIST controllers to a safe reset state. To start the testing, a single scan operation sets the StartBIST register in each BIST controller. BIST executes simultaneously on all arrays for the precomputed number of clock cycles that ensures completion for the largest array. Therefore, BIST execution time depends on the size of the largest array. For  $I_{DDQ}$  testing, a special procedure was followed to counteract the effect of high subthreshold current.

### 13.7.5 Testing IBM S/390

The DFT framework of the 500-MHz CMOS S/390 microprocessor uses a wide variety of tests and techniques to ensure the highest reliability of components within a system [Stolicny 1998]. Some of the same test patterns applied in chip manufacturing are applied again at the system test level. Scan-based design is the fundamental requirement for all these techniques; that is, all system latches are scannable and configurable into one or more scan chains. The chip uses full scan and follows most level-sensitive scan design (LSSD) rules. It uses both an edge-triggered latch and a race-free latch clocked by a single-system clock. The chip contains a logic BIST (LBIST) engine to allow logic testing with minimal tester support. The LBIST places random patterns into the scan chain and compresses the results into a signature register. Register arrays are tested through the scan chain, while nonregister memories are tested with a programmable RAM BIST (RAMBIST). During logic test, RAMs become transparent and data are allowed to pass directly from RAM inputs to RAM outputs, making test generation simpler and faster. The overall objective of the S/390 chip test is 99% static stuck-at-fault coverage and 90% dynamic test coverage. Dynamic test coverage is the ratio of the total number of transition faults detected to the total number of transition faults in the design.

LBIST patterns are the first to be fault simulated during logic test generation and detect the majority of the logic faults. On the S/390 chip set, test coverage is 95% static coverage with 256,000 LBIST patterns. The basic operation consists of loading the scan chains from the pseudorandom pattern generators (PRPGs) with scan clocks, applying system clocks to perform the actual logic test, then unloading the scan chains into the signature registers. As multiple patterns are applied, the load and unload operations combine into a single scan.

The RAMBIST engine is programmed more like a small microprocessor than a finite state machine. It contains a small microcode array initialized through the scan chain with the intended program/test application. Once

initialized, the RAMBIST decodes instructions from its microcode array and forms patterns to apply to the memory under test. Application of memory tests involve the scan initialization of the RAMBIST engines and microcode arrays, a series of clocks long enough to complete the test, and a scan-out of the resulting compression registers. Smaller memories have their results compressed into MISRs. With typical RAMBIST techniques, the set of tests applied is hard coded into the RAMBIST finite state machine. With programmable RAMBIST, unique memory tests can be accomplished at the chip, module, and system levels.

### 13.7.6 Testing Hewlett-Packard's PA8500

The PA8500 is a 0.25- $\mu\text{m}$  fabrication, superscalar processor [Brauch 1998]. The small geometry provides a significant increase in speed and allows the large first-level cache to be placed entirely on a chip as the CPU. On-chip memories communicate directly with the CPU, eliminating the need to connect memory I/Os to external pins. One option for testing on-chip memories is to take the additional steps needed to port the memory I/Os and use a direct-access test (DAT) technique. However, a large two- or four-word-accessible cache requires many I/O pins (or pads). In addition, the cache must receive a large amount of data at high speeds to achieve acceptable fault coverage. To achieve a fast and thorough production test, the cache test hardware's first requirement is the ability to perform March tests. In general, March tests are an effective way to find several kinds of functional faults. The second requirement is the ability to apply arbitrary patterns as part of more general characterization tests.

Hewlett-Packard must assure that their hardware can support tests such as Walking Ones/Walking Zeros and GALPAT. These tests can write a value to a single memory cell and then perform reads on the entire array. The written value is then "walked" through every cell in the memory. Address sequencing is a fundamental part of most memory tests, so it is important to provide a great deal of flexibility in this area. When combined with pseudorandom patterns, pseudorandom address sequences can detect faults that deterministic tests might miss. The final required test function is the ability to create a bitmap of the memory. Generating bitmaps means determining the location of one or more failing cells. In addition to helping with electrical characterization and debugging, a bitmap can be a powerful tool when used to isolate and document possible process problems.

