Built-In Self Test for Digital Signal Processor Cores in Virtex-4 and Virtex-5 Field Programmable Gate Arrays

by

Mary Deepti Pulukuri

A thesis submitted to the graduate faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
August 9th, 2010

Keywords: Built-In Self-Test, FPGA, DSP,
Adder, Multiplier

Approved by

Charles Stroud, Chair, Professor of Electrical and Computer Engineering
Adit Singh, Professor of Electrical and Computer Engineering
Victor Nelson, Professor of Electrical and Computer Engineering
Vishwani Agrawal, Professor of Electrical and Computer Engineering

Abstract

Current Field Programmable Gate Arrays (FPGAs) incorporate special cores, apart from logic, such as digital signal processor (DSP) cores. The DSP cores can be cascaded to implement complex functions. An effective test approach for testing the logic and configuration memory associated with these embedded cores is essential. The thesis presents an effective approach for testing digital signal processor cores embedded in Virtex-4 and Virtex-5 FPGAs using Built-In Self-Test (BIST) methodology. Since the BIST circuitry can be programmed in the logic present inside the FPGA that is not being tested at the time, there is no area overhead or performance penalty.

The implementation and verification of the developed BIST configurations was done on various families and sizes of Virtex-4 and Virtex-5 FPGAs. The developed BIST configurations also detected manufacturing faults in some of the Virtex-4 engineering sample parts.

Acknowledgements

I would like to thank Dr. Stroud for his guidance and support throughout my research for my thesis. I am grateful for his teaching that trained me to be a better engineer. I would like to thank Dr. Singh, Dr. Nelson and Dr. Agrawal for serving on my committee and for their helpful suggestions in improving my thesis. I would like to thank my colleagues Brad, Joey, Jie and Alex for their valuable advice and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the advancement of semiconductor manufacturing technology and the reduction of feature size from 4 microns to 45 nanometers, logic design is becoming denser with the integration of billions of transistors on a single integrated circuit. An example of such a dense logic circuit is the field programmable gate array (FPGA)[1].

As the complexity of integrated circuits increases, the challenges in testing also increase [2]. Testing such complex integrated circuits by the user is a challenging problem since the manufactures of FPGAs provide limited information about the internal circuitry. Hence the challenge lies in figuring out the architecture of the logic resources and then applying accurate tests to ensure complete testing of the FPGA.

1.1 Overview of FPGAs

FPGAs have been popular since the mid 1980s for implementing any complex digital logic design. The ability of the FPGA to be reprogrammed easily and quickly without changing the fabrication or the wiring makes the use of the FPGA very flexible [3]. Over the years the FPGA architecture has increased in size and complexity. The number of transistors in the largest FPGA now is over a billion [1].

Figure 1.1 shows the general architecture of an FPGA. The FPGA is a two dimensional array of logic blocks. The logic blocks can be programmed to implement any arbitrary digital logic circuit [4]. The basic element of the FPGA is the configurable logic block (CLB) [5]. The CLBs consists of look up tables (LUTs), multiplexers and flip-flops. They can be configured to

perform any desired combinational or sequential logic function [6]. Combinational logic is implemented using LUTs and multiplexers. Sequential logic is implemented using flip-flops [8]. The number of inputs to the LUT is fixed for a given FPGA but varies with different FPGAs and ranges from three to six [1]. Some of the LUTs can be programmed to function as small random access memory (RAM) units. The Input/Output blocks (IOB) located on the extreme and center columns [6] [7] of the device provide access to the logic blocks integrated inside the FPGA. The number of CLBs and IOBs differs based on the family and how big the device is. The number of CLBs in Xilinx Virtex-4 FPGAs varies from 1,536 to 22,272 [6] and the number of CLBs in Xilinx Virtex-5 FPGAs varies from 2,400 to 25,920 [7]. The memory modules in Xilinx Virtex-4 FPGAs are 18KB dual port RAMs [6] and 36KB dual port RAMs in Xilinx Virtex-5 FPGAs [7]. These memory modules, called block RAMs (BRAMS), can be combined to provide larger memory blocks.

The Xilinx Virtex-4 and Xilinx Virtex-5 FPGAs also have digital signal processor (DSP) cores incorporated in their structures [6] [7]. The DSP cores are used to implement DSP applications in a faster and more efficient manner compared to the DSP implementation in the earlier family of Xilinx Virtex-2 FPGAs [9]. The DSP core architecture mainly consists of a 2-port multiplier, a 3-port adder/subtractor unit and a 48-bit accumulator register [9] [10]. The multiplier in Xilinx Virtex-4 FPGA is an 18x18-bit two's complement multiplier [9] and Xilinx Virtex-5 FPGAs incorporate a 25x18-bit two's complement multiplier [10]. A common function of the DSP core is the multiply and accumulate (MAC) operation. Besides the multiplier and the adder, the DSP cores in Xilinx Virtex-5 FPGAs also have a 48-bit comparator unit and an Arithmetic & Logic Unit (ALU) mode of operation that is used to implement 48-bit boolean logic functions [10]. Multiplexers select different input/output paths within the DSP core. The

DSP cores can also be cascaded to facilitate the implementation of larger input/output functions [9] [10]. The number of DSP slices in Virtex-4 FPGAs ranges between 32 and 512 and the number of DSP slices in Virtex-5 FPGAs ranges between 32 and 1,056.

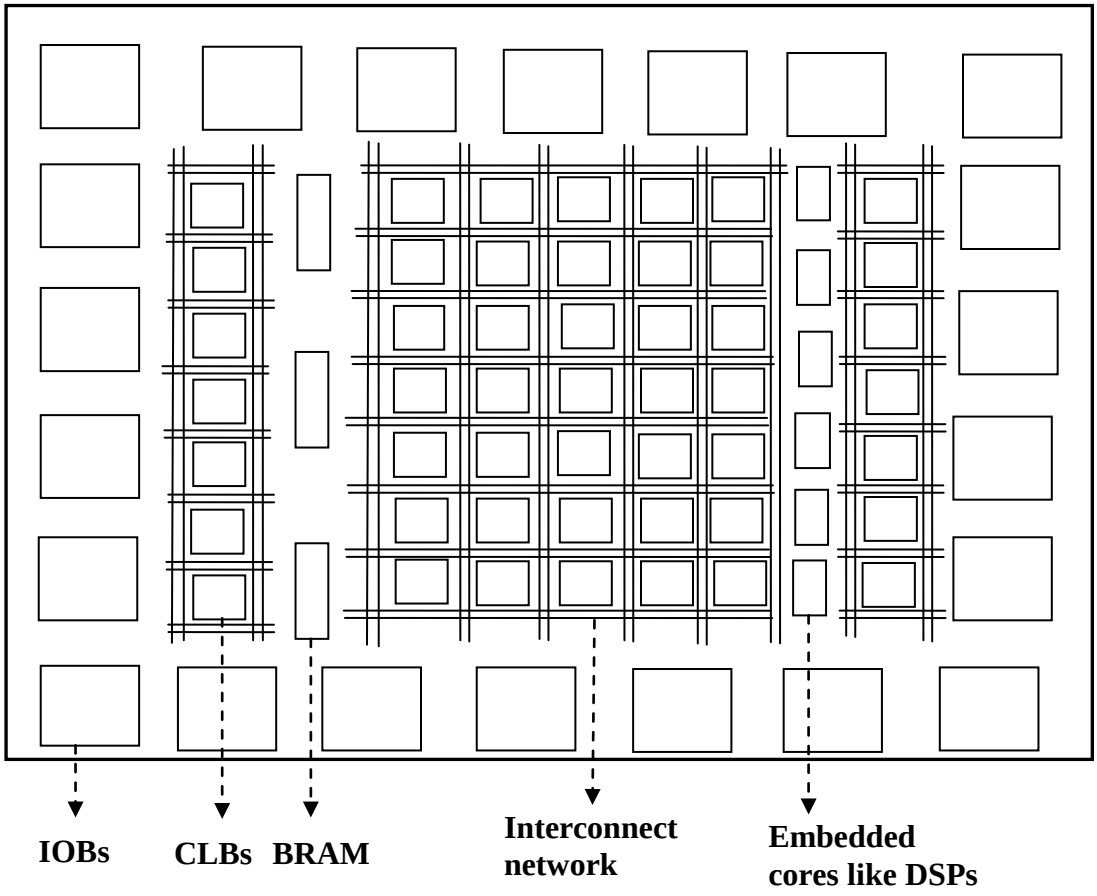


Figure 1.1 General Architecture of an FPGA

The logic blocks are interconnected by a series of horizontal and vertical routing lines [11]. The routing lines differ in lengths based on the number of logic blocks they span [1]. The routing channel between the logic blocks is determined by a matrix of programmable switches called programmable interconnect points (PIPs) [1] [11]. The logic blocks and the interconnection between them can be easily reprogrammed by changing the configuration data that is downloaded to the FPGA [4].

1.2 Overview of Built-In Self-Test

Because of the increased use of very large scale integrated (VLSI) circuits in digital systems, the reliability of these circuits is crucial. Hence the need for test methods at lower costs. But the increased complexity of the digital systems makes testing expensive [12]. There are different ways of testing the FPGA. In *external testing* the circuit that generates test patterns for the circuit under test and the circuit that observes the response of the circuit under test are external to the circuit under test. In *built-in self-test* (BIST) the test pattern generation circuit and the circuit that analyzes the output response of the circuit under test are internal to the circuit under test. In *offline testing* [1], the FPGA is tested while the system is not in its usual mode of operation. In *application-dependent testing* [1] the FPGA is tested for the specific system function that is being implemented. In this case, the design for testability (DFT) circuitry is implemented in the digital system that is being implemented in the FPGA. This increases the area occupied by the digital system circuit that is being implemented.

BIST is a DFT technique where test circuitry is implemented in the FPGA itself. Figure 1.2 shows the general BIST architecture [1]. The test pattern generator (TPG) generates test patterns for completely testing the device under test. The output response analyzer (ORA) observes the output response of the device under test for the input test patterns and reports any failures due to faults in the device under test. The test controller coordinates the operation and execution of the TPG, ORA, and device under test.

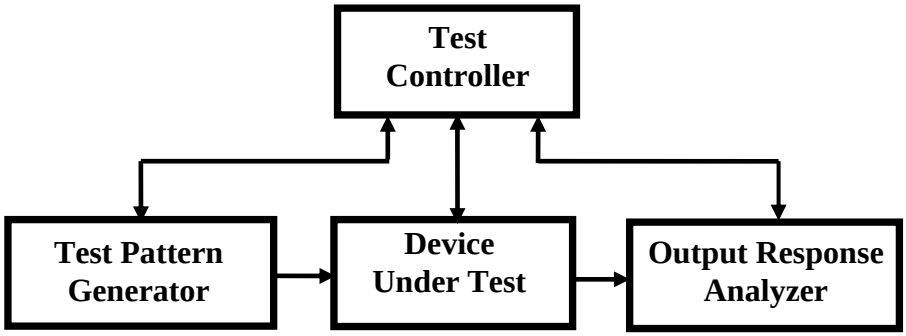


Figure 1.2 BIST Architecture [1]

1.3 Overview of FPGA BIST

The regularity in the structure of the FPGA makes pseudo-exhaustive testing possible without the need for expensive fault simulation [13]. In BIST, the FPGA is tested using a series of test configurations. The test configurations are repeatedly programmed into the FPGA to ensure that all the operational modes of the FPGA are thoroughly tested and the device functions fault-free irrespective of the system function that will be implemented [1] [13].

Some of the logic blocks in the FPGA that are not under test are configured as TPGs and ORAs [13]. Sometimes faults can go undetected if there are faults in a logic block that is configured as part of the TPG. Faults can also go undetected if faults exist in the interconnection between the TPG and the block under test (BUT). Faulty logic blocks in the TPG or faults in the interconnection between the TPG and BUT fail to generate the desired test patterns to completely test the BUT. To avoid missing the detection of any fault due to a faulty TPG, two or more TPGs are used [1]. Two ORAs observe the output response of every BUT which is also compared to the output responses of two other BUTs. As shown Figure 1.3 each BUT is observed by the ORA beside it and the ORA below. The bottom BUT in the array is observed by the ORA beside it and the ORA at the top of the array which also observes the output response of the top BUT of the array. This comparison of output responses is called *circular comparison* and is done to help

locate the faulty BUT [1]. At the end of the BIST sequence, the ORA contents, the BIST results, can be retrieved to detect and determine individual BUT failures using diagnostic procedures. [1] [13].

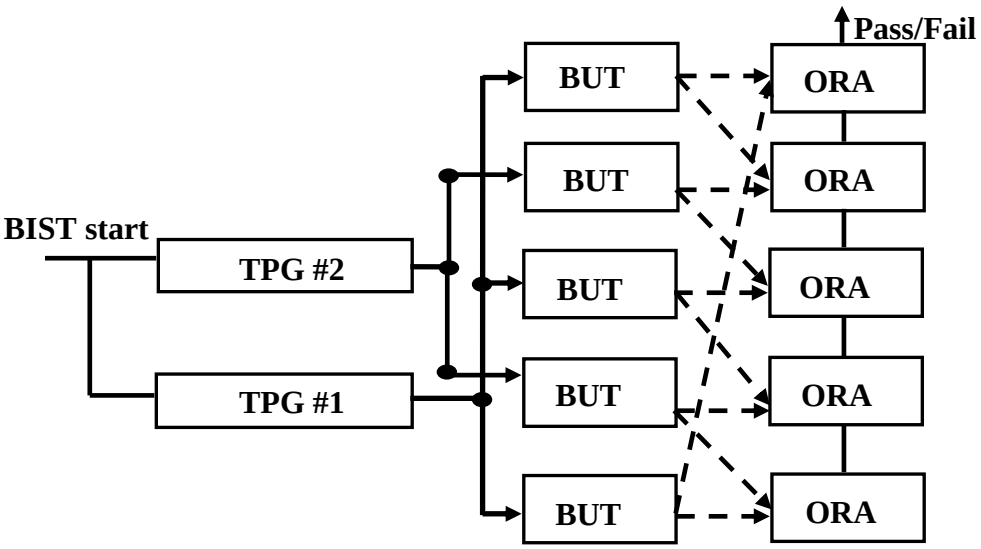


Figure 1.3 FPGA BIST Architecture [13]

1.4 Thesis Statement

Although some research has been done on BIST for DSP cores in general [14], there is little literature on BIST for DSP cores in FPGAs. However, prior research has been done in designing BIST algorithms for adders and multipliers. An adder BIST approach is presented in [15] and multiplier BIST approaches are presented in [16], [17] and [18]. The challenge in testing the DSP cores in the FPGA lies in testing the adder and the multiplier circuitry, as the remaining components in the DSP core, such as multiplexers and flip-flops, can be easily tested.

The research work presented in this thesis discusses the development, architecture and operation of BIST implementations for DSPs in Xilinx Virtex-4 and Virtex-5 FPGAs. This is achieved by making improvements on the previous work done for adders and multipliers to generate more effective test patterns and to keep the test time as low as possible independent of the specific adder and multiplier architectures. BIST configurations for testing the DSP cores in

Xilinx Virtex-4 and Xilinx Virtex-5 FPGA devices are generated based on the architecture presented in the data-sheets provided by the manufacture. The resulting BIST configurations are downloaded and executed in the FPGA. Effectiveness of the BIST configurations is established via fault injection in the configuration memory of actual hardware. [29]

The remaining chapters in the thesis are organized as follows: Chapter 2 presents an overview of the architecture of Virtex-4 and Virtex-5 FPGAs as well as their embedded DSP cores. It also presents prior work done in testing DSPs, multipliers and adders. Chapter 3 presents the architecture and operation of BIST designed for testing the DSP cores in Virtex-4 and Virtex-5 FPGAs. Chapter 4 presents the actual implementation of this BIST architecture in Virtex-4 and Virtex-5 FPGAs. It also presents the results and fault coverage of the BIST. Chapter 5 summarizes and concludes the thesis with suggestions for future work.

Chapter 2

Background Information

The ease of reprogramming an FPGA makes it attractive for the implementation of any complex logic system. With the incorporation of embedded memories and specialized cores for signal processing, FPGAs can be used for almost any application [25]. This chapter presents the architecture of the logic resources used to implement the TPGs and ORAs for BIST for DSP cores in Virtex-4 and Virtex-5 FPGAs and explains the architecture of the DSPs cores. This chapter also presents prior work done in testing multiplier and adder logic functions which are also used in DSPs.

2.1 Configurable Logic Blocks in Virtex-4 and Virtex-5 FPGAs

The TPGs and ORAs for BIST for DSP cores in Virtex-4 and Virtex-5 FPGAs are implemented in CLBs. The Virtex-4 CLB comprises four slices. Each slice is connected to a switch matrix through which it accesses the global routing resources. Figure 2.1 shows the simplified architecture of a Virtex-4 CLB. Each pair of CLB slices are arranged in two separate columns. The two slices in the left column are called SLICEMs because they also function as small memories and the two slices in the right column are called SLICELs since they function as logic only [6].

Each slice has two 4-input look up tables (LUTs), two memory elements, a carry chain and multiplexers. The memory elements can function as edge-triggered D flip-flops or as level-sensitive latches. The D flip-flops can either be driven by the output of the LUT or can be driven by the inputs to the slice [6]. The multiplexers in the CLB slices are used to combine the LUTs

within a CLB or in different CLBs to be able to implement higher input logic functions. The carry chain in the slices enables faster addition and subtraction [6].

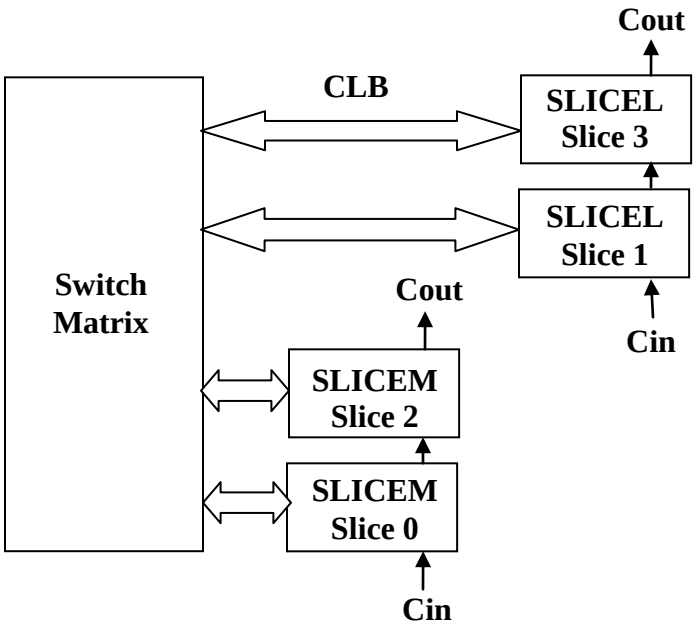


Figure 2.1 Simplified Architecture of a Virtex-4 CLB [6]

The Virtex-5 CLB comprises only two slices. Like the Virtex-4 CLB, the Virtex-5 CLB slices are connected to the switch matrix through which the global routing resources can be accessed. Figure 2.2 shows the simplified architecture of a Virtex-5 CLB. Each slice arranged in a separate column, is independent from the other, and has separate carry chains [7]. Each slice has four 6-input LUTs, four memory elements, multiplexers, and a carry-chain. Each slice has three multiplexers which can be used to combine up to four LUTs to be able to implement logic functions of up to eight inputs. Higher input logic functions can be implemented by using more slices. The carry chain with its dedicated carry logic enables fast addition and subtraction [7]. The memory elements in Virtex-5 CLBs are similar to the memory elements in Virtex-4 CLBs. They can be configured as edge-triggered D flip-flops or as level sensitive latches by user controlled configuration memory bits. They can be driven by the output of the LUTs or by the slice inputs [7].

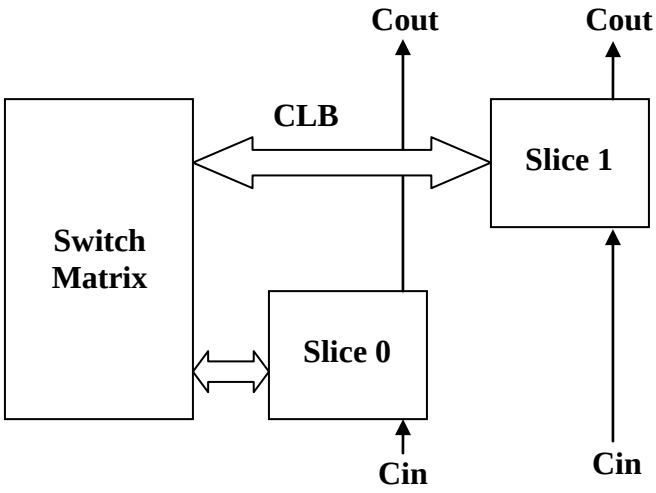


Figure 2.2 Simplified Architecture of a Virtex-5 CLB [7]

2.2 Architecture of DSP cores in Virtex-4 and Virtex-5 FPGAs.

Virtex-4 and Virtex-5 FPGAs incorporate DSP cores in their architectures and can be used for implementing large math functions, DSP applications such as finite impulse response (FIR) filters or to perform complex arithmetic computation without the need of using the general FPGA logic [9]. The architecture of a DSP tile in Virtex-4 FPGAs is shown in Figure 2.3 where two DSP slices form a DSP tile. Each DSP slice has an 18×18-bit two's complement multiplier that generates two 36-bit partial products. The A and B input ports in the DSP slice provide 18-bit access to each port of the multiplier. The final stage adder of the multiplier is separated from the multiplier and is incorporated in a separate three-port 48-bit adder/subtractor. The C input port is shared by both the DSP slices in the DSP tile and provides 48-bit access to the adder/subtractor through the 48-bit Y and Z multiplexer busses [9]. The partial products from the multiplication process are sign-extended to 48-bits and summed in the adder/subtractor. The partial products are fed to the adder/subtractor via the 48-bit X and Y multiplexer busses. The accumulator register, denoted by P in Figure 2.3, provides the only other 48-bit access to the

adder/subtractor through the X and Z multiplexer busses. Seven OPMODE signals dynamically control the select inputs to the X, Y and Z multiplexers [9].

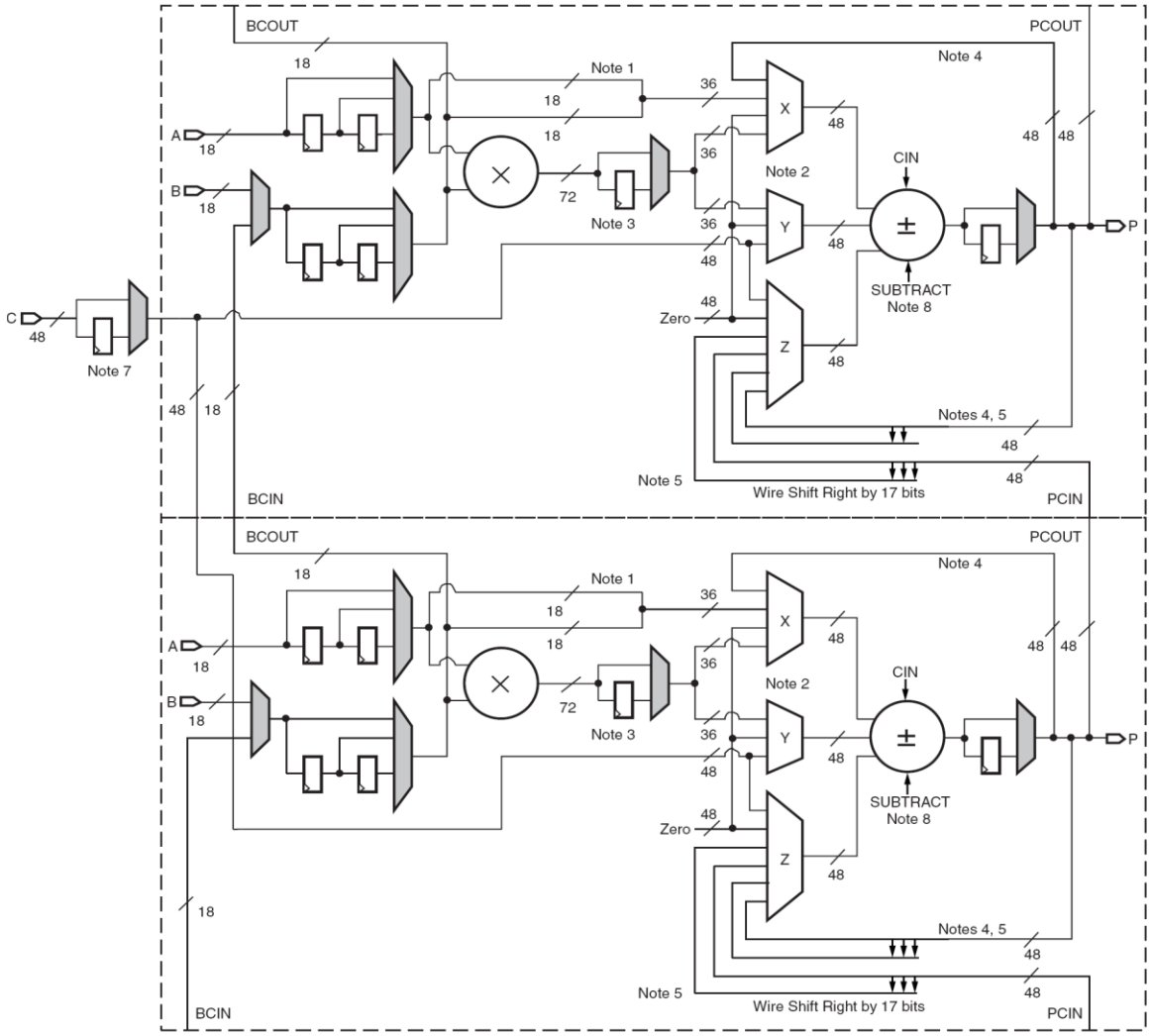


Figure 2.3 DSP Tile in Virtex-4 Devices [9]

The adder/subtractor performs P=Z±(X+Y+Cin) and produces a 48-bit two's complement result [9]. Here P is the output port, Cin is the carry-in input, and X, Y and Z are 48-bit multiplexer busses. The subtract input to the adder/subtractor shown in Figure 2.3 chooses between add or subtract operation of the adder/subtractor. A logic 1 on the subtract input chooses the subtract operation and a logic 0 on the subtract input chooses the add operation [9].

The data input paths, the control signal input paths and the output paths of the DSP slice have optional pipeline registers. Each pipeline register introduces a delay of one clock cycle in the path. The number of pipeline registers in the path can be controlled by user-defined configuration memory bits that control the select inputs to the shaded multiplexers in Figure 2.3 [9]. Maximum clock frequency is achieved when all pipeline registers are included in the I/O paths of the DSP slice. The A and B input ports can have up to two pipeline registers in their paths. The C and P ports can have up to one pipeline register. The input control signals paths that select the input paths to the adder/subtractor can have up to one pipeline register in their paths. The output of the multiplier also has a pipeline register (Mreg) as shown in Figure 2.3 (next to note 3) [9]. The Mreg introduces a clock cycle delay before the partial products are summed in the adder/subtractor.

The DSP slices in a column of DSPs can be cascaded to form larger DSPs. The B and P ports in a slice can be cascaded to the slice above. A user-defined configuration memory bit selects the B-input source to be direct or cascaded from the slice below. OPMODE values dynamically select cascading of the P port at the input to the Z multiplexer [9]. Table 2.1 illustrates all possible OPMODE values that control the inputs the X, Y and Z multiplexers in Virtex-4 and Virtex-5 FPGAs.

The DSP slice in Virtex-5 FPGA has the same functionality as the DSP slice in Virtex-4 FPGA with some additional features. The simplified architecture of a single Virtex-5 DSP slice is shown in Figure 2.4. The DSPs in Virtex-5 FPGAs incorporate a larger 25×18-bit multiplier. The A input port of the Virtex-5 DSP slice is 30-bits wide and the least significant 25 bits of the A port provide 25-bit access to the multiplier [10]. The C port is independent to both the DSP

slices where each slice has its own 48-bit C port. The A and B ports can be concatenated to provide another 48-bit access to the adder/subtractor [10].

Table 2.1 OPMODE Values for Virtex-4 and Virtex-5 FPGAs [9] [10]

Opmode values for the X-multiplexer outputs				
Z Opmode[6:4]	Y Opmode[3:2]	X Opmode[1:0]	X output	Comments
xxx	xx	00	0	default
xxx	01	01	M	When the multiplier is used
xxx	xx	10	P	
xxx	xx	11	A:B	
Opmode values for the Y-multiplexer outputs				
Z Opmode[6:4]	Y Opmode[3:2]	X Opmode[1:0]	Y output	Comments
xxx	00	xx	0	default
xxx	01	01	M	When the multiplier is used
xxx	10	xx	48'fffffffffff	Used for the logic unit bitwise operations (Illegal selection for Virtex-4)
xxx	11	xx	C	
Opmode values for the Z-multiplexer outputs				
Z Opmode[6:4]	Y Opmode[3:2]	X Opmode[1:0]	Z output	Comments
000	xx	xx	0	default
001	xx	xx	PCIN	
010	xx	xx	P	
011	xx	xx	C	
100	10	00	P	Used for MACC extend (Illegal selection for Virtex-4)
101	xx	xx	17-bit shift PCIN	
110	xx	xx	17-bit shift P	
111	xx	xx	xx	Illegal selection for Virtex-4 and Virtex-5

The adder/subtractor in Virtex-5 DSPs has been extended to function as a two-input 48-bit logic unit but the architecture of the basic adder/subtractor in DSPs of Virtex-5 FPGAs is same as the architecture of the adder/subtractor in DSPs of Virtex-4 FPGAs. ALUMODE control signals select between the adder/subtractor/logic unit operation [10]. The subtract signal does not exist as a unique input in DSP slices of Virtex-5 FPGAs. Instead, an ALUMODE value of

“0000” selects the add operation defined by the equation P=Z+X+Y+Carryin, where X, Y and Z are 48-bit multiplexer buses. An ALUMODE value of “0011” selects the subtract operation defined by the equation P=Z–(X+Y+Carryin) [10]. Table 2.2 illustrates the ALUMODE values for all the adder/subtractor logic equations that can be implemented.

Table 2.2 ALUMODE Values Determining the Adder/Subtractor Operation [10]

ALUMODE[3:0]	DSP operation
0000	Z + X + Y + CARRYIN
0011	Z – (X + Y + CARRYIN)
0001	-Z + (X + Y + CARRYIN) - 1
0010	Not (Z + X + Y + CARRYIN)

The bitwise logic operations performed by the logic unit include bitwise logical AND, OR, NOT, NAND, NOR, XOR and XNOR operations. ALUMODE inputs along with OPMODE[3:2] select the type of logical function as summarized in Table 2.3 [10]. Like the DSPs in Virtex-4 devices, the DSPs in Virtex-5 devices also have pipeline registers in their I/O paths and control signal paths. The A and B ports can have up to two pipeline registers, the C and P ports can have one pipeline register, the control signal paths can have one pipeline register and the Mreg pipeline register at the output of the multiplier, can be included by the user based on the performance desired. Higher performance is achieved when all the pipeline registers are included. Multiplexers that are controlled by configuration bits select the number of pipeline registers in these paths [10].

A 48-bit pattern detector is incorporated for comparison of two 48-bit patterns and is used for applications such as convergent rounding, overflow/underflow detection for saturation arithmetic, and auto resetting counters/accumulators [10]. The output of the DSP slice can be compared with a 48-bit pattern specified by the user. The pattern to the DSP slice can be provided through the C port or can be specified in the configuration memory bits. The output

Patterndetect goes to a logic '1' if the output of the DSP slice matches the pattern, and the output Patternbdetect goes to logic '1' if the output of the DSP slice matches the complement of the pattern [10]. A mask can be used to hide certain bits in the pattern detector. The bits hidden by the mask are not considered during comparison. Like the pattern, the mask can be provided through the C port or can be specified in the configuration memory bits [10]. The overflow and underflow flags are set by the DSP slice when the output of the adder/subtractor goes beyond a range of patterns determined by the number of 1s in the mask. If N is the number of 1s in the mask, the pattern values range from positive 2^N to negative 2^N-1. When addition goes beyond 2^N, the Patterndetect output switches from logic '1' to logic '0', which causes the overflow flag to be set. When subtraction goes beyond 2^N-1, the Patternbdetect output switches from logic '1' to logic '0', which causes the underflow flag to be set [10].

Table 2.3 Control Values for Logic Functions in Virtex-5 FPGAs [10]

OPMODE[3:2]	ALUMODE[3:0]	Logic function
00	0100	X XOR Z
00	0101	X XNOR Z
00	0110	X XNOR Z
00	0111	X XOR Z
00	1100	X AND Z
00	1101	X AND (NOT Z)
00	1110	X NAND Z
00	1111	(NOT X) OR Z
10	0100	X XNOR Z
10	0101	X XOR Z
10	0110	X XOR Z
10	0111	X XNOR Z
10	1100	X OR Z
10	1101	X OR (NOT Z)
10	1110	X NOR Z
10	1111	(NOT X) AND Z

The SIMD (Single Instruction Multiple Data) mode is used to split the addition/subtraction/logic unit into two 24-bit (two24) or four 12-bit (four12) adder/subtracter/logic units. The adder/subtracter unit has two independent carryout signals and

four independent carryout signals in the two24 and four12 modes respectively. When used as in single 48-bit adder/subtractor unit mode, there is only one carryout signal [10].

The A port, along with the B and P ports, can be cascaded in DSPs of Virtex-5 FPGAs [10]. Cascade signals CARRYCASCIN and CARRYCASCOUT are used to implement 96-bit adders, subtractors or logic units. The cascade signals such as MULTSIGNIN and MULTSIGNOUT are used to extend the multiply and accumulate (MACC) function to create 96-bit accumulators. The most significant bit of the output of the multiplier is cascaded through its MULTSIGNOUT port to the MULTSIGNIN port of the DSP slice above. The OPMODE value for the "MACC extension" feature is given in Table 2.1 [10].

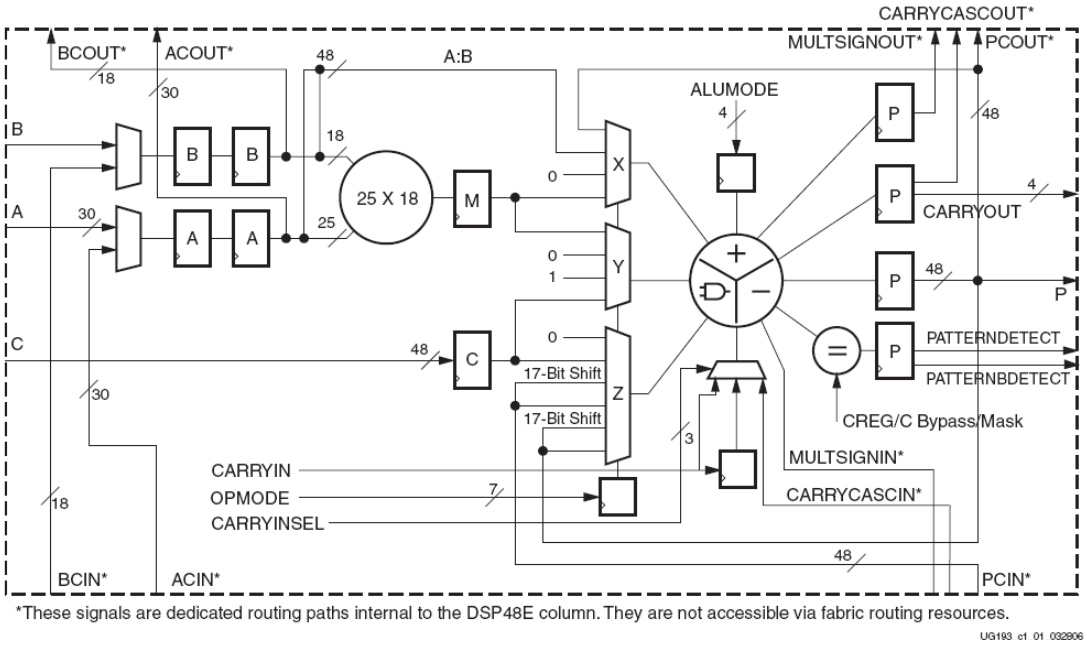


Figure 2.4 DSP Slice in Virtex-5 FPGAs [10]

2.3 Carry Look Ahead (CLA) Adders

Carry look ahead (CLA) adders are widely used in most applications where high speed addition is performed. The basic structure of a CLA adder is summarized in Figure 2.5. Each adder cell receives a pair of inputs (A_i and B_i) and a carry-in (C_i) to generate sum (S_i), propagate (P_i), and generate (G_i) signals. The P_i and G_i signals along with the carry-in signals produce

carry-out signals in the look ahead carry unit (LCU) for the subsequent adders. The equations shown in Figure 2.5 summarize the logic functions of the CLA adder. The propagate signal can be generated by using an OR gate as summarized in the adder equations using the "POR" implementation in Figure 2.5. Another way of generating the propagate signal is by using the XOR gate used for the sum as summarized in the adder equations using the "PXOR" implementation in Figure 2.5.

Larger CLA adders can be constructed by connecting the carry-out of one 4-bit LCU unit to the carry-in of the next 4-bit CLA unit [19]. This type of CLA adder is called the ripple CLA adder. Another approach feeds the propagate (PG) and generate (GG) signals produced in the LCU to a second stage LCU to construct a 16-bit CLA adders. Larger CLA adders of 48-bits like the adder used in DSP cores in Virtex-4 and Virtex-5 FPGAs can be constructed either by rippling the carry outputs of the second stage LCU or by adding an additional stage of LCUs. Additional LCUs reduce delay at the expense of additional area overhead.

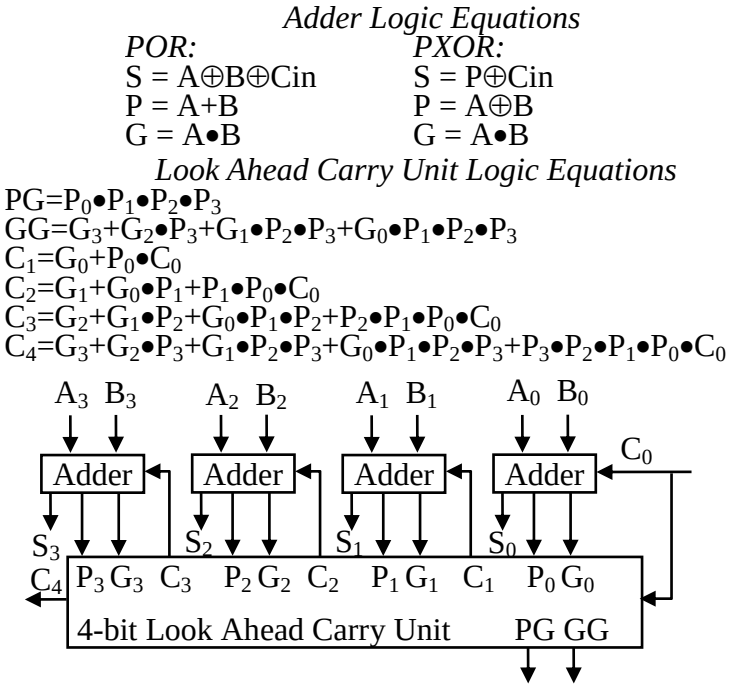


Figure 2.5 Basic Structure of a 4-bit CLA

2.4 Booth Multipliers

An $m \times n$ array multiplier performs multiplication be generating n partial products for each of the m-bits of the multiplicand. These partial products are summed using an array of adders to generate the final result. Booth multipliers reduce the number of partial products to be summed by "recoding", meaning grouping together some bits of either one of the operands, thereby speeding up the multiplication process [16].

The architecture of the multiplier is divided into three groups of cells. They are named as the "recoding cells (r cells)", the "partial product cells (pp cells)" that calculate the partial products and the "adder cells" that sum the partial products. One of the two operands of the multiplier is recoded [16]. If the Booth multiplier has a 2-bit recoding then the recoded operand is divided into groups where each group has 2 bits. If X is the recoded operand, the bits in the group would be X_{2j}, X_{2j+1} where j varies from 0 to $N_x/2$ and N_x is the number of bits in the operand X [16]. These two bits and the most significant bit (MSB) of the previous group, X_{2j-1}, are fed to the recoding cells. The recoding cells produce signals which determine the functions that must be performed on the second operand in order to generate the partial products that will be calculated by the partial product cells. The partial products are then summed by the adder cells [16].