## References

Abramovici, M., M. M. Breuer, and A. D. Friedman (1990), *Digital Systems Testing and Testable Design*, IEEE Press, Piscataway, NJ.

Abramovici, M. and C. Stroud, (1996), Using ILA BIST for FPGAs, *Proc. IEEE International Test Conference*, pp. 68 – 75.

Actel (1991), *ACT Family Field-Programmable Gate Array Databook*, Actel Corporation, Santa Clara, CA.

Altera (1992), *Applications Handbook*, Altera Corporation, San Jose, CA

AMD (1976), *An 2900 Bipolar Microprocessor Family*, AMD, Sunnyvale, CA.

Annaratone, M. A., and M. G. Sami (1982), An Approach to Functional Testing of Microprocessors, *Digest of Papers 12th Annual International Symposium on Fault-Tolerant Computing*, pp.158 – 164.

Bhavsar, D. K. and J. Edmondson (1997), Alpha 21164 Testability strategy, *IEEE Des. Test Comput.*, Vol. 14, No. 1, pp. 25 – 33.

Brahme, D. and J. A. Abraham (1984), Functional testing of microprocessors, *IEEE Trans. Comput.*, Vol C-33, No. 6, pp. 475 – 485.

Brauch, J. and J. Fleischman (1998), Design of cache test hardware on the HP PA8500, *IEEE Des. Test Comput.*, Vol. 15, No. 3, pp. 58 – 63.

Carbine, A. (1998), Pentium Pro Processor design for test and debug *IEEE Des. Test Comput.*, Vol. 15, No. 3, pp. 77 – 82.

Chan, P. K. and S. Mourad (1994), *Digital Design Using Field Programmable Gate Arrays*, Prentice Hall, Upper Saddle River, NJ.

D&T (1997), *IEEE Des. Test Comput.*, Vol. 14, No. 1, pp. 8 – 41.

D&T (1998), *IEEE Des. Test Computers*, Vol. 15, No. 3, pp. 56 –104

Feng, C., W. K. Huang, and F. Lombardi (1995), A New diagnosis approach for short faults in the interconnects, *Proc. IEEE Symposium on Fault-tolerant Computing*, pp. 331 – 338.

Fetherston, R. (1998), Testability features of the AMD-K6 microprocessor, *IEEE Des. Test Comput.*, Vol. 15, No. 3, pp. 64 – 69.

Foote, T. D. et al. (1998), Testing the 500-MHz IBM S/390 microprocessor, *IEEE Des. Test Comput.*, pp. 389 – 395.

Gelsing, P., S. Iyengar, J. Krauskopt, and J. Nadir (1989), Computer aided design and built in self test on the i486 CPU, *Proc. International Conference on Computer Design*, p. 199.

Gizopoulos, D., A. Paschalis, Y. Zorian (1995), An effective BIST scheme for carry-save and carry-propagate array multipliers, *Proc. 4th IEEE Asian Test Symposium*, pp. 286 – 292.

Gizopoulos, D., A. Paschalis, Y. Zorian (1996), An effective BIST scheme for datapaths, *Proc. IEEE International Test Conference*, pp. 76 – 85.

Hamdy et al. (1988), Dielectric based antifuse for logic and memory IC, *International Electron Device Meeting*, pp. 786 – 788.

Huang, W. K. and F. Lombardi (1996a), An approach for testing programmable/configurable Field programmable gate arrays, *Proc. 14th IEEE VLSI Test Symposium*, pp. 450 – 455.

Huang, W. K., X. T. Chen, and F. Lombardi (1996b), On the diagnosis of programmable interconnect systems: theory and application,” *Proc. 14th IEEE VLSI Test Symposium*, pp. 204 – 209.