Wallace-tree multipliers with "Booth encoding" speed up the multiplication process further [17]. The Booth encoding feature halves the number of partial products, and Wallace-tree addition with the output CLA adder to sum the final stage partial products result in the fastest addition [17]. This Wallace/Booth multiplier in [17] is divided into three parts: the Booth encoder for generating the partial products, the Wallace-tree unit that adds the partial products and generates a sum and carry vector and a final stage CLA adder that adds the sum and carry

vectors to generate the final result. The Wallace-tree unit consists of half adder and full adder units [17].

2.5 Prior Work Done in Testing CLA Adders

For a 4-bit CLA adder implementation that uses an OR gate to produce the propagate signal, a minimum set of ten vectors was proposed in [19] to detect all single stuck-at faults. For larger ripple CLA adders, a set of eleven vectors was proposed in [19] to detect all single stuck-at faults. For the ripple CLA adder that uses an XOR gate for calculating the propagate signal, a minimum set of twelve vectors was proposed in [19] to detect all single stuck-at faults. But these sets of vectors apply only to ripple CLA adder implementations [19].

Another test algorithm that tests any *n*-bit CLA adder implementation is proposed in [15]. The CLA adder in [15] is divided into three units: the top level structure of the n-bit CLA, which is referred to as the "M_{CLA}" unit, the "M_{PGX}" unit that calculates the propagate and generate signals (in this test algorithm, the propagate signal is calculated using an OR gate) and the "M_{CLG}" unit that calculates all the carry signals [15]. The sum is calculated using the XOR operation. The faults in the M_{CLG} unit are difficult to test and hence tests are generated for a set of faults that cover all single stuck-at faults on the input paths of the M_{CLG} unit [15]. The known tests for the M_{CLG} unit are traced via the M_{PGX} unit to the primary input paths of the M_{CLA} unit to obtain a test sequence to detect all the single stuck-at faults in the CLA. Table 2.4 shows the test sequence for a 2-bit CLA. This test sequence can be extended to a 4-bit CLA as shown in Table 2.5 [15].

The input patterns for an *n*-bit CLA can be generated using a twisted ring counter approach as shown in Figure 2.6. This TPG can be implemented using *n* XOR gates, *n* XNOR

gates and an (n+1) bit shift register with an inverter to form a twisted ring counter [15]. Reference [15] claims 100% single stuck-at gate level fault coverage.

Table 2.4 Test Sequence for a 2-bit CLA [15]

Test #	A1B1	A0B0	C0
1	10	10	1
2	10	00	1
3	00	11	1
4	01	01	0
5	01	11	0
6	11	00	0

Table 2.5 Test Sequence for a 4-bit CLA [15]

Test #	A3B3	A2B2	A1B1	A0B0	C0
1	10	10	10	10	1
2	10	10	10	00	1
3	10	10	00	11	1
4	10	00	11	11	1
5	00	11	11	11	1
6	01	01	01	01	0
7	01	01	01	11	0
8	01	01	11	00	0
9	01	11	00	00	0
10	11	00	00	00	0

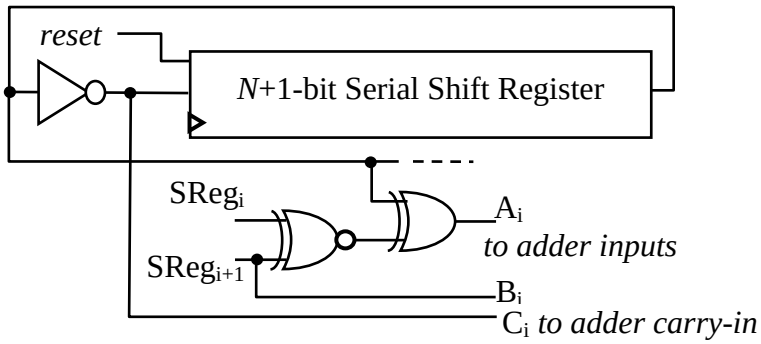


Figure 2.6 Adder Test Algorithm using Twisted Ring Counter [15]

2.6 Prior Work Done in Testing Booth and Booth/Wallace Multipliers

A multiplier test algorithm for Booth multipliers is proposed in [16] and claims high fault coverage of over 99%. The number of test vectors is 256 and is independent of the size of the

multiplier. The BIST TPG can easily be implemented using an 8-bit counter [16]. This test algorithm claims to pseudo-exhaustively test all the multiplier cells described in Section 2.4. Figure 2.7 shows the BIST architecture used by the test algorithm [16]. The test patterns are generated by an 8-bit counter [16]. The 8-bit counter applies all 256 patterns to the inputs of the multiplier [16]. This algorithm will be referred to in this thesis as the 4×4 test algorithm. Here the four MSB bits of the counter are applied to one input of the multiplier and the four LSB bits of the counter are applied to the other input of the multiplier. Starting from the LSB of the multiplier operands, the two sets of counter bits are replicated and repeated for each group of four bits of the multiplier operands [16]. For an 8-bit multiplier the 4×4 test algorithm will apply the test patterns illustrated in Table 2.6, where A[7:0] and B[7:0] are the inputs of the two ports of the multiplier and C[7:0] indicate the outputs of the 8-bit counter.

Table 2.6 Test Patterns for an 8-bit Multiplier Using 4×4 Test Algorithm

Multiplier Inputs	A7	A6	A5	A4	A3	A2	A1	A0	B7	B6	B5	B4	B3	B2	B1	B0
Counter Outputs	C7	C6	C5	C4	C7	C6	C5	C4	C3	C2	C1	C0	C3	C2	C1	C0

Another multiplier test algorithm was proposed in [17]. Figure 2.8 shows the BIST architecture for the test algorithm [17]. This test algorithm targets Wallace-tree multipliers with Booth encoding. The CLA adder used to sum the final partial products in this algorithm is a multi-stage LCU CLA adder [17].

Like the 4×4 test algorithm, the test algorithm in [17] also does not modify the structure of the multiplier and an 8-bit counter is used to generate the test patterns for any size multiplier. X and Y are the input operands, where the X operand has Booth encoding. In this algorithm, for the multiplier input port which has the Booth encoding, the five MSB bits of the counter are replicated and repeatedly applied to each group of five bits starting from the LSB of the

multiplier operand with Booth encoding [17]. For the other input of the multiplier, the remaining three LSB bits of the counter are replicated and applied to each group of three bits starting from the LSB of the other multiplier operand. This algorithm is referred to in this thesis as the 5×3 test algorithm. In the proposed BIST approach, the output response is compacted by an accumulator and compared with the fault-free signature to detect faults [17]. For an 8-bit multiplier the 4×4 test algorithm will apply the test patterns illustrated in Table 2.7 where A[7:0] and B[7:0] are the inputs of the two ports of the multiplier and port A has the booth encoding. C[7:0] indicate the outputs of the 8-bit counter.

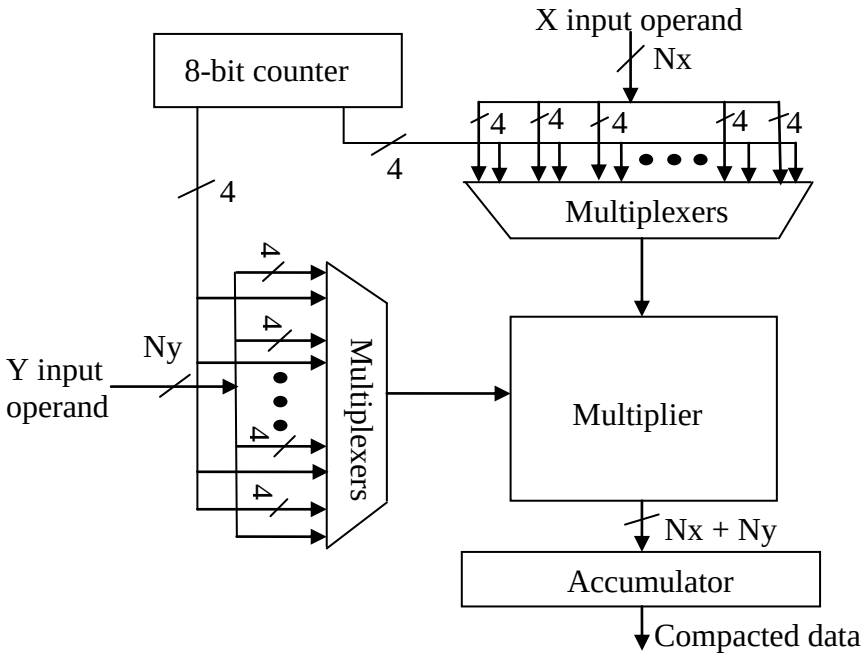


Figure 2.7 4×4 Multiplier Test Algorithm [16]

Table 2.7 Test Patterns for an 8-bit Multiplier Using 5×3 Test Algorithm

Multiplier Inputs	A7	A6	A5	A4	A3	A2	A1	A0	B7	B6	B5	B4	B3	B2	B1	B0
Counter Outputs	C5	C4	C3	C7	C6	C5	C4	C3	C1	C0	C2	C1	C0	C2	C1	C0

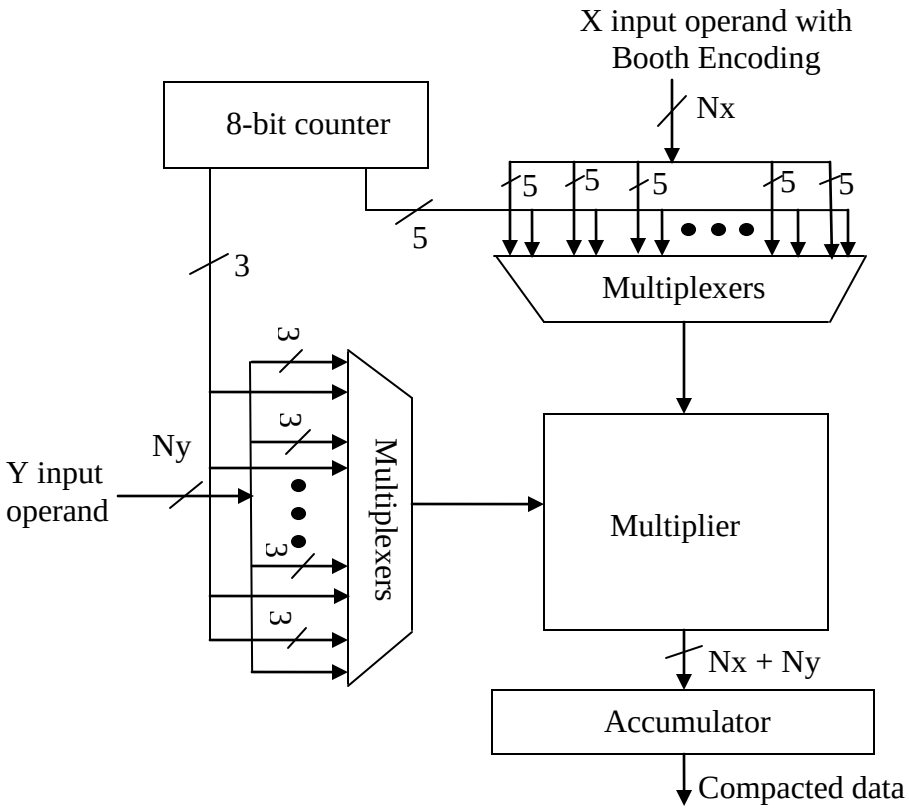


Figure 2.8 5×3 Multiplier Test Algorithm [17]

The authors in [20] mention that the DSPs in Virtex-4 devices can be tested by applying pseudo-random patterns, generated by linear feedback shift registers (LFSRs) to the input ports of the DSP slice, but the authors do not provide specific test algorithms or test sequences for testing the logic in the DSP cores. Reference [20] also does not mention the fault coverage obtained. Furthermore, to apply an exhaustive set of pseudo-random patterns would require a 84-bit LFSR and $2^{84} - 1$ clock cycles of test application time.

A BIST approach for the 18×18-bit multipliers embedded in Virtex-II Pro FPGAs was proposed and implemented in [21]. This BIST approach was the first BIST approach implemented for multipliers in FPGAs. The 4×4 test algorithm proposed in [16] was used to test the multipliers embedded in Virtex-II Pro FPGAs. A 10-bit counter was used for the test pattern generator, where the eight LSB bits of the counter were used to apply the 4×4 test algorithm to

the two 18-bit inputs of the multiplier and the two MSB bits of the counter are used to test the clock enable and reset control inputs to the multiplier in registered modes of operation [21]. A minimum set of three BIST configurations were developed to test the multipliers in all modes of operation. The three BIST configurations include BIST for one "combinational mode" and two "registered modes" of the multiplier. The BIST configuration for the "combinational mode" is used only to test the logic in the multiplier without any registers [21]. The two BIST configurations for the registered modes are used to test the programmable active levels of the clock enable and reset control inputs of the registers and the active edge of the clock to the registers. The BIST configurations were developed in VHDL models and require a complete download of each of the three BIST configurations [21].

The comparison based ORA shown in Figure 2.9 compares the outputs of the multiplier blocks under test (BUTs) and produces a pass/fail result for each BIST configuration [21]. This ORA design is easy to implement and can be implemented in two LUTs of a CLB slice since the contents of each ORA have to be shifted out to obtain the pass/fail result of each ORA since the ORAs were connected to form a scan chain to shift out the BIST results [21].

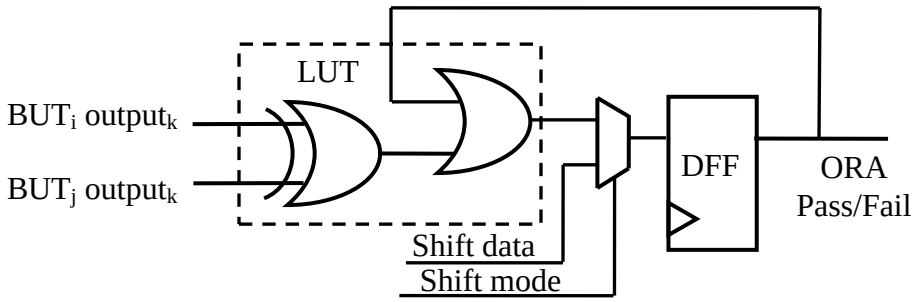


Figure 2.9 ORA Design for Multiplier BIST in Virtex- II Pro FPGAs [21]

2.7 Restatement of Thesis:

Although no prior work has been done on testing DSP cores in FPGAs, the adder test algorithm proposed in [15] and the multiplier test algorithm proposed in [17] can be modified for

better fault coverage and can be applied to completely test the adder and the multiplier in DSP cores of Virtex-4 and Virtex-5 FPGAs.

The TPG for the adder test algorithm proposed in [15] can be easily implemented in the CLBs of Virtex-4 and Virtex-5 FPGAs. Although the number of test vectors increases with the size of the adder, the adder in DSP cores of Virtex-4 and Virtex-5 FPGAs can be completely tested with a reasonably small set of test vectors.

The TPG for the multiplier test algorithm proposed in [17] can also be easily implemented and applied to multipliers in DSP cores of Virtex-4 and Virtex-5 FPGAs. The TPG can be implemented in the CLBs of the FPGAs. The test vectors for the multiplier test algorithm are a small set of finite test vectors. These 256 test vectors can be applied to multipliers of any size. Besides the adder and the multiplier, the rest of the DSP logic must also be tested. This thesis seeks to develop a minimum set of BIST configurations to completely test the DSPs in Virtex-4 and Virtex-5 devices.

Chapter 3

BIST for DSP Cores in Virtex-4 FPGAS

This chapter begins by proposing improvements to the previously proposed multiplier and adder test algorithms for higher fault coverage and describes the development of BIST for DSP cores in Virtex-4 FPGAs through the application of the improved multiplier and adder test algorithms to test the logic in these DSP cores. The BIST architecture along with the BIST configurations and test sequences for the DSP cores are discussed. The chapter also discusses the retrieval of BIST results and explains how the maximum clock frequency of the BIST configurations can be improved. The chapter concludes by summarizing the experimental BIST results and the fault coverage obtained on actual Virtex-4 FPGAs.

3.1 BIST Approach for DSPs in Virtex-4 FPGAs

The DSP cores in Virtex-4 FPGAs mainly consist of the adder and the multiplier units. Besides the adder and the multiplier the DSP cores include multiplexers and flip-flops used as pipeline registers. Since the multiplexers and the flip-flops can be easily tested, the challenge lies in testing the adder and the multiplier units in the DSP cores. The data sheets for the DSP cores incorporated in Virtex-4 FPGAs do not describe the architecture of the adder and the multiplier explicitly. However, one of the Spartan-3 application notes mentions that the architecture of the multiplier is based on a modified Booth architecture [22]. From the data sheets [9] it is understood that sequential logic is not used since there is no specification of clock cycle latency. Of the various combinational logic multipliers such as array, Booth, modified

Booth, Wallace-tree, and modified Booth/Wallace-tree multipliers, the modified Booth/Wallace-tree multiplier seems to be the most likely option because of its higher performance.

From the data sheets it is clear that the adder that is used to sum the final partial products of the multiplication is separated from the multiplier. Of the various combinational logic adders such as ripple carry, carry select, carry skip, carry save and carry look ahead (CLA) adders, the CLA adder seems to be the most likely option because of its higher performance [23] and also because CLA adders are typically used to sum the final partial products in modified Booth/Wallace-tree multipliers [17].

3.1.1 Adder Test

The adder test algorithm described in [15] can be used to test the adder in the DSP cores. However, fault simulation for the adder test algorithm in [15] revealed that two test patterns that were required to achieve 100% fault coverage were missing. Modifying the BIST architecture in [15] by replacing the inverter with a D flip-flop and using the *Qbar* output to drive the input of the shift register as illustrated in Figure 3.1 produces the two missing patterns. The modified architecture includes an N+1 bit shift register, N XOR gates, N XNOR gates and a D flip-flop, where N is the number of bits in the adder. This BIST architecture generates a total of $2\times(N+2)$ test vectors for completely testing the CLA adders in the DSP cores. Since the adder in the DSP cores is 48-bits wide, a 50-bit shift register (49-bit shift register plus the D flip-flop) is used to generate 100 test vectors that completely test the adder, as verified through fault simulation. The test patterns generated by the modified architecture are also illustrated in Figure 3.1 for a 4-bit adder and the generated missing test patterns are denoted as “new”. Table 3.1 gives a comparison of fault coverage achieved for the adder test algorithms described in Section 2.5 of Chapter 2. From Table 3.1 it is observed that the adder test vectors described in [19] effectively test only

ripple CLA adders but the adder test algorithm described in [15] effectively tests all implementations of the CLA adder but failed to give 100% fault coverage because of the missing vectors. The "Modified BIST" indicates the modification made to the adder test algorithm that generated the missing test vectors described in this section.

Table 3.1 Stuck-at Fault Simulation Results for 48-bit Adders

Adder Implementation	Gate Delays	Number of Faults	Test Algorithm Vector Set		
			vector set [19]	BIST [15]	Modified BIST
Ripple Carry Adder	96	1296	100%	99.9%	100%
Ripple CLA	28	1392	100%	99.9%	100%
Ripple LCU	12	1542	95.7%	99.9%	100%
Multi-stage LCU	10	1506	95.9%	99.9%	100%

The adder/subtractor equation P=Z±(X+Y+Cin) [9] indicates that the adder in the DSP slice is a two-stage adder as shown in Figure 3.2. The C-port provides the only 48-bit access to the adder. The accumulator register, P, is the only other 48-bit access to the adder. Hence two clock cycles are required to apply a single test vector to the adder. During the first clock cycle a part of the test vector (48-bits of the 97-bit test vector) is loaded into the accumulator register from the C-port through the Y or the Z multiplexers while applying 0s to the other two multiplexer ports, the CIN and the SUBTRACT signals. During the second clock cycle, the 48-bit test vector in the accumulator register is applied to one of the ports of the adder through the X or the Z multiplexers. Also during the second clock cycle the remaining 48-bits of the test vector are applied to the other port of the adder through the Z or the Y multiplexers (based on the stage of the adder being tested) and the test vector bit for the CIN or SUBTRACT signals (based on the stage of the adder that is being tested) is applied. For cases during which the overall test vector applies a logic '1' to the SUBTRACT signal, the first part of the vector that is loaded to the accumulator register is inverted so that when the SUBTRACT signal inverts this part of the

vector while its being applied to the adder port during the second clock cycle, the correct set of vectors is still applied to the second stage adder. The two clock cycle test vector application also provides complete testing of the accumulator register. Each stage of the adder is tested independently and completely in a total of 200 clock cycles.

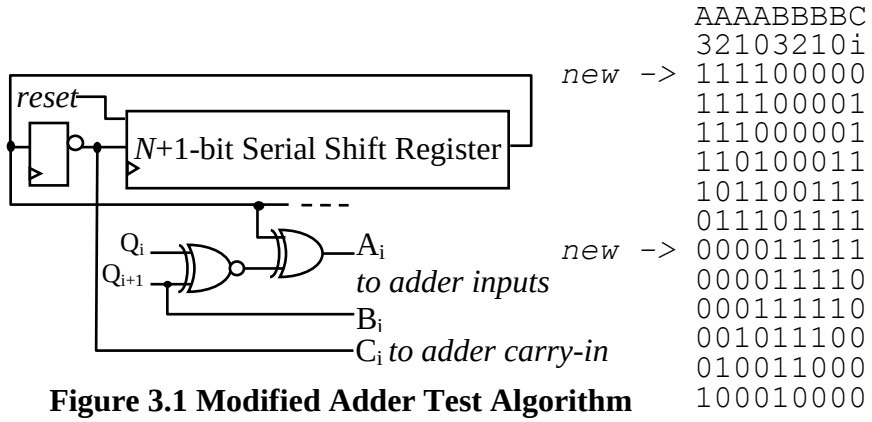


Figure 3.1 Modified Adder Test Algorithm

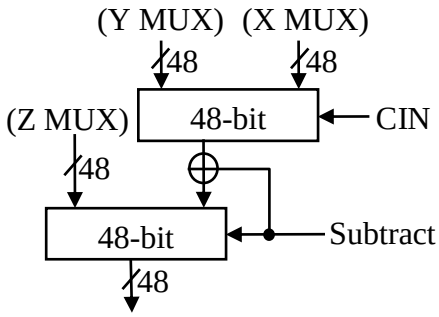


Figure 3.2 2-stage CLA adder

3.1.2 Multiplier Test

Fault simulation was performed on gate level models of various 8×8 bit multipliers. Based on the fault simulation results it is determined that applying both the 5×3 and the 3×5 test algorithms is the most effective way of testing the multiplier cores in Virtex-4 DSPs. Hence the multiplier is tested in two sessions of 256 clock cycles each. During the first session, the five MSBs of the 8-bit counter are applied to port A of the multiplier and the three LSBs of the 8-bit counter are applied to port B of the multiplier. During the second test session, the five MSBs are

applied to port B of the multiplier and the three LSBs are applied port A of the multiplier. Figure 3.3 summarizes the multiplier BIST approach.

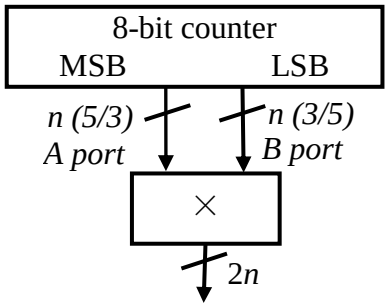


Figure 3.3 Multiplier BIST approach

3.2 BIST Architecture

Figure 3.4 illustrates the DSP BIST architecture for any given Virtex-4 FPGA. Two TPGs drive alternate rows of DSP tiles that have the same configuration. The two TPGs generate identical test patterns to test all the DSP tiles. The control register that controls the TPG is four bits wide. Of the four bits the two LSBs (called MODE1 and MODE0) control the test algorithm generated by the TPG. The second MSB (called INVCS) controls the active levels of control signals such as the OPMODE bits and the active level of the carryin input to the adder unit. The MSB provides a global reset to the TPG. The control register is implemented in a CLB and the values for the control register are shifted in through Boundary Scan interface while shifting the LSB first. The control register values for resetting the TPG and for the various test modes are summarized in Table 3.2. The 'X' in Table 3.2 indicates a don't care bit.

Table 3.2 Control Register Values for TPG control

Control Register Values <3:0> RESET	INVCS	MODE1	MODE0	Operation
1	X	X	X	Resets TPG
0	1	X	X	Inverts active level of control signals
0	X	0	0	Sets the multiplier test algorithm
0	X	0	1	Sets the adder test algorithm
0	X	1	0	Sets the cascade test algorithm

Values to the control register can also be given through a system pins interface when the Boundary Scan interface is not used. For the system pins interface, the clock and the control inputs to the TPG, such as the TPG reset, the INVCS control and the MODE1 and MODE0 control signals, are input pins to the device.

Multiple TPGs are used so that faults in any of the TPGs can not escape detection. Each TPG drives both the slices in a tile for the individual control of both slices in the tile during cascade modes of operation. The DSP slices are configured in cascade mode of operation in pairs instead of cascading all the DSP slices in a column, so that the maximum BIST clock frequency is not slowed down. This approach of cascading DSP slices in pairs also ensures that all the DSP slices do not fail the test due to the unconnected cascade inputs on the bottom-most DSP slice in a column of DSPs and circular comparison can still be used effectively to analyze the outputs of the DSPs, disagreeing with the authors' claim in [20]. The bottom slice in a DSP tile is denoted by s0 and the top slice in a DSP tile is denoted by s1. Each DSP slice is monitored by two sets of ORAs and compared with the outputs of two like DSP slices. Each set of ORAs monitors two similar DSP slices, implying a set of ORAs that monitor slice 0 in a DSP tile also monitor slice 0 in the DSP tile below. The two bottom-most sets of ORAs (one each for slice 0 and slice 1) in a column of ORAs monitor the top-most and the bottom-most DSP tiles in the column of DSP tiles, forming two circular comparison chains where one chain monitors slice 0 in all the DSP tiles and the other chain monitors slice 1 in all the DSP tiles.

The set of ORAs that monitor slice 0 of the bottom-most DSP tile have clock enables so that these ORAs can be disabled at specific times during the cascade mode BIST sequence to avoid ORA failure indications due to unconnected cascade inputs on slice 0 of the bottom-most DSP tile. The architecture of the ORA is illustrated in Figure 3.5. Each ORA comprises a look-

up table and a flip flop. The ORAs are synthesized into CLBs where eight ORAs fit into a single CLB. The dedicated carry logic in the CLBs, as illustrated in Figure 3.5, can be used to create an iterative OR chain of ORAs where the Test Data In (TDI) line is connected to the carry-in of the first CLB in the column of ORAs and the carry-out of the last CLB in the last column of ORAs [24], which is also the response of the last ORA in the chip array, is connected to the Test Data Out (TDO) line. This provides a single bit pass/fail result for the entire test. The TDO line goes to logic '1' when any one ORA in the iterative OR chain detects a mismatch due to faults. This reduces the total test time for the fault-free tests since the ORA contents can be obtained via partial configuration memory read-back for only those tests that fail. Each DSP slice is observed by 48 ORAs in two rows and three columns of CLBs. Figure 3.6 illustrates the mapping of the individual output bits from a single DSP slice to the ORAs. Figure 3.6 helps in determining the faulty DSPs in a column of DSPs and the individual faulty DSP outputs when partial configuration memory read-back is done. All the DSPs are tested concurrently so the length of the test sequence is independent of the size of the chip array.

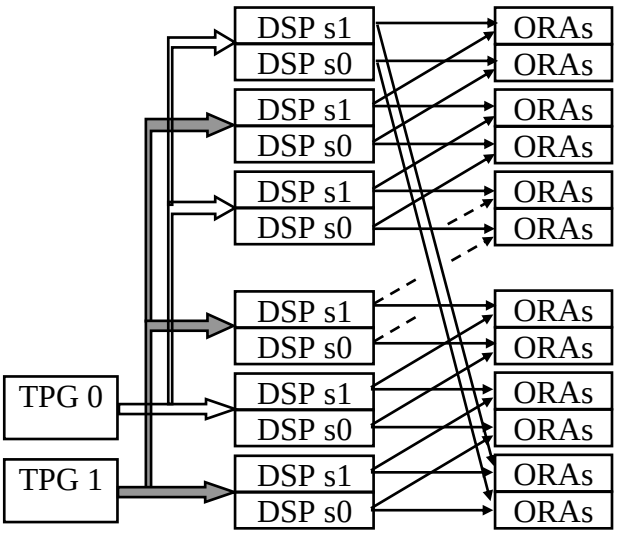


Figure 3.4 DSP BIST Architecture

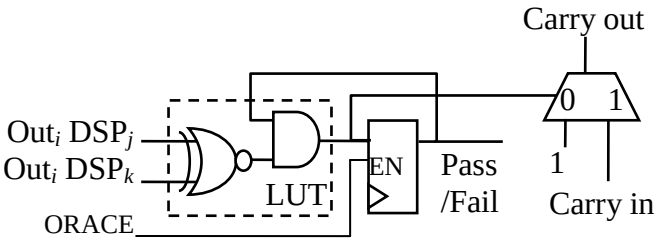


Figure 3.5 ORA Architecture

Figure 3.7 illustrates the general architecture of the TPG. The TPG for DSP BIST comprises a 10-bit counter where the two MSB bits are used for the individual control of the four 256 clock cycle test groups during the 1024 clock cycle test sequence, a 50-bit shift register for the adder test, a finite state machine (FSM) for control of OPMODE control signals and two 9-bit linear feedback shift registers (LFSRs) for generating weighted pseudo random control signals. The eight least significant bits of the counter are used to apply test patterns to the multiplier. The TPG is modeled in VHDL in 266 lines of codes and is synthesized into 44 CLBs.

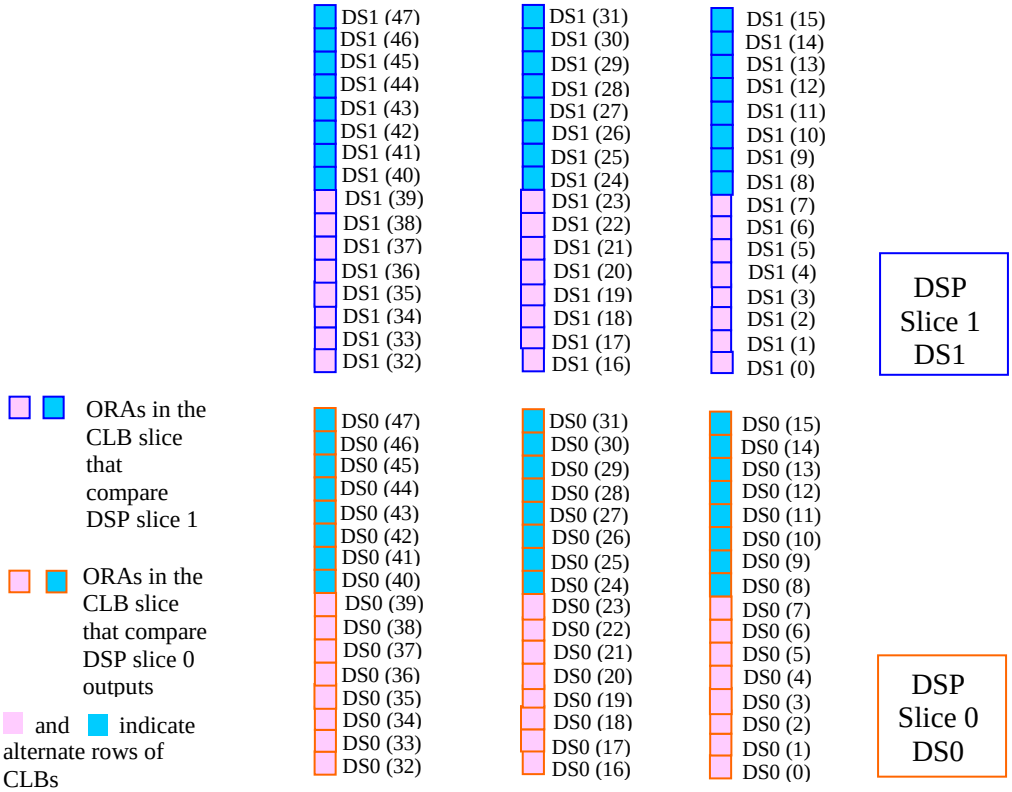


Figure 3.6 ORA map for a DSP tile

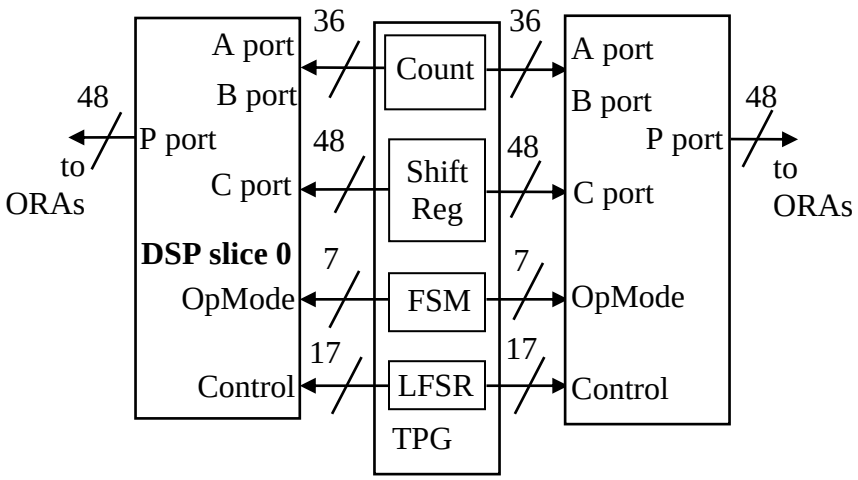


Figure 3.7 TPG Architecture

3.3 BIST Configurations and Test Sequences

The DSP cores in Virtex-4 devices are tested using three independent BIST sequences which correspond to each of the three test modes of operation: the multiplier, adder and cascade modes of operation. Table 3.3 summarizes the test sequences. Each BIST sequence is 1024 clock cycles long and divided into four groups of 256 clock cycles. During each group of 256 clock cycles, specific I/O paths through the multiplier, adder and cascade modes of operation are tested.

The 5×3 multiplier test algorithm is applied to the multiplier in slice 1 while the 3×5 multiplier test algorithm is applied to the multiplier in slice 0 during the first group of the multiplier test sequence. During the second group of the multiplier test sequence the 5×3 multiplier test algorithm is applied to the multiplier in slice 0 and the 3×5 multiplier test algorithm is applied to the multiplier in slice 1. The 5×3 multiplier test algorithm is applied by replicating the five MSBs of the vector generated by the 8-bit counter and applying the replicated bits to the 18-bit A port while the three LSBs of the vector generated by the 8-bit counter are replicated and applied to the 18-bit B port. During the 3×5 multiplier test algorithm the three LSBs of the vector generated by the 8-bit counter are replicated and applied to the 18-bit A port

while the five MSBs of the vector generated by the 8-bit counter are replicated and applied to the 18-bit B port. The application of different multiplier test algorithms to slice 0 and slice1 ensures that the A and B ports in both the slices receive different test patterns on every clock cycle so that the single stuck-at faults on the multiplexers that select between direct and cascade paths on port B of the DSP slice can be tested. During the third group of 256 clock cycles of the multiplier test sequence, the multiply and add function is tested. The multiplier is not tested in the fourth group of the multiplier test sequence. This group only tests for the A port concatenated with the B port (denoted as A:B in Table 3.3) bypass of the multiplier.

Each stage of the two stage adder is tested separately during the first two groups of 256 clock cycles in the adder test sequence. The first stage adder is tested during the first group of 256 clock cycles in the adder test sequence and the second stage adder is tested during the second and third groups of 256 clock cycles in the adder test sequence. The P output of the DSP slice that is left shifted by 17 bits can be fed back to the adder through the Z multiplexer (denoted as Z(ShiftP) in Table 3.3). This path to the adder is tested during the fourth group of 256 clock cycles in the adder test sequence.

Table 3.3 BIST Sequences

Test	Multiply	Adder	Cascade
First 256 ccs	P = A×B	P = Z(C) P = X(P)+Y(C)	P1 = A:B+Z(PC) P0 = Z(C)
Second 256 ccs	P = A×B	P = Y(C) P = Y(C)+Z(P)	P1 = A:B+Z(ShiftPC) P0 = Z(C)
Third 256 ccs	P = A×B+C	P = Z(C) P = Y(C)+Z(P)	P1 = Z(C) P0 = A:B+Z(PC)
Fourth 256 ccs	P = A:B+C	P = Y(C) P = Y(C)+Z(ShiftP)	P1 = Z(C) P0 = A:B+Z(ShiftPC)

During the third and fourth groups of 256 clock cycles in the adder and the multiplier test sequences, weighted pseudorandom patterns generated by linear feedback shift registers (LFSRs) are applied to test the various clock enables, resets and carry-in sources in the DSP. Weighted

pseudorandom patterns are used so that the pipeline registers of the DSP are reset less often since frequent resets of the pipeline registers can cause the fault detection data to be lost before it reaches the output of the DSP. Figure 3.8 illustrates the architecture of one of the 9-bit LFSRs, LFSRA. The second LFSR, LFSRB uses the reciprocal polynomial of LFSRA. Table 3.4 summarizes the weighted pseudorandom patterns in terms of their LFSR sources.

During the cascade mode test sequence, the two DSP slices are independently controlled to test the cascade multiplexers and the cascade interconnect between adjacent DSPs. Slice 0 and Slice 1 have different P equations, where P0 indicates slice 0 equation and P1 indicates slice 1 equation in Table 3.3. During the first group of 256 clock cycles in the cascade test sequence, slice 1 receives the P output of slice 0 as its input (denoted by Z(PC) in Table 3.3) and in the second group slice 1 receives the shifted P output of slice 0 as its input (denoted by Z(ShiftPC) in Table 3.3). During the third group of 256 clock cycles in the cascade test sequence, slice 0 receives the P output of the previous slice 1 as its input (denoted by Z(PC) in Table 3.3) and in the fourth group slice 0 receives the shifted P output of the previous slice 1 as its input (denoted by Z(ShiftPC) in Table 3.3). ORA failures due to unconnected cascade inputs on the bottom-most DSP slice occur in the third and fourth groups of the cascade test sequence. Therefore, the ORAs that monitor the DSPs at the bottom of the array are disabled by the TPG during the third and fourth groups of 256 clock cycles in the cascade test sequence.

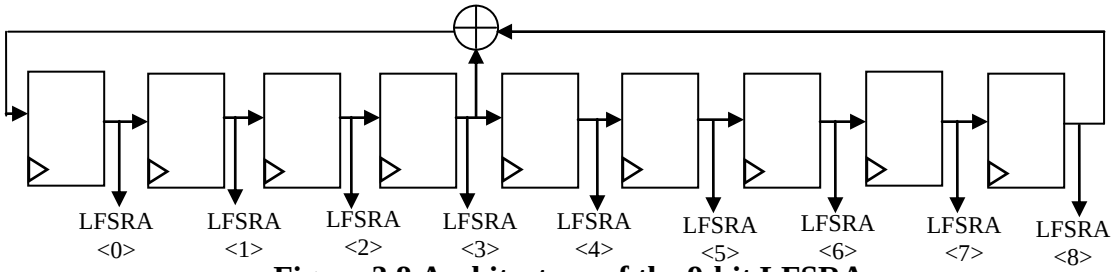


Figure 3.8 Architecture of the 9-bit LFSRA

Table 3.4 Weighted Pseudorandom Patterns

DSP Signal	Pattern
CEA (clock enable for Areg)	LFSRA<0>
CEB (clock enable for Breg)	LFSRA<1>
CEM (clock enable for M reg)	LFSRA<2>
CEP (clock enable for Preg)	LFSRA<3>
CECARRYIN (clock enable when carryin used for rounding applications)	LFSRA<4>
CECTRL (clock enable for CARRYINSEL, SUBTRACT and OPMODE registers)	LFSRB<5>
CECINSUB (clock enable when carryin is defined by the user)	LFSRA<6>
CARRYINSEL<1:0> (control register to select the carryin source)	LFSRA<8:7>
CARRYIN (user defined carryin)	LFSRB<0>
SUBTRACT (user defined subtract)	LFSRB<1>
CEC (clock enable for C reg)	LFSRB<2>
RSTA(reset for Areg) for slice 0	LFSRB<7> and LFSRB<5> and LFSRB<3>
RSTB(reset for Breg) for slice 0	LFSRB<1> and LFSRB<3> and LFSRB<5>
RSTM(reset for Mreg) for slice 0	LFSRB<6> and LFSRB<2> and LFSRB<0>
RSTP(reset for Preg) for slice 0	LFSRB<4> and LFSRB<0> and LFSRB<6>
RSTCARRYIN (reset for all sources of carryin) for slice 0	LFSRB<5> and LFSRB<0> and LFSRB<7>
RSTCTRL (reset for CARRYINSEL, SUBTRACT and OPMODE registers) for slice 0	LFSRB<6> and LFSRB<7> and LFSRB<8>
RSTA(reset for Areg) for slice 1	LFSRB<6> and LFSRB<4> and LFSRB<2>
RSTB(reset for Breg) for slice 1	LFSRB<0> and LFSRB<2> and LFSRB<4>
RSTM(reset for Mreg) for slice 1	LFSRB<5> and LFSRB<1> and LFSRB<8>
RSTP(reset for Preg) for slice 1	LFSRB<3> and LFSRB<8> and LFSRB<5>
RSTCARRYIN (reset for all sources of carryin) for slice 1	LFSRB<4> and LFSRB<8> and LFSRB<6>
RSTCTRL (reset for CARRYINSEL, SUBTRACT and OPMODE registers) for slice 1	LFSRB<5> and LFSRB<6> and LFSRB<7>
RSTC (reset for C reg)	LFSRB<0> and LSRB<1> and LFSRB<2>

The DSP cores are tested in five BIST configurations. During these five BIST configurations, the DSP configuration memory bits are tested in all functional modes. Table 3.5

summarizes the BIST configurations developed during the initial stages of BIST development (modifications to these BIST configurations will be explained in the later sections). In Table 3.5, column 1 indicates the BIST configuration download number, column 2 indicates the number of pipeline registers in the I/O paths of the DSP slice, column 3 indicates the active level of the DSP slice control signals, column 4 indicates whether the B port of the DSP slice is in cascade or direct mode of operation and column 5 indicates the test sequence number that is applied for each of the BIST configurations. The multiplier is tested during the first, second and fourth test sequences, the adder is tested during the third and fifth test sequences and the cascade modes of operation are tested during the sixth and seventh test sequences. Instead of connecting all the DSP slices in cascade mode at the same time alternate DSP slices are connected in cascade mode to avoid seeing failures due to the unconnected cascade input lines of the bottom-most DSP slice in the array. During the sixth test sequence, slice 0 is in cascade mode of operation and during the seventh test sequence, slice 1 is in cascade mode of operation.