Huang, W. K., F. J. Meyer, and F. Lombardi (1997), Testing memory modules in SRAM-based configurable FPGAs, *IEEE International Workshop on Memory Technology*, pp. 204 – 209.

Inoue, T., S. Miyazaki, and H. Fujiwara (1997), *Universal fault diagnosis for lookup table FPGAs*, Technical Report NAIST-IS-TR97020, Graduate School of Information Science, Nara Institute of Science and Technology. (<http://isw3.aist-nara.ac.jp/IS/TechReport/report/97020.ps>).

Inoue, T., S. Miyazaki, and H. Fujiwara (1998), *Universal Fault Diagnosis for Lookup Table FPGAs*, *IEEE Des. Test Comput.*, Vol. 15, No. 1, pp. 39 – 44.

International Test Conference (1997), *Proc. IEEE International Test Conference*.

International Test Conference (1998), *Proc. IEEE International Test Conference*.

Ivanov A. and Y. Zorian (1992), Count-based BIST compaction schemes and aliasing probability computation, *IEEE Trans. Comput. Aided Des.*, Vol.11, No.6, pp.768 –777.

Kautz, W. H. (1974), Testing for faults in wiring networks, *IEEE Trans. Comput.*, Vol. C-23, No. 4, pp. 358 –363.

Kuban, J. R. and W. C. Bruce (1984), Self-testing the Motorola MC6804P2, *IEEE Des. Test Comput.*, Vol. 1, No. 5, pp. 33 –41.

Kumar, J. (1995), Emulation verification of the M68060 for concurrent verification, *Proc. International Conference on Computer Design*, pp. 150 –158.

Kumar, J. (1997), Prototyping the M68060 for concurrent verification, *IEEE Des. Test Comput.*, Vol. 14, No. 1, pp. 34 –41.

Levitt, M. E. (1997), Designing UltraSparc for testability, *IEEE Des. Test Comput.*, Vol. 14, No. 1, pp. 10 –17.

Liu, T., F. Lombardi, and J. Salinas (1995), Diagnosis of interconnects and FPICs using a structured walking-1 approach, *Proc. 14th IEEE VLSI Test Symposium*, pp.256 –261.

Mao, W. et al. (1990), Quietest: a quiescent current testing methodology for detecting leakage faults, *Proc. IEEE International Test Conference*, pp. 280 –283.

Moundanos, D., J. A. Abraham, and Y. V. Hoskote (1998), Abstraction techniques for validation coverage analysis and test generation, *IEEE Trans. Comput.*, Vol. 47, No. 1, pp. 2 –14.

Needham, W. (1998), Microprocessor Testing Today, *IEEE Des. Test Comput.*, Vol. 15, No. 3, pp. 56 –57.

Pichon, F. (1996), Testability features for a submicron voice-coder ASIC, *Proc. IEEE International Test Conference*, pp.377 –385

Quicklogic (1993), *Very High-Speed Programmable ASIC*, Quicklogic, Santa Clara, CA.

Rajski, J., and J. Tyszer (1993), Test responses compaction in accumulators with rotate carry adders, *IEEE Trans. Comput. Aided Des. Integrated Circuits Sys.*, Vol. 12, No.4, pp.531 – 539, April

Rajsuman, R. (1992), *Digital Hardware Testing: Transistor-level Fault Modeling and Testing*, Artech House, Norwood, MA.

Renovell, M., J. Figueras, and Y. Zorian (1997a), Test of RAM-Based FPGA: Methodology and Application to the Interconnect, *Proc. 15th IEEE VLSI Test Symposium*, pp. 203 –237.