BIST configurations #2 and #3 are run twice since the TPG control inputs need to be changed to run the multiplier and the adder test sequences during the same BIST configuration. A total of seven test sequences are applied in five downloads to the FPGA thereby reducing the number of downloads to the FPGA by two. The download time can be minimized using partial reconfigurations, through which only the configuration memory that contain DSP configuration memory bits are written instead writing the whole configuration memory. This can be done by maintaining constant placement of the TPGs, the ORAs and the DSPs and by keeping the routing constant between them. Table 3.6 illustrates the improvement in download time for the largest devices from each of the three families of Virtex-4 FPGAs, FX140, SX55 and LX200, (thereby representing the longest download times), when partial configuration is used and all the DSPs in

the devices are tested concurrently. The download time with partial reconfiguration in column 2 of Table 3.6 illustrates the download time for all five configurations where the first download is a compressed download and the remaining downloads are partial reconfigurations. The download time without partial reconfiguration in Table 3.6 illustrates the download time for all five configurations where all five downloads are compressed. The maximum clock frequency for download using the Boundary Scan interface is 50MHz.

Table 3.5 Initially Developed BIST Configurations

BIST Config #	Pipeline Registers	Signals Active Level	B Input Source		Test Modes Applied		
			Slice 0	*Slice 1*	*Mult*	*Add*	*Casc*
1	All Regs=0	High	Direct	Direct	#1		
2	All Regs=1	High	Direct	Direct	#2	#3	
3	A/Breg=2, Others=1	Low	Direct	Direct	#4	#5	
4	Preg=1, Others=0	High	Direct	Cascade			#6
5	Preg=1, Others=0	Low	Cascade	Direct			#7

Table 3.6 Improvement in Download Time using Partial Reconfiguration

Device	Download time with partial reconfiguration (sec)	Download time without partial reconfiguration (sec)
FX140	0.30128	1.47814
SX55	0.27915	1.32346
LX200	0.22468	1.11734

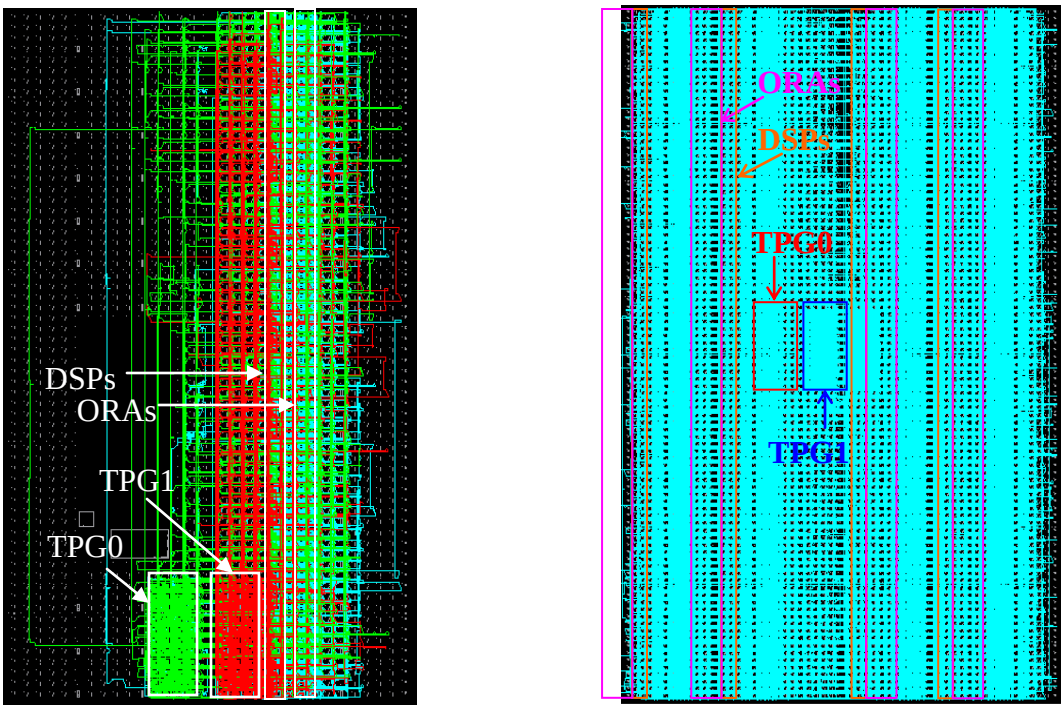
3.4 BIST Generation

The TPG model written in VHDL is synthesized for an FX12 device since the FX12 device has a large area in the chip array that is occupied by the Power PC and the TPG is carefully constrained in an area close to the DSPs that does not interfere with the location of the Power PC. The TPG location for all other devices is offset with respect to the location of the TPG in the FX12 device based on the number of rows and columns in individual devices. The synthesized TPG is in NCD format that can be viewed in FPGA Editor, a Xilinx tool that gives a

graphical representation of the device. The synthesized TPG in NCD format is then converted to XDL (Xilinx Design Language) format.

Three programs written in C (developed as part of this thesis work) generate the BIST configurations for any size or family of Virtex-4 devices. The *V4DSPBIST.exe* program calls another program *TPGXDLEXT.exe*. The latter program extracts the TPG, from the synthesized XDL and writes it to an output file. The *V4DSPBIST.exe* program reads that output file and places and instantiates the two TPGs. It also instantiates and interconnects the remaining BIST architecture, the DSPs under test and the ORAs, and generates a DSP BIST template in XDL format. The DSP BIST template in XDL format is converted to NCD format for routing the BIST architecture. The routed DSP BIST template is then converted back to XDL format to be used by the modification program, *V4DSPMOD.exe*. This modification program, written in C, is used to modify the routed DSP BIST template to generate the five BIST configurations in XDL format. These BIST configuration files are then converted back to NCD format to generate the download configuration bit files.

The NCD files of the DSP BIST template and the BIST configuration can be viewed in FPGA Editor. The routed DSP BIST templates for the FX12 and SX35 devices are illustrated in Figure 3.9a and Figure 3.9b, respectively. For DSP columns to the right of the center column, the ORAs are located in three consecutive columns of CLBs on the right side of DSP and for the DSP columns to the left of the center column the ORAs columns are positioned to the left of the DSP. To avoid the PowerPC modules in FX family devices larger than FX40, the ORAs are located to the right side of DSPs that are located left of the center column and for the DSP column to the right of the center column, the ORAs are located on the left side of the DSPs.



a)routed FX12 **b)routed SX35**

Figure 3.9 BIST Template as Seen in FPGA Editor

3.5 Detection of Faulty DSPs and Fault Coverage

To verify the fault detection capabilities of the DSP BIST, faults were injected into the configuration memory bits that control the DSPs in an FX12 device. Figure 3.10 illustrates individual fault coverage achieved for each of the seven BIST sequences (BIST sequences #2 and #3 are the same download and represent BIST configuration #2 in Table 3.5. Similarly, BIST sequences #4 and #5 are the same download and represent BIST configuration #3 in Table 3.5. This is because configuration downloads #2 and #3 in Table 3.5 are run twice as explained in Section 3.3. BIST sequences #6 and #7 represent BIST configurations #4 and #5 respectively in Table 3.5). Cumulative fault coverage of 97.4% is achieved. Of the 154 faults injected four faults were not detected but could be detected by adding more BIST configurations to achieve 100% fault coverage. However, these undetected faults are in non-functional modes of operation making the additional BIST configurations impractical.

The DSP BIST configurations were able to detect faulty DSPs in some of the engineering sample parts of SX35 and LX-60 devices. Of the five SX35 and nine LX60 engineering sample parts tested, DSP BIST detected up to five faulty DSPs in four SX35 engineering sample parts and one faulty DSP each in two LX60 engineering sample parts. The faulty DSP slices in the Virtex-4 SX35 and LX60 engineering sample parts are summarized in Table 3.7. The corresponding faulty DSP output bit positions observed by the ORAs are also shown in Table 3.7. Column 2 in the table describes the position of the faulty DSP slice as named by the BIST generation program. For example, *DSP_r90c46* implies the DSP slice in the 45th DSP row and 46th column of the chip array and *DSP_r52c46* implies the DSP slice in the 26th DSP row and 46th column of the chip array. Column 3 describes the test sequence number during which each of the faulty DSP slices described in Column 2 were detected. Column 4 gives the failing DSP output bits where P is the output port of the DSP slice. Each engineering sample part shown in Table 3.7 was tested three times for each of the seven test sequences. The failing DSP output bit positions differed during each of the three tests for every engineering sample part shown in Table 3.7 as illustrated for the first SX 35 device

Table 3.7 Faulty DSP Slices in Virtex-4 SX35 and LX60 Engineering Sample Parts

Device	**Test number**	**Slice Description**	**Failing DSP output bit positions**		
			1st test	**2nd test**	**3rd test**
SX35 part#1	**1**	DSP_r90c46	P0-P3	P0-P10	P0-P47
		DSP_r52c46	P0–P21	P0-P47	P0-P2
		DSP_r56c19	P0,P1	P0-P20	P0-P3
		DSP_r8c35	P0-P47	P0-P47	P0-P47
		DSP_r80c8	P0-P47	P0-P47	P0-P47
	2	DSP_r90c46	P0-P10	P0-P1	P0-P1
		DSP_r52c46	P0-P27	P0-P27	P0-P37
		DSP_r56c19	P16-P29 P32-P47	P0-P11 P16-P31	P0-P5 P16-P47
		DSP_r8c35	P0-P31 P42-P47	P0-P37	P0-P37

Device	Test Number	Slice Description	Failing DSP output bit positions		
			1st test	2nd test	3rd test
		DSP_r80c8	P0-P5 P16-P47	P0-P37	P0-P37
	3	DSP_r90c46	P0-P18	P0-P47	P0-P18
		DSP_r52c46	P0-P27	P0	P0-P29
		DSP_r56c19	P0-P7 P16-P35	P16-P29 P32-P47	P0-P3 P16-P47
		DSP_r8c35	P0-P35	P0-P35	P0-P35
		DSP_r80c8	P0-P35	Po-P35	P0-P3 P16-P47
SX35part#3	**1**	DSP_r72c19	P0-P19		
	2	DSP_r72c19	P0-P27		
Device	**Test number**	**Slice Description**	**Failing DSP output bit positions for all three iterations**		
SX35part#4	**1**	DSP_r64c8	P0-P47		
		DSP_r74c35	P0-P3		
		DSP_r92c19	P0-P2		
	2	DSP_r64c8	P0-P5, P16-P47		
		DSP_r74c35	P0, P1		
	3	DSP_r64c8	P0-P3, P16-P47		
		DSP_r74c35	P0-P16		
SX35part#5	**2**	DSP_r52c35	P0		
	3	DSP_r52c35	P0-P47		
LX60part#6	**1**	DSP_r46c15	P0-P40		
	2	DSP_r46c15	P0-P12		
	3	DSP_r46c15	P0-P18		
LX60part#8	**2**	DSP_r90c15	P0-P47		
	3	DSP_r90c15	P0-P47		

3.6 BIST Timing Analysis

To determine the maximum clock frequency of DSP BIST, timing analysis was done using the Xilinx timing analysis tool *TRCE.exe*, for all Virtex-4 FPGAs. Figure 3.10 illustrates the maximum BIST clock frequency (in MHz) for all five configurations for an SX35 device when the TPG is placed at the bottom of the array. From Figure 3.10 it is observed that the BIST clock frequency for configuration #1 is always low. Since configuration #1 has no pipeline registers, the timing tool cannot calculate the accurate BIST clock frequency for this

configuration since it assumes the possibility of a dynamic cascade of all DSPs in the device even though the DSPs are not cascaded during this BIST configuration. Configuration #5 has the next slowest BIST clock frequency because the DSPs are clocked on the falling edge of the clock while the TPGs and ORAs are clocked on the rising edge of the clock in this configuration. The cascade routing between the DSP slices in configuration #5 also decreases the maximum BIST clock frequency. To improve the overall clock frequency of BIST, the DSPs in configuration #5 are clocked on the rising edge of the clock since configuration #3 takes care of testing DSPs when clocked on the falling edge of the clock.

Figure 3.11 illustrates the BIST clock frequency (in MHz) for all five BIST configurations for an SX35 device when the DSPs are clocked on the falling edge of the clock only in configuration #3. So, now configuration #3 has the slowest BIST clock frequency. From the timing analysis results performed on all Virtex-4 devices for the slowest BIST configuration, configuration #3, it is observed that the position of the TPG in the array has a significant impact on the BIST clock frequency. Figure 3.12 illustrates the maximum clock frequency (in MHz) for the slowest BIST configuration (#3) for all Virtex-4 FPGAs with respect to the TPG position at the bottom of the array or at the middle of the array as shown in Figure 3.8a.

From Figure 3.12 it is seen that higher BIST clock frequency is achieved when the TPG is placed at the middle of the array when compared to the placement of the TPG at the bottom of the array. This is because the top-most and the bottom-most DSP slices are placed at an equal distance from the TPG when the TPG is placed at the middle of the array. This makes the routing distance between the top-most DSP slice and the TPG shorter compared to the longer routing distance when the TPG is placed at the bottom of the array. Therefore, the TPG is placed at the middle of the array for all devices except the FX12 and FX20 devices that have a PowerPC

module at the middle of the array. The maximum clock frequency for BIST is less than 50MHz for some of the larger Virtex-4 FPGAs, like LX100, LX160, LX200 and FX140. Sub-array testing can be done for these devices where each half of the array is tested separately.

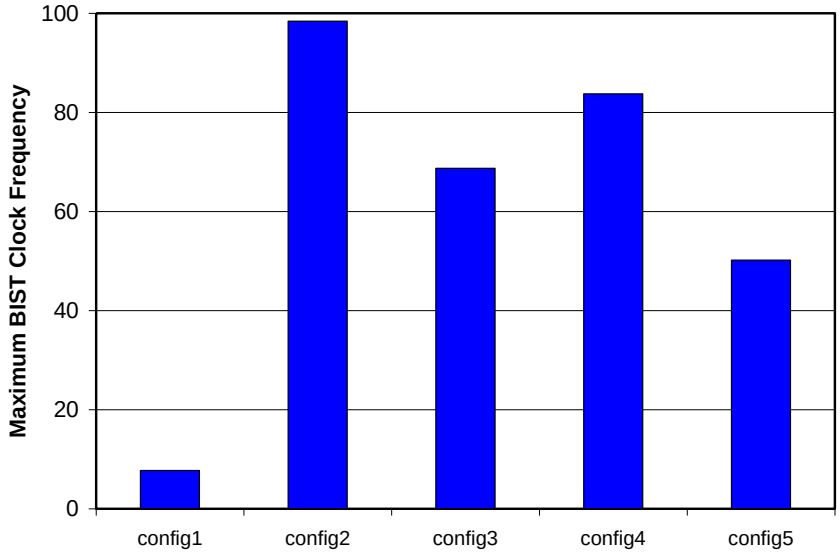


Figure 3.10 Maximum BIST Clock Frequency for an SX35 Device When DSPs in Configurations #3 and #5 are Clocked on Falling Edge of the Clock

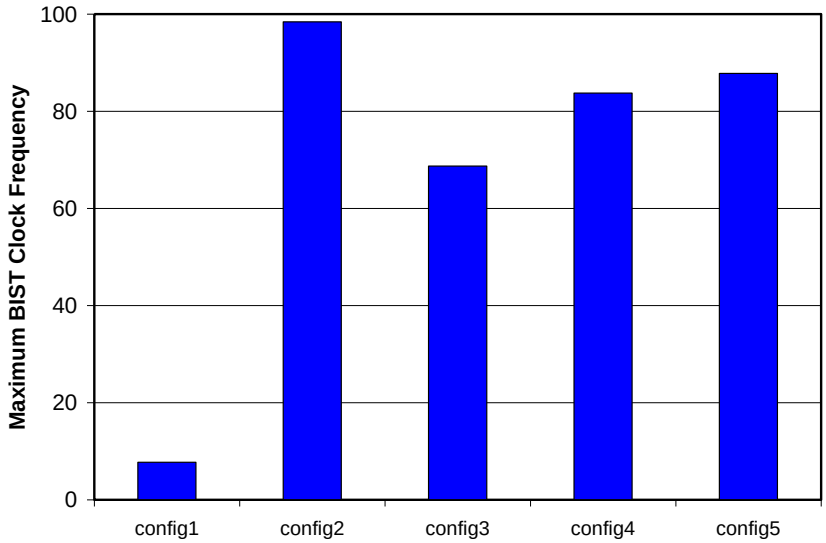


Figure 3.11 Maximum BIST Clock Frequency for an SX35 Device when DSPs in Configuration #3 are Clocked on Falling Edge of the Clock

Figure 3.13 illustrates the maximum clock frequency (in MHz) for the sub-arrays as a function of the TPG position in the array. In Figure 3.13, "Bottom BIST Bottom TPG" refers to BIST for the bottom half of the array when the TPG is placed at the bottom of the array, "Top

BIST Bottom TPG" refers to BIST for the top half of the array when the TPG is placed at the bottom of array, "Bottom BIST Middle TPG" refers to BIST for the bottom half of the array when the TPG is placed at the middle of the array, and "Top BIST Middle TPG" refers to BIST for the top half of the array when the TPG is placed at the middle of the array. From Figure 3.13 it is seen that the maximum BIST clock frequency for the top half of the array when the TPG is placed at the middle of the array is more than the maximum clock frequency for the bottom half of the array. This is because when the TPG located at the middle of the array, for the top half of the array, the TPG routes across and then up to the DSPs above whereas for the bottom half of the array, the TPG routes down to the bottom of the array to the DSPs and then routes across and up to the DSPs above.

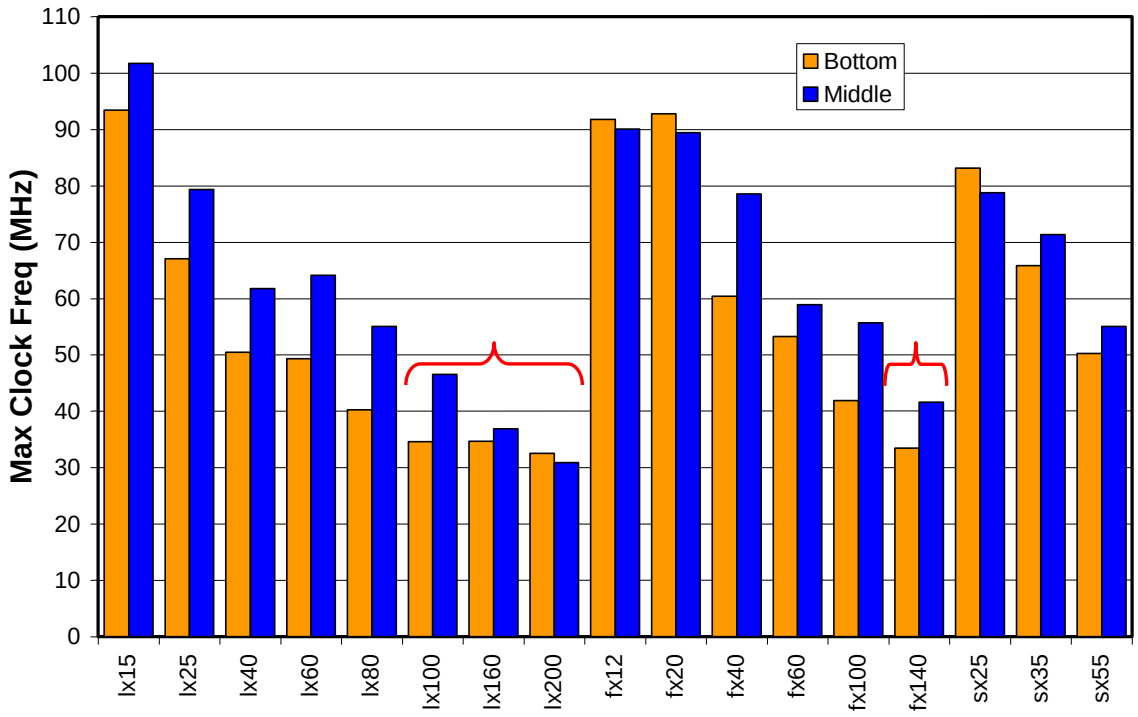


Figure 3.12 Maximum BIST Clock Frequency

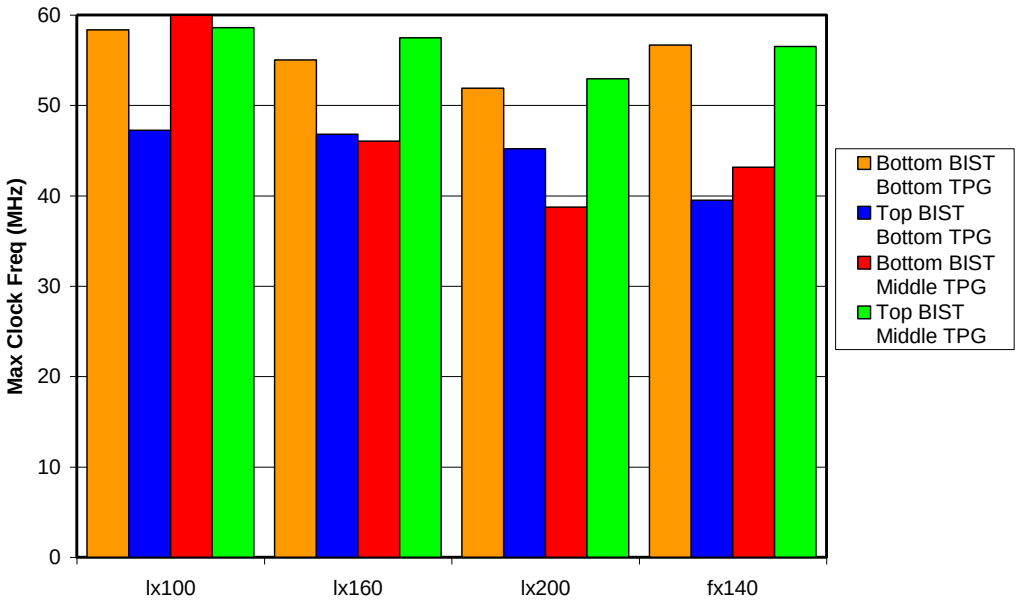
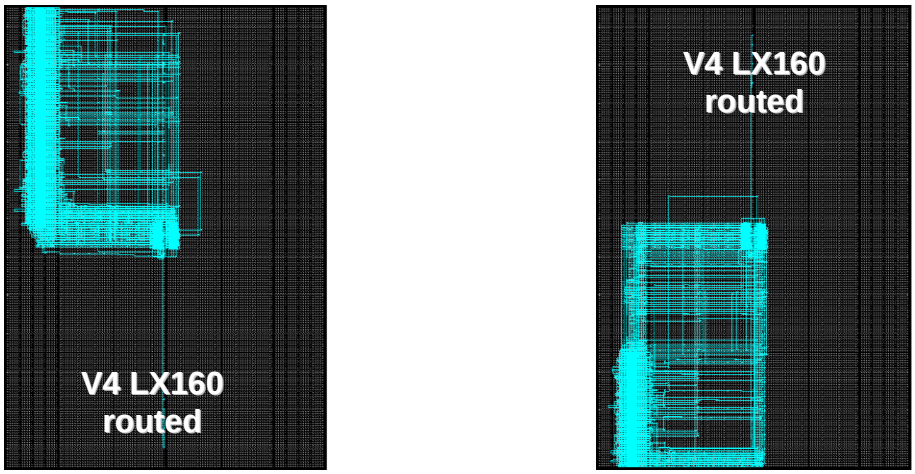


Figure 3.13 Maximum Clock Frequency for Sub-Arrays

Figure 3.14 illustrates the routing paths for the top and bottom halves of the array when the TPG is placed at the middle of the array. Hence, the routing path is longer for the bottom half of the array compared to the top half of the array which explains the slower clock frequency. Therefore, to make the clock frequency for the bottom half of the array as fast as the top half of the array, the TPG is placed at the bottom when testing the bottom half of the array as shown in Figure 3.15.



a) Routing for the Top Half of the Array **b) Routing for the Bottom Half of the Array**

Figure 3.14 Routing Paths for the Sub-Arrays with TPG at the Middle of the Array

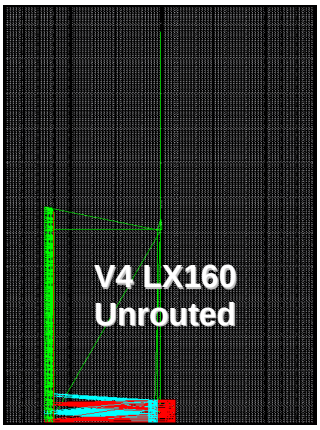


Figure 3.15 TPG Position for the Bottom Sub-Array

The BIST clock frequency can be further improved by inverting the clock on the CLB slices in which the TPGs and the ORAs are implemented for BIST configuration #3 that has inverted clock on the DSP slices. The increase in BIST clock frequency (in MHz) for BIST configuration #3 that has inverted clock on the DSP slices as well as the TPGs and ORAs is illustrated in Figure 3.16.

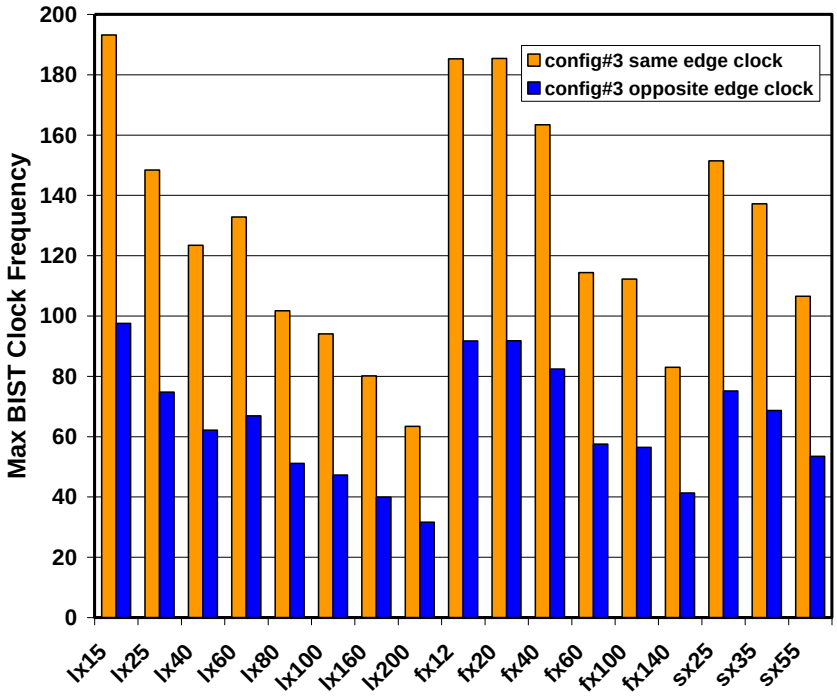


Figure 3.16 Timing Analysis Based on Clock Edge for Configuration #3

Figure 3.17 illustrates the maximum BIST clock frequency (in MHz) for BIST configurations #2 through #5 for DSP BIST when the TPGs, ORAs and DSPs have the same clock edge for all the configurations. BIST configuration #1 is not included in Figure 3.18 since timing analysis does not give an accurate result for configuration #1.

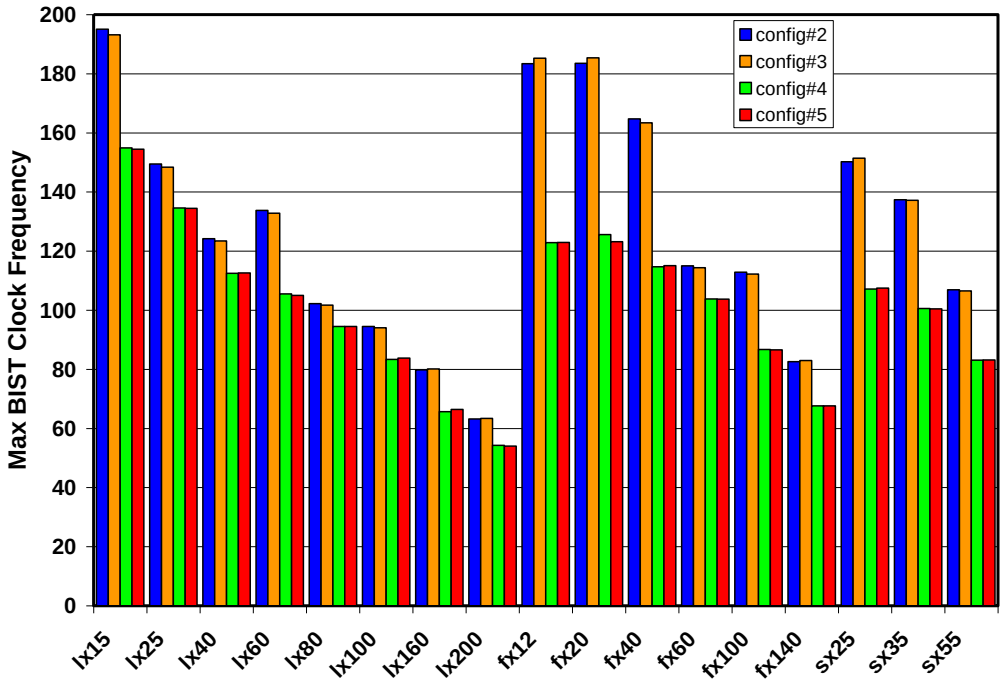


Figure 3.17 Timing Analysis for DSP BIST Configurations #2 through #5

Table 3.8 illustrates the increase in download time and test time caused by inverting the clock on the TPGs and ORAs for BIST configurations that have inverted clock on the DSP slices. This increase in download time happens because for BIST configuration #3 where the DSPs are clocked on the falling edge of the clock, the configuration memory of the TPGs and the ORAs has to be rewritten in order to match the clock edge of the TPGs and the ORAs with the falling clock edge of the DSPs, since the TPGs and ORAs are configured to clock on the rising edge of the clock in the previous BIST configuration. The order in which the BIST configurations are generated for the data presented in Table 3.8 is #1 through #5.

From Table 3.8, it is observed that using same edge clock for BIST configurations that have falling edge clock increases the download and test time by a maximum of 7.9% when BIST configurations are downloaded in the following order: #1, #2, #3, #4 and #5. This increase is not significant when compared to the overall download and test time.

Table 3.8 Configuration File Size and Test Time Increase for Same Edge Clock

	# Bits for Config #1			# Bits for Configs #2,#3,#4 & #5		Increase in Download Time (sec)		Increase in Test Time (sec)	
Device	**Full download**	**Compress download**	**%**	**Opposite edge clock**	**Same edge clock**	**Full download**	**Compress download**	**Full download**	**Compress download**
lx15	4,765,568	2,091,808	43.8	62,720	183,808	1.02507	1.05620	1.02503	1.05598
lx25	7,819,904	3,259,552	41.6	84,736	238,848	1.01949	1.04608	1.01947	1.04596
lx40	12,259,712	4,747,072	38.7	106,752	293,888	1.01513	1.03855	1.01512	1.03848
lx60	17,717,632	5,813,376	32.8	106,752	293,888	1.01049	1.03161	1.01049	1.03156
Lx80	23,291,008	7,512,768	32.2	128,768	348,928	1.00940	1.02881	1.00939	1.02877
lx100	30,711,680	9,421,696	30.6	150,784	403,968	1.00820	1.02644	1.00820	1.02642
lx160	40,347,008	10,342,528	25.6	150,784	403,968	1.00625	1.02412	1.00625	1.02410
lx200	51,367,808	11,714,848	22.8	150,784	403,968	1.03290	1.02133	1.00491	1.02132
fx12	4,765,568	1,924,288	40.3	62,720	183,808	1.02507	1.06093	1.02503	1.06068
fx20	7,242,624	2,277,344	31.4	62,720	183,808	1.01657	1.05174	1.01655	1.05156
fx40	14,936,192	4,548,160	30.4	84,736	238,848	1.01025	1.03326	1.01025	1.03320
fx60	21,002,880	7,805,024	37.1	194,816	514,048	1.01505	1.03990	1.01505	1.03986
fx100	33,065,408	11,609,824	35.1	238,848	624,128	1.01156	1.03251	1.01156	1.03249
fx140	47,856,896	15,736,192	32.8	282,880	734,208	1.00937	1.02817	1.00937	1.02816
sx25	9,147,648	4,948,448	54	194,816	514,048	1.03417	1.06206	1.03413	1.06196
sx35	13,700,288	7,367,104	53.7	282,880	734,208	1.03227	1.05899	1.03225	1.05893
sx55	22,745,216	13,322,656	58.5	723,200	1,835,008	1.04737	1.07915	1.04735	1.07910

With the inversion of clock on the TPGs on ORAs to match the clock edge on which the DSPs are clocked, the BIST clock frequency no longer depends on the edge of the clock used to clock the DSPs. So, configuration # 5 can be changed to clock the DSPs on the falling edge of the clock as this change might detect some of the undetected faults in Section 3.5. Table 3.9 illustrates the increase in the bitstream file size caused by the inversion of clock edge on the TPGs and the ORAs to match the clock edge of the DSPs. for an SX55 device. From Table 3.9, it is observed that the file size also depends on the order in which the BIST configurations are generated. Table 3.10 illustrates the BIST configurations for Virtex-4 DSP BIST in the order in which they should be generated.

Table 3.9 Download File Sizes (in Bits) for an SX55 Device

Download #	DSPs in configurations #3 and #5 have falling edge clock		Clock egde on TPGs and ORAs is matched with clock edge on DSPs		Clock edge on TPGs and ORAs is matched with clock edge on DSPs	
	BIST #		BIST #		BIST #	
Download #1 (compressed)	1	13294720	1	13294720	3	13294720
Download #2 (partial reconfiguration)	2	180800	2	180800	5	180800
Download #3 (partial reconfiguration)	3	180800	3	736704	1	736704
Download #4 (partial reconfiguration)	4	180800	4	736704	2	180800
Download #5 (partial reconfiguration)	5	180800	5	736704	4	180800
Total	14017920		15685632		14573824	

Table 3.10 BIST Configurations for Virtex-4 DSP BIST

BIST Config #	Pipeline Registers	Signals Active Level	B Input Source		Test Modes Applied			Clock edge of TPGs and ORAs
			Slice 0	*Slice 1*	*Mult*	*Add*	*Casc*	
1	A/Breg=2, Others=1	Low	Direct	Direct	#1	#2		Low
2	Preg=1, Others=0	Low	Direct	Cascade	#3			Low
3	All Regs = 0	High	Direct	Direct	#4			High
4	All Regs = 1	High	Direct	Direct	#5	#6		High
5	Preg=1, Others=0	High	Cascade	Direct			#7	High

3.7 Summary

A minimum set of BIST configurations was developed to test the DSP cores in Virtex-4 FPGAs. Fault detection capabilities and fault diagnosis were verified by injecting faults into the configuration memory bits controlling the DSP cores in an FX12 device. DSP BIST was also able to detect faulty DSP cores in some of the SX35 and LX60 engineering sample parts. Fault

coverage of 97.4% is achieved for the faults injected in the configuration memory of the DSP. The functional fault coverage as determined by fault simulations is much higher. Fault coverage for the faults injected in the configuration memory of the DSP can be improved to 100% by adding more BIST configurations if desired. Since these undetected faults are in nonfunctional modes of operation the value of additional BIST configurations is questionable. Maximum BIST clock frequency was improved by changing the position of the TPG in the chip array. To further improve the BIST frequency on larger Virtex-4 devices, where the BIST frequency is less than 50MHz, sub-array testing is done. Sub-array testing also minimizes the power dissipation caused by testing a large number of DSPs simultaneously, as this can cause problems in the system. When same edge clock is used on the TPGs, ORAs and DSPs for all configurations, the maximum BIST clock frequency is well over 50 MHz. But sub-array testing may still be required for larger devices that have large numbers of DSPs to minimize power dissipation.

Chapter 4

BIST for DSP Cores in Virtex-5 FPGAS

This chapter describes the implementation of BIST for DSPs in Virtex-5 FPGAs. The BIST architecture, along with the BIST configurations and test sequences for DSP cores in the FPGAs, are discussed. The chapter also discusses the retrieval of BIST results and the timing analysis of BIST for all Virtex-5 FPGAs. The chapter concludes by summarizing the experimental BIST results and the fault coverage achieved.

4.1 BIST Approach for DSPs in Virtex-5 FPGAs

Since most of the features of DSPs in Virtex-5 FPGAs are similar to the features of DSPs in Virtex-4 FPGAs, as explained in Chapter 2, the test algorithms used to test the DSPs in Virtex-4 FPGAs can also be applied to test DSPs in Virtex-5 FPGAs. The additional features in DSPs of Virtex-5 FPGAs can be tested by making modifications to the TPG.

4.1.1 Adder and Multiplier Tests

The adder test algorithm described in Section 3.1.1 of Chapter 3 can be used to test the adder in Virtex-5 FPGAs as well. Although the adder in Virtex-5 FPGAs can be accessed through two 48-bit input ports and can be tested using a one clock cycle per vector approach, the two-clock cycle per vector approach described in Section 3.1.1 of Chapter 3 is used to be able to also test the accumulator register with the adder. The 5×3 and the 3×5 multiplier test algorithms described in Section 3.1.2 of Chapter 3 can be applied to test the multiplier in Virtex-5 FPGAs. Like the test sequences for DSPs in Virtex-4 FPGAs, the test sequences for DSPs in Virtex-5 FPGAs are also 1024 clock cycles long and are divided into four groups of 256 clock cycles

each. The adder and the multiplier test sequences are illustrated in Table 4.1. During the first group of 256 clock cycles of the adder test sequence, the first stage of the adder is tested and, during the second group of 256 clock cycles, the second stage adder is tested. During the third and fourth groups of 256 clock cycles in the adder test sequence, other paths to the adder are tested. The 5×3 multiplier test algorithm is applied to the multiplier in slice 1 while the 3×5 multiplier test algorithm is applied to the multiplier in slice 0 during the first group of 256 clock cycles in the multiplier test sequence. During the second group of 256 clock cycles in the multiplier test sequence the 5×3 multiplier test algorithm is applied to the multiplier in slice 0 and the 3×5 multiplier test algorithm is applied to the multiplier in slice 1. The 5×3 multiplier test algorithm is applied by replicating the five MSB bits of the vector generated by the 8-bit counter and applying the replicated bits to the 30-bit A port while the three LSB bits of the vector generated by the 8-bit counter are replicated and applied to the 18-bit B port. During the 3×5 multiplier test algorithm the three LSB bits of the vector generated by the 8-bit counter are replicated and applied to the 30-bit A port while the five MSB bits of the vector generated by the 8-bit counter are replicated and applied to the 18-bit B port. The application of different multiplier test algorithms to slice 0 and slice1 ensures that the A and B ports in both slices receive different test patterns on every clock cycle so that the stuck-at faults on the multiplexers that select between direct and cascade paths on A and B ports of the DSP slice can be tested. During the third group of 256 clock cycles in the multiplier test sequence, the multiply and add function is tested and during the fourth group of the multiplier test sequence the A port concatenated with the B port, bypass of the multiplier, is tested.