- Renovell, M. et al. (1997b), Test Pattern and Test Configuration Generation Methodology for the Logic of RAM-Based FPGA, *Proc. IEEE Asian Test Symposium*, pp. 254–259.
- Renovell, M. et al. (1998a), SRAM-Based FPGA: Testing the LUT/RAM Modules, *Proc. IEEE International Test Conference*, pp.1102–1111.
- Renovell, M. et al. (1998b), Testing the Interconnect of RAM-Based FPGAs, *IEEE Des. Test Comput.*, Vol. 15 No. 1, pp. 45–50.
- Renovell, M. et al. (1999), SRAM-Based FPGA: Testing the Embedded RAM Modules, *Jl of Electron. Test.: Theory Appli.*, Vol. 14, No. 1/2, pp. 159–167.
- Robach, C. and G. Saucier (1978), Dynamic testing of control units, *IEEE Trans. Comput.*, Vol. C-27, No. 7, pp. 617–623.
- Shen, J. P. and F. J. Ferguson F. J. (1984), The design of easily testable VLSI array multipliers, *IEEE Trans. Comput.*, Vol. C-33, No. 6, pp. 554–560.
- Sridhar, T. and J. P. Hayes (1981), A functional approach to testing bit-sliced microprocessors,” *IEEE Trans. Comput.*, Vol. C-30, No.8, pp. 563–571.
- Stolicny, C. (1998), Alpha 21164 Manufacturing Test Development and Coverage Analysis” *IEEE Des. Test Comput.*, Vol. 15, No. 3, pp. 98–104.
- Stroud, C. et al. (1996), Using ILA Testing for BIST in FPGAs, *Proc. IEEE International Test Conference*, pp. 68–75.
- Stroud, C. et al. (1997), BIST-Based Diagnostics of FPGA Logic Blocks, *Proc. IEEE International Test Conference*, pp. 539–54.
- Stroud, C. et al. (1998), Built-In Self Test of FPGA Interconnect, *Proc. IEEE International Test Conference*, pp. 404–411.
- Thatte, S. M. and J. A. Abraham (1980), Test generation for microprocessors, *IEEE Trans. Comput.*, Vol. C-29, No.6, pp. 429–441.
- Trimberger, S. M., (Ed.) (1994), *Field-Programmable Gate Array Technology*, Kluwer Academic, Hingham, MA.
- Wang, S. J. and T. M. Tsai (1997), Test and Diagnosis of Faulty Logic Blocks in FPGAs, *International Conference on Comput. Aided Des., ICCAD97*, pp. 722–727.
- Xilinx (1989), *The Programmable Logic Data Book*, Xilinx, Inc.
- Zhao, L., D. M. H. Walker, and F. Lombardi (1998), Detection of Bridging Faults in Logic Resources of Configurable FPGAs Using IDDQ, *Proc. IEEE International Test Conference*, pp. 1037–1046.
- Zorian, Y., and A. Ivanov (1992), An Effective BIST Scheme for ROMs, *IEEE Trans. Comput.*, Vol.41, No.5, pp.646–653.
- Zorian, Y., and A.J. Van De Goor and I. Schanstra (1994), An Effective BIST Scheme for Ring-Address Type FIFO's, in *Proc. IEEE International Test Conference*, pp.378–387

## Problems

13.1 Discuss the suitability of using a functional fault model approach to testing microprocessors and FPGAs. State the benefits and the drawbacks.

13.2 In a paper on testing an 8-bit microprocessor, [Thatte 1980] reported the results of simulating a sample SSFs. The fault coverage obtained was 96%. Do you consider this coverage acceptable? Why or why not? Using the same methodology, is it possible to reach such fault coverage for modern microprocessors

13.3 In a RAM-based FPGA, one RAM cell connecting two metal segments is defective and the two metal segments cannot be connected. What are the consequences of such a failure? Consider different scenarios for the location of the defect in the logic circuit.

13.4 Consider the testing of the LUT of a Xilinx FPGA. In Fig. 13.6*b*, a configuration was recommended to connect the LBs in a one-dimensional array and shift the results through the LB's flip-flops. Use the schematic of Fig. 13.3*a* to detail the connection of two consecutive LBs.