Table 4.1 Multiplier and Adder Test Sequences

Test	First 256 ccs	Second 256 ccs	Third 256 ccs	Fourth256 ccs
Multiply	P = A×B	P = A×B	P = A×B+C	P = A:B+C
Adder	P = Z(C) P = X(P)+Y(C)	P = Y(C) P = Y(C)+Z(P)	P = Z(C) P = Y(C)+Z(P)	P = Y(C) P = Y(C)+Z(ShiftP)

4.1.2 Pattern Detector Test

From the datasheet it is understood that the pattern detector mainly checks for the equality between the output of the adder/subtractor/logic unit and a pattern given by the user. This pattern can be given dynamically through the C port or statically through the configuration memory bits. A mask field masks individual bits that are not considered during the comparison process [10]. Like the pattern, the mask can be given by the user dynamically through the C port or statically through the configuration memory bits. Multiplexers controlled by configuration memory bits select between dynamic or static patterns and masks. The pattern detector has two outputs, Patterndetect and Patternbdetect. The Patterndetect output goes to logic '1' if the output of the adder/subtractor/logic unit matches the user-defined pattern and the Patternbdetect output goes to logic '1' if the output of the adder/subtractor/logic unit matches the complement of the user-defined pattern. Although the data sheet does not explicitly define the architecture of the pattern detector, it is mentioned that the pattern detect logic performs a bitwise ((P= =pattern)||mask) and then ANDs the results to a single bit result [10]. From this equation the architecture for the Patterndetect logic is reasoned to be as illustrated in Figure 4.1.

Figure 4.1 illustrates a 4-bit architecture for the pattern detect logic where o[4:1] indicate the output of the adder/subtractor/logic unit, p[4:1] indicate the user-defined pattern and m[4:1] indicate the mask. The AND gate shown Figure 4.1 can be completely tested by walking a 0

through a field of 1s and applying the all 1s pattern. Table 4.2 illustrates the test vectors to test a 4-bit pattern detect logic.

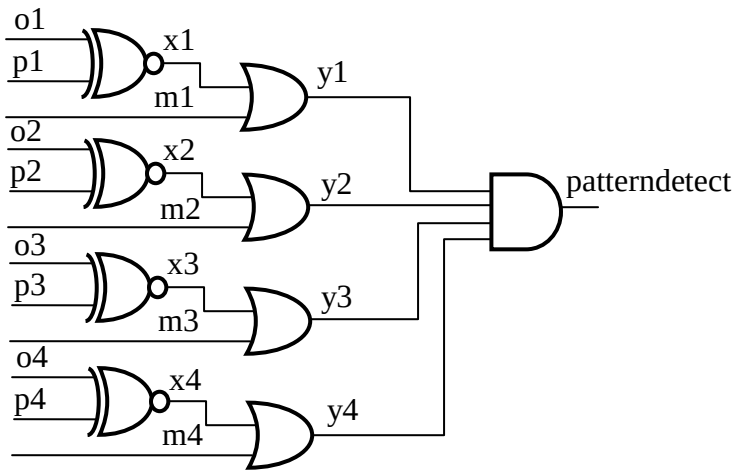


Figure 4.1 Architecture for a 4-bit Patterndetect Logic

Table 4.2 Test Vectors for Testing the 4-bit Patterndetect Logic

o1	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
p1	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0
o2	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
p2	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
o3	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
p3	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0
o4	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0
p4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
m [4:1]	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
x1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1
X2	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1
x3	1	1	1	1	1	0	1	1	1	1	1	1	1	0	1	1
x4	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0
y1	1	0/1	1	1	1	1	1	1	1	0/1	1	1	1	1	1	1
y2	1	1	1	0/1	1	1	1	1	1	1	1	0/1	1	1	1	1
y3	1	1	1	1	1	0/1	1	1	1	1	1	1	1	0/1	1	1
y4	1	1	1	1	1	1	1	0/1	1	1	1	1	1	1	1	0/1

From Table 4.2 it observed that all four combinations of input patterns (00, 01, 10 and 11) are applied to the XNOR gates. While applying these set of patterns the mask is set to have patterns with alternate 0s and 1s that are applied statically through the configuration memory

bits. This allows the application of all four combinations of patterns (00, 01, 10 and 11) on the inputs of the OR gates. The shaded portions in Table 4.2 illustrate all the vectors that are applied to the Patterndetect logic.

The datasheet also mentions that the Patternbdetect logic performs the logic equation ((P= =~pattern)||mask) [10]. Although duplicating the architecture shown in Figure 4.1 with XOR gates instead of XNOR gates would satisfy this equation, this implementation requires a lot of logic. Hence, the architecture for the Patternbdetect logic is reasoned to be as illustrated in Figure 4.2. In Figure 4.2, o[4:1] indicate the output of the adder/subtractor/logic unit, p[4:1] indicate the user-defined pattern and m[4:1] indicate the mask. The NOR gate shown in Figure 4.2 can be completely tested by walking a 1 through a field of 0s and applying the all 0s pattern. This is achieved by inverting any one of the input bits at the XNOR gates in Table 4.2. Table 4.3 illustrates the test vectors for the Patternbdetect logic.

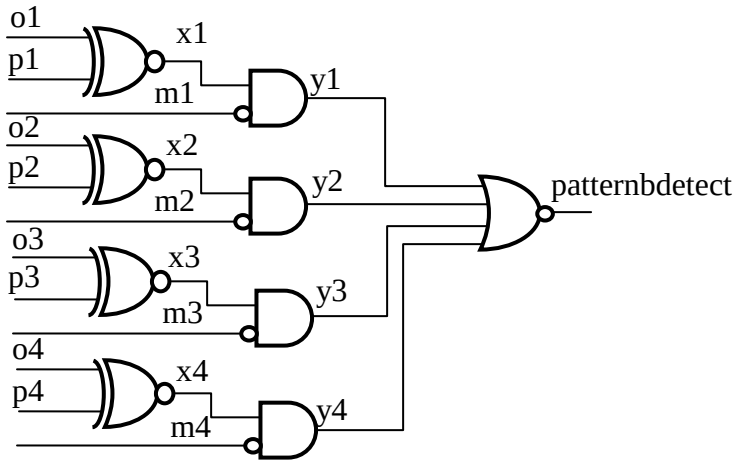


Figure 4.2 Architecture for a 4-bit Patternbdetect Logic

The set of test patterns illustrated in Table 4.3 applies all possible input combinations (00, 01, 10 and 11) to the XNOR gates. The mask is set to have patterns with alternate 0s and 1s applied statically through configuration memory bits. This ensures that the AND gates in Figure

4.2 are completely tested. The shaded portions in Table 4.3 illustrate all the vectors that are applied to the Patternbdetect logic.

Table 4.3 Test Vectors for Testing the 4-bit Patternbdetect Logic

o1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
p1	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0
o2	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1
p2	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
o3	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1
p3	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0
o4	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1
p4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
m [4:1]	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
x1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
X2	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
x3	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
x4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
y1	0	1/0	0	0	0	0	0	0	0	1/0	0	0	0	0	0	0
y2	0	0	0	1/0	0	0	0	0	0	0	0	1/0	0	0	0	0
y3	0	0	0	0	0	1/0	0	0	0	0	0	0	0	1/0	0	0
y4	0	0	0	0	0	0	0	1/0	0	0	0	0	0	0	0	1/0

Figure 4.3 illustrates the TPG architecture used to generate test vectors for the pattern detector. A 49-bit shift register and a 47-bit shift register are cascaded to form a 96-bit shift register. A transition of either logic ‘1’ or logic ‘0’ on the MSB bit of the 49-bit shift register enables the 47-bit shift register. The 49-bit shift register starts shifting first until there is a transition of either logic ‘1’ or logic ‘0’ on its MSB bit and from then onwards the 49-bit shift register is enabled only when the MSB bit of the 47-bit shift register undergoes a transition from logic ‘1’ or logic ‘0’. The ‘cs’ bit in Figure 4.3 is set to ‘0’ to generate the test vectors illustrated in Table 4.2 and is set to ‘1’ to generate the test vectors illustrated in Table 4.3. The 49-bit shift register is the same shift register that generates test patterns for the adder test. The 47-bit shift register is disabled during the adder test.

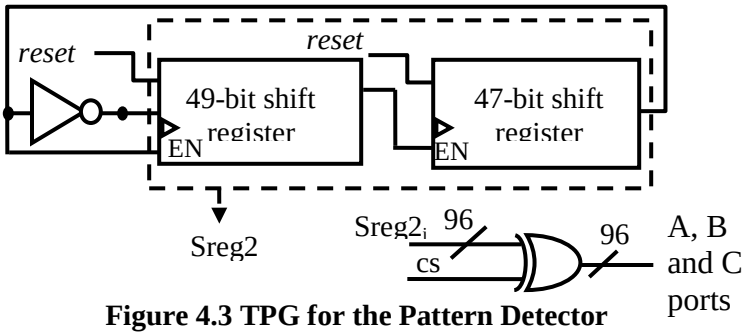


Figure 4.3 TPG for the Pattern Detector

In addition to the static and dynamic masks the pattern detector can select its mask from two other masks, selected by user-defined attributes ‘mode1’ and ‘mode2’. These masks used for rounding applications are determined by the C-port input and change as the C-port input changes [10]. The ‘mode1’ attribute selects the mask to be the complement of the C-port input left shifted by 1 and the ‘mode2’ user attribute selects the mask to be the complement of the C-port input left shifted by 2. Multiplexers select the final mask from four options: a dynamic mask given through the C-port, a static mask given through configuration memory bits, and the two other masks explained above. The multiplexers that choose the final mask and the final pattern are modeled as illustrated in Figure 4.4 [10]. The SEL_MASK user attribute, also a configuration memory bit, selects between the dynamic mask given through the C-port and the static mask given through the configuration memory bits. The mask selected by the SEL_MASK user attribute is fed into a 3-input multiplexer that selects between the mask selected by the SEL_MASK attribute and the ‘mode1’ and ‘mode2’ masks [10]. The user attribute SEL_ROUNDING_MASK, the select signal to this three input multiplexer, is a combination of two configuration memory bits, CB1 and CB2 as illustrated in Figure 4.5.

The auto-reset feature of the pattern detector is used to reset (when user attribute AUTORESET_PATTERN_DETECT, which is also a configuration memory bit, is set to be TRUE) the output P register of the DSP slice on the subsequent clock cycle after match is

detected between the pattern and the output of the DSP slice or if a pattern was detected on the previous clock cycle but is now not detected [10]. The auto-reset feature can also be used to not reset (when AUTORESET_PATTERN_DETECT is set to be FALSE) the output P register if one of the above explained conditions is met. The user attribute, AUTORESET_ PATTERN_ DETECT_ OPTINV, also a configuration memory bit, is set to MATCH to reset or not reset the P register on the subsequent clock cycle if a pattern is detected and is set to NOT_MATCH to reset or not reset the output P register if a pattern was detected on the previous clock cycle but is now not detected. The architecture of the auto-reset logic is explicitly defined in the datasheet as illustrated in Figure 4.6 [10].

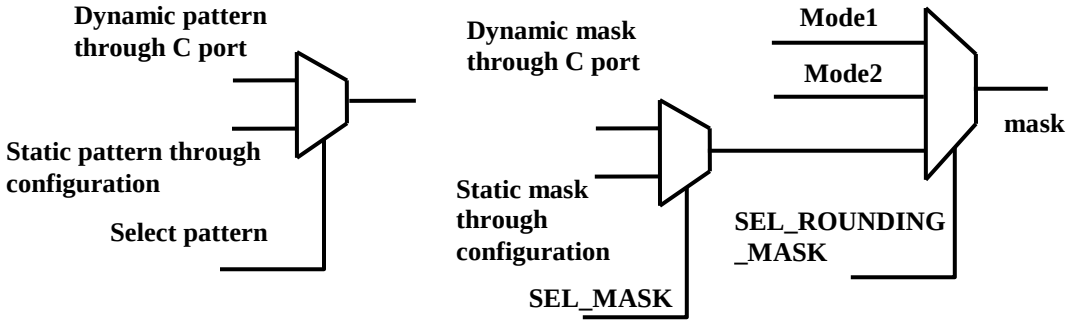


Figure 4.4 Multiplexer Architecture for Selecting the Pattern and the Mask [10]

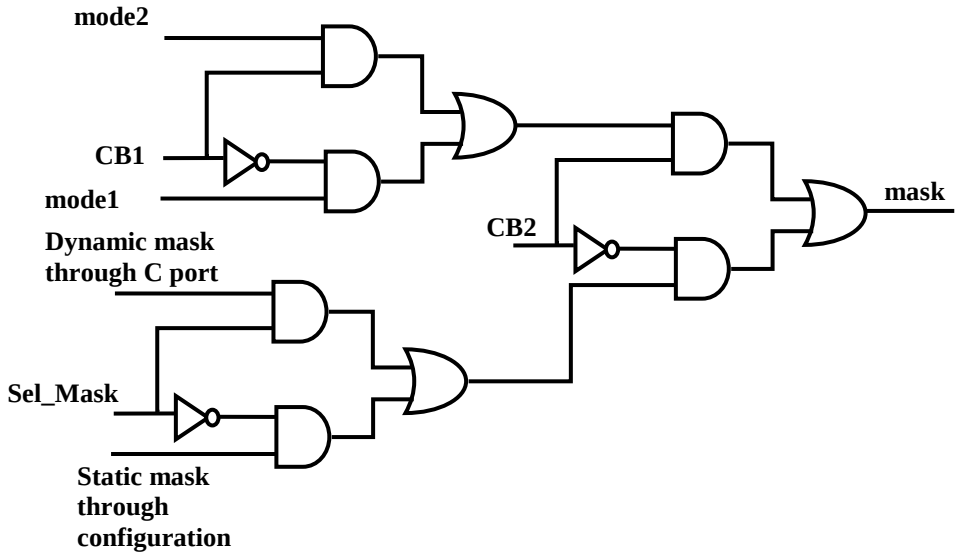


Figure 4.5 Detailed Multiplexer Architecture for Selecting Mask

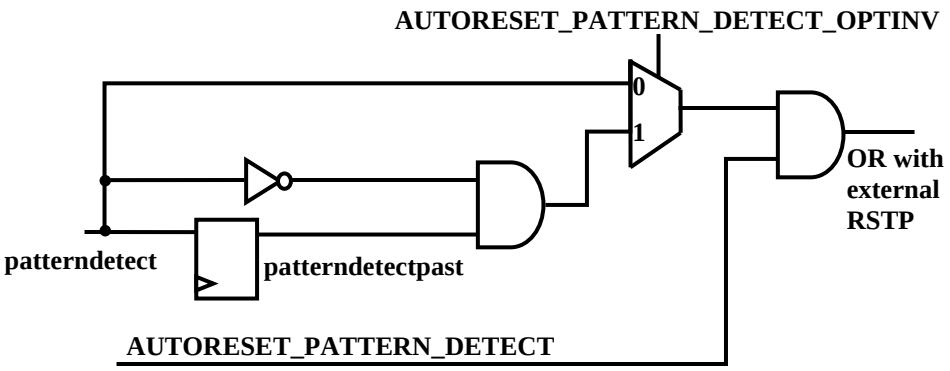


Figure 4.6 Auto Reset Logic [10]

The overflow and underflow logic associated with the pattern detector is used to check for overflow or underflow beyond any particular bit position between 0 and 46. The mask determines the threshold for overflow or underflow while the pattern is set to 000…00 <47:0>. The value beyond which the output of the DSP slice overflows is 2^N-1 where *N* is the number of ones in the mask field and the value beyond which the output of the DSP slice underflows is the twos complement form of negative 2^N. The architecture of the overflow and underflow logic is explicitly defined in the datasheet as illustrated in Figure 4.7 [10].

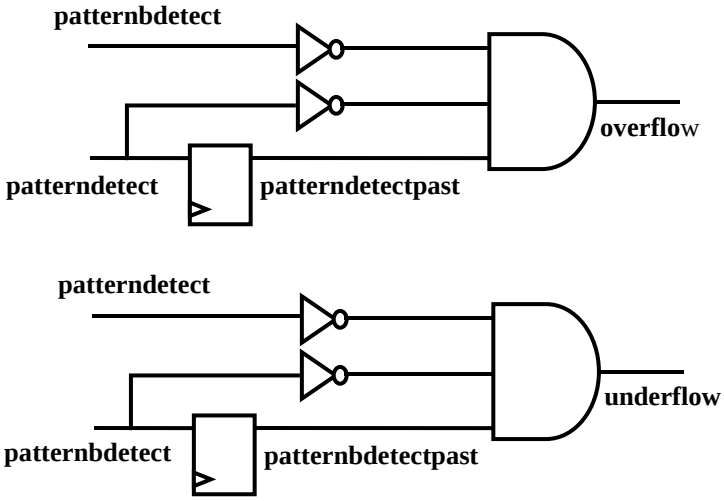


Figure 4.7 Overflow and Underflow Logic [10]

The BIST configurations illustrated in Table 4.4 completely test the pattern detector and its associated multiplexer paths, auto-reset, overflow and underflow logic. To apply the correct

test vectors to the pattern detector, the output of the adder/subtractor/logic unit is set to be same as the concatenated A and B ports given through the X multiplexer while applying 0s to the Y and Z multiplexers. The pattern detector test sequence is 1024 clock cycles long and during the entire sequence OPMODE bits <1:0> select the A concatenated with B input through the X multiplexer port while OPMODE bits <3:2> apply 0s to the Y multiplexer port and OPMODE bits <6:4> apply 0s to the Z multiplexer port. Static or dynamic patterns and masks are set as per the BIST configuration. APDO and APD in Table 4.4 indicate Auto-Reset Pattern Detect Optinv and Auto-Reset Pattern Detect user attributes, respectively.

Table 4.4 BIST Configurations for the Pattern Detector

P_{STATIC} <47:0>	M_{STATIC} <47:0>	SEL_PATTERN	SEL_MASK	CB1	CB2	APDO	APD
0101..01	1111..11	static	dynamic	0	0	0	1
1111..11	0101..01	dynamic	static	1	0	1	1
1010..10	1111..11	static	dynamic	0	0	0	0
1111..11	1010..10	dynamic	static	1	0	1	0
x	x	dynamic	dynamic	0	1	0	1
x	x	dynamic	dynamic	1	1	0	1

An 8-bit pattern detector with its auto-reset, overflow and underflow logic along with the multiplexers that select the mask and the pattern were written in ASL (Auburn Simulation Language) using AUSIM (Auburn University Simulator) to determine the efficiency of the test algorithm and the BIST configurations. Of the 568 single stuck-at gate-level faults and the 2276 bridging faults generated using AUSIM, the BIST configurations with the test vectors illustrated in Tables 4.2 and 4.3 detect all detectable single stuck-at and bridging faults.

4.1.3 ALU Logic Mode Test

As described in Section 2.2 of Chapter 2, the DSPs in Virtex-5 FPGAs have an ALU logic mode of operation controlled by ALUMODE control signals and OPMODE bits <3:2>. The ALU logic unit in the DSPs performs bitwise logical XOR, XNOR, OR, AND, NAND,

NOR and NOT operations on two 48-bit inputs as described in Table 2.3 of Chapter 2 [10]. When the logic mode is used in the DSP slices, the multiplier is not used and the 30-bit A and 18-bit B input ports of the DSP slice are concatenated to form a 48-bit input to the logic unit. The other 48-bit input is provided by the C input port of the DSP slice [10]. The ALUMODE test sequence is 1024 clock cycles long and during the entire test sequence the OPMODE bits <1:0> select the A concatenated with B input through the X multiplexer and the OPMODE bits <6:4> select the C input through the Z multiplexer.

The ALU logic mode can be tested using the same 8-bit counter that is used to test the multiplier in the DSPs. During a 256 clock cycle period the ALUMODE is tested for all the logical operations described in Table 2.3 of chapter 2. As described in Table 2.2 of chapter 2, the ALUMODE values ‘0000’ and ‘0011’ correspond to the adder and subtractor functions that are tested during the adder test sequence, the remaining adder/subtractor functions in Table 2.2 that correspond to ALUMODE values ‘0001’ and ‘0010’ are tested during the ALUMODE test sequence as well. Counter<0> generates test patterns for the CARRYIN input of the adder/subtractor unit. Counter<1> generates test patterns for the A concatenate B input of the adder/subtractor/logic unit and counter<2> generates test patterns for the C input of the adder/subtractor/logic unit. Counter bits <6:3> generate values for the ALUMODE control bits and counter<7> generates values for the control bit OPMODE <3> while the control bit OPMODE<2> is held constant at logic ‘0’. During the 256 clock cycle period all possible combinations of test vectors (00, 01, 10 and 11) are applied to the two inputs of the logic unit. Bridging faults in the logic unit can be tested by increasing the width of the counter to 10-bits and using the additional bits to control every other bit of the two inputs to the logic unit.

4.1.4 Cascade Mode Test

The DSP slices in Virtex-5 FPGAs can be cascaded to implement larger multipliers and adders used for extended MACC (Multiply and Accumulate) functions. Unlike the DSPs in Virtex-4 FPGAs where the B and P ports of a DSP slice can be cascaded, the A, B and P ports of a DSP slice can be cascaded in Virtex-5 FPGAs. The sign bit of the multiplier output can also be cascaded in Virtex-5 FPGAs. The choice between direct and cascade paths for the A and B ports is made by configuration memory bits [10]. The A and B ports can have up to two pipeline registers in their direct and cascade paths. The number of pipeline registers in these paths is determined by configuration memory bits. The user attributes A_INPUT and B_INPUT select between direct and cascade paths of the A and B ports. The user attributes AREG, ACASREG, BREG and BCASREG select the number of pipeline registers. The acceptable combinations of values for the user attributes that select the number of pipeline registers are summarized in Table 4.5. The architecture of multiplexers that choose between direct and cascade paths and the number of pipeline registers in these paths is illustrated in Figure 4.8 [10].

Table 4.5 Values for A and B Pipeline Registers [10]

Areg and Breg of DSP slice that cascades to the slice above	Areg and Breg of DSP slice that receives cascade inputs from slice below
0	0
1	1
2	1,2

Form Figure 4.8 it is seen that the A and B cascade paths in the DSP slice are tested when all paths through MUX 4 are tested. This is achieved by setting the values of user attributes AREG and BREG to ‘2’ while setting values of user attributes ACASREG and BCASREG to ‘2’ and by setting the values of user attributes AREG and BREG to ‘1’ while setting values of user attributes ACASREG and BCASREG to ‘1’. The DSP slices in Virtex-5 FPGAs are cascaded in

pairs. Since slice 1 and slice 0 are individually configured to operate in cascade mode, a total of four BIST configurations are required to test the cascade mode of operation.

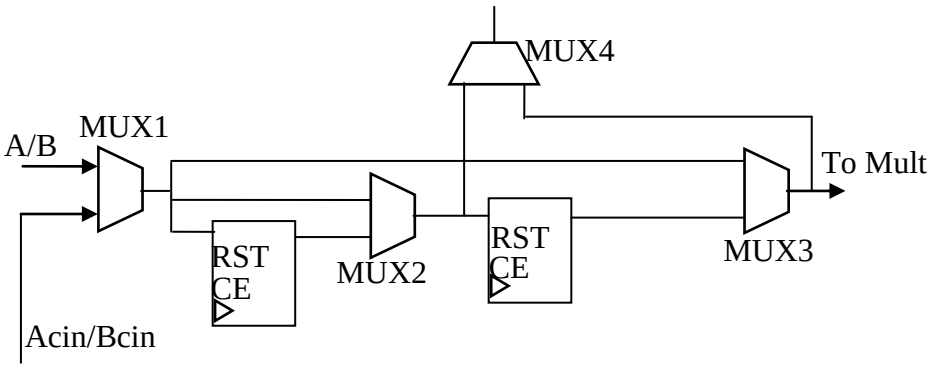


Figure 4.8 Multiplexer Architecture that Selects Between Direct and Cascade Paths of A and B Ports [10]

4.1.5 SIMD Mode Test

The SIMD mode of operation in Virtex-5 FPGAs, as described in Section 2.2 of Chapter 2, splits the 48-bit adder into two 24-bit adders or four 12-bit adders. The USE_SIMD user attribute selects between these three adder architectures [10]. Hence it can be understood that the 48-bit adder in Virtex-5 FPGAs is constructed from four 12-bit CLA adders, where each 12-bit CLA adder is constructed from three basic CLA adder units described in Section 2.3 of Chapter 2. In the two 24-bit adder architecture mode, configuration bits block the propagation of the carryout signal between the second and the third 12-bit adders [10]. In the four 12-bit adder architecture mode the carryout signals are not allowed to propagate between individual 12-bit adders and are blocked by configuration bits. In the one 48-bit adder architecture mode, the propagation of carryout signals between individual 12-bit adders is not blocked by configuration bits [10].

By testing the SIMD mode of operation in the one-48 and four-12 modes, all configuration bits that select between the three adder architectures will have been tested. The SIMD ‘one 48’ mode is tested during configurations #2 and #3 as summarized in Table 4.7.

Configurations #2 and #3 also test the functionality of the adder. The SIMD four 12-bit mode is tested during configuration #8 when the DSPs are in cascade mode of operation as summarized in Table 4.7. Since the functionality of the adder is already tested in configurations #2 and #3, the faults associated with the SIMD four 12-bit mode can be tested while testing the cascade mode of operation without adding another configuration just for testing the adder in the SIMD 'four 12' mode of operation.

4.1.5 MACC Extend Mode Test

Cascade signals CARRYCASCIN, CARRYCASCOUT, MULTSIGNIN and MULTSIGNOUT are internal to the DSP and are used to build larger adders and multipliers. The most significant bit (MSB) of the multiplier output functions as MULTSIGNOUT and is used in MACC extension applications to build a 96-bit MACC. The MULTSIGNOUT bit is cascaded to the DSP slice above through its corresponding MULTSIGNIN port. The carryout from the DSP slice below is also added along with the MULTSIGNOUT bit [10]. The test patterns used to test the multiplier are also applied to test the MACC extend feature. The MACC extend feature is tested during configuration #2 as summarized in Table 4.7.

4.2 BIST Architecture

The BIST architecture used for DSPs in Virtex-5 FPGAs is same as the BIST architecture used for DSPs in Virtex-4 FPGAs, as described in Section 3.2 of Chapter 3. Two TPGs are used to drive alternate rows of DSP tiles so that a faulty TPG can be detected. Both slices of a DSP tile are driven by the same TPG for individual control of slices during cascade mode of operation. The control register that controls the TPG in Virtex-5 FPGAs is five bits. Of the five bits, the three LSBs, called MODE2, MODE1 and MODE0, control the test mode of the TPG. The second MSB, called INVCS, controls the active levels of control signals such as the

OPMODE bits, ALUMODE bits and the active level of the carryin input to the adder unit. The MSB provides a global reset to the TPG. The control register is implemented in a CLB and the values for the control register are shifted in through the Boundary Scan interface, where the LSB is shifted in first. The control register values for resetting the TPG and for the various test modes are illustrated in Table 4.6. Each DSP slice is observed by two sets of ORAs. Similarly configured slices of the DSP are compared using column-based circular comparison. The ORAs that observe the bottom-most DSP slice in a column of DSPs are clock enabled so as to mask the failure indications due to the unconnected cascade inputs on the bottom-most DSP slice in a column of DSPs.

The architecture of ORAs that observe DSP slices in Virtex-5 FPGAs is different from the architecture of ORAs that observe DSP slices in Virtex-4 FPGAs. Since the CLBs in Virtex-5 FPGAs have 6-input look-up tables, each look-up table can compare two different output bits of two individual DSP slices, unlike the CLBs in Virtex-4 FPGAs, where the look-up tables observed only one output bit from two individual DSP slices. If any one of the two bits being compared mismatches, the ORA outputs a logic '0'. The ORA architecture for DSPs in Virtex-5 FPGAs is illustrated in Figure 4.9 [24]. A DSP slice in a Virtex-5 FPGA has 56 outputs. Figure 4.10 illustrates the inputs and outputs associated with a DSP slice. Since each CLB slice has four 6-input look-up tables, seven CLB slices are needed to analyze all 56 outputs. The ORAs that analyze the DSPs under test are placed in two columns of CLBs beside the column of DSPs. The bottom slices, slice 0s, of the DSP tiles are compared by the first column of ORAs and the top slices, slice 1s, of the DSP tiles are compared by the second column of ORAs. The ORAs that analyze each DSP slice under test occupy five rows in a single column of CLBs. Since only seven CLB slices are required to analyze all the DSP outputs and each row has two CLB slices,

the remaining three CLB slices are *dummy ORAs* and do not analyze a DSP under test but generate the carry logic used to construct the iterative OR chain.

Table 4.6 Control Register Values for TPG Control

Control Register Values <4:0> RESET	INVCS	MODE2	MODE1	MODE0	Operation
1	X	X	X	X	Resets TPG
0	1	X	X	X	Inverts active level of control signals
0	X	0	0	0	Sets the multiplier test mode
0	X	0	0	1	Sets the adder test mode
0	X	0	1	0	Sets the logic test mode
0	X	0	1	1	Sets the patern detector test mode
0	X	1	0	0	Sets cascade test mode

Figure 4.11 shows the orientation of the ORAs with respect to the DSP slices under test in Virtex-5 FPGAs. The dedicated carry logic in the CLBs of Virtex-5 FPGAs can also be used to create an iterative OR chain of ORAs where the Test Data In (TDI) line of the Boundary Scan module is connected to the carry-in of the first CLB in the column of ORAs and the carry-out of the last CLB in the last column of ORAs is connected to the Test Data Out (TDO) line of the Boundary Scan module. This provides a single bit pass/fail result for the entire test. The TDO line goes to logic '1' when any one ORA in the iterative OR chain fails. Individual ORA results can be read back to determine which ORAs have failed the test when the overall test fails.

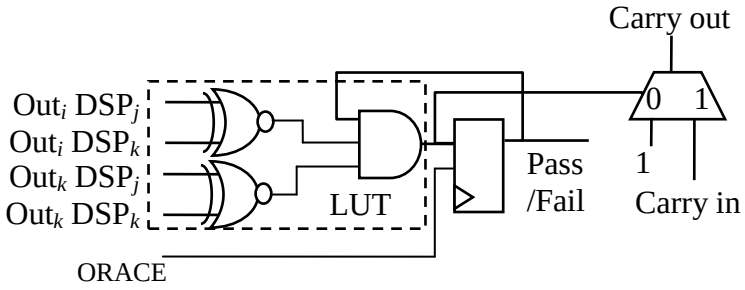


Figure 4.9 ORA Architecture [24]

Figure 4.12 illustrates the general architecture of the TPG used to test DSPs in Virtex-5 FPGAs. The TPG for DSP BIST comprises a 10-bit counter where the two MSB bits are used for

the individual control of the four 256 clock cycle test groups during the 1024 clock cycle test sequence, a 49-bit shift register for the adder test, a finite state machine (FSM) for control of OPMODE control signals and two 10-bit linear feedback shift registers (LFSRs) for generating pseudorandom signals for testing the resets and clock enables on the pipeline registers as well as other control signals in the DSP slice that choose the source for the carryin input of the adder unit. The eight least significant bits of the counter are used to apply test patterns to the multiplier and the ALU logic mode. The 49-bit shift register, along with a 47-bit shift register, generate test vectors for the pattern detector. The TPG is written in VHDL (580 lines of code) and is synthesized into 99 CLB slices.

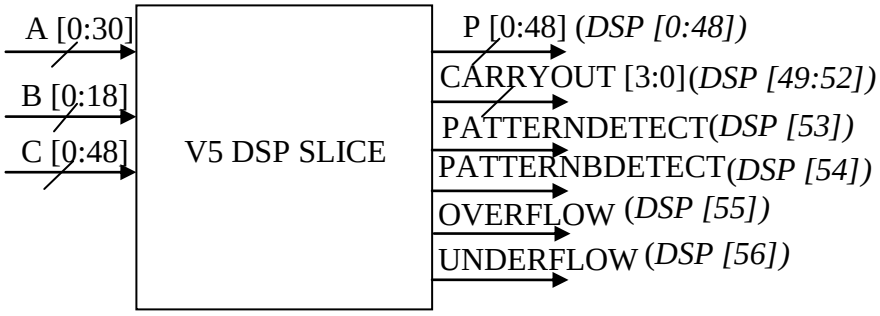


Figure 4.10 I/O of a Virtex-5 DSP Slice

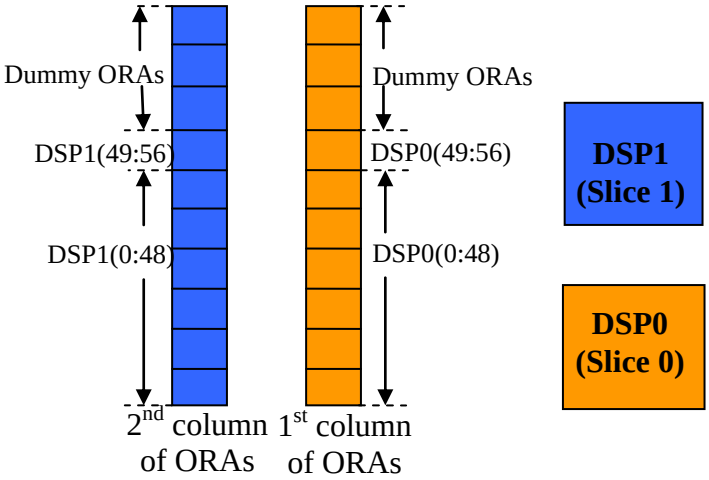


Figure 4.11 DSP ORA Orientation in Virtex-5 FPGAs

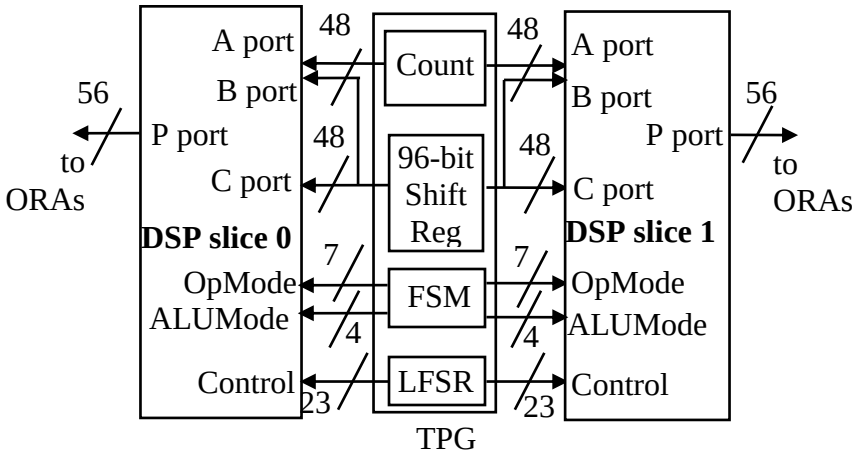


Figure 4.12 TPG Architecture

4.3 BIST Configurations and Sequences

The DSP cores are tested in eleven BIST configurations. During these eleven BIST configurations, the DSP configuration memory bits are tested in all configurable combinations. Table 4.7 summarizes the BIST configurations for the pattern detect logic explained in Section 4.1.2. Table 4.8 illustrates all the BIST configurations that are downloaded to test the DSP cores. In Table 4.8, column 1 indicates the BIST configuration download number, column 2 indicates the number of pipeline registers in the I/O paths of the DSP slice and the values for various user attributes, column 3 indicates the active level of the DSP slice control signals, column 4 indicates whether the A or B port of the DSP slice is in cascade or direct mode of operation and column 5 indicates the test modes that are being tested for each configuration.. BIST configurations #2 and #3 are repeated five times to test the various modes of operation, as illustrated in Table 4.8. Hence the DSPs are tested in 11 BIST configurations and 20 BIST sequences. The multiplier and MACC extend tests run during configuration #3 do not functionally test the multiplier and the MACC extend feature since the USE_MULT attribute is set to ‘none’ in this configuration to detect faults associated with this attribute. However, the multiplier and the MACC extend feature are functionally tested during configuration #2.

Table 4.7 BIST Configurations for Pattern Detect Logic

User Attribute Values	BIST Configuration #					
	1	**2**	**3**	**4**	**5**	**6**
Sel_Mask	C	C	Mask	Mask	C	C
Sel_Rounding_Mask	Sel_Mask	Sel_Mask	Sel_Mask	Sel_Mask	Mode1	Mode2
Sel_Pattern	Pattern	Pattern	C	C	C	C
Auto_Reset_Pattern_Detect	False	False	True	True	True	False
Autoreset_Pattern_Detect_Optinv	Match	Match	Match	Not_Match	Match	Not_Match
Mask <47:0>	FF…FF	FF…FF	55…55	AA…AA	55…55	55…55
Pattern <47:0>	55…55	AA…AA	FF…FF	FF…FF	FF…FF	FF…FF

A total of 20 test sequences are applied in eleven downloads to the FPGA. The FPGAs are repeatedly reconfigured and tested until they have been tested in all modes of operation. BIST Sequences for testing the DSPs in Virtex-5 devices are summarized in Table 4.9 below. The variables used in Table 4.8 are explained in Table 4.10. The download time can be minimized using partial configurations, through which only portions of the configuration memory that contain DSP configuration memory bits are written instead writing the whole configuration memory. This can be done by maintaining constant placement of the TPGs, the ORAs and the DSPs and by keeping the routing constant between them.

Table 4.8 BIST Configurations for Virtex-5 DSPs

BIST #	Pipeline registers and Attribute values	Signal Active Level	A & B Input Source		Test Modes Applied						
			Slice0	Slice1	Mult	Add	ALU	Pat Det	Casc	MACC Extend	SIMD /casc
1	all regs=0 use_mult=mult use_SIMD=one48 use_patdet=no_patdet	H	D	D	yes	no	yes	no	no	no	no
2	all regs=1 use_mult=mult_s use_SIMD=one 48 use_patdet=patdet patdet config #1 [T4.7]	H	D	D	yes	yes	yes	yes	no	yes	no

BIST #	Pipeline registers and Attribute values	Signal Active Level	A & B Input Source		Test Modes Applied						
			Slice0	Slice1	Mult	Add	ALU	Pat Det	Casc	MACC Extend	SIMD /casc
3	all regs=1 use_mult=none use_SIMD=one 48 use_patdet=patdet patdet config #2 [T4.7]	L	D	D	no	yes	yes	yes	no	no	no
4	all regs=1 use_mult=mult_s use_SIMD=one 48 use_patdet=patdet patdet config #3 [T4.7]	H	D	D	no	no	no	yes	no	no	no
5	all regs=1 use_mult=mult_s use_SIMD=one 48 use_patdet=patdet patdet config #4 [T4.7]	H	D	D	no	no	no	yes	no	no	no
6	all regs=1 use_mult=mult_s use_SIMD=one 48 use_patdet=patdet patdet config #5 [T4.7]	H	D	D	no	no	no	yes	no	no	no
7	all regs=1 use_mult=mult_s use_SIMD=one 48 use_patdet=patdet patdet config #6 [T4.7]	H	D	D	no	no	no	yes	no	no	no
8	Areg=2 Breg=2 Acascreg=2 Preg=1 Bcascreg=2 all other reg=0 use_mult=none use_SIMD=four12 use_patdet=no_patdet	H	D	C	no	no	no	no	yes	no	yes
9	Areg=2 Breg=2 Preg=1 Acascreg=1 Bcascreg=1 all other reg=0 use_mult=mult use_SIMD=one 48 use_patdet=no_patdet	H	D	C	no	no	no	no	yes	no	no
10	Areg=2 Breg=2 Preg=1 Acascreg=2 Bcascreg=2 all other reg=0 use_mult=mult use_SIMD=one 48 use_patdet=no_patdet	H	C	D	no	no	no	no	yes	no	no

BIST #	Pipeline registers and Attribute values	Signal Active Level	A & B Input Source		Test Modes Applied						
			Slice0	Slice1	Mult	Add	ALU	Pat Det	Casc	MACC Extend	SIMD /casc
11	Areg=2 Breg=2 Preg=1 Acascreg=1 Bcascreg=1 all other reg=0 use_mult=mult use_SIMD=one 48 use_patdet=no_patdet	H	C	D	no	no	no	no	yes	no	no

Table 4.9 BIST Sequences for Virtex-5 DSP

Test Mode	First 256 ccs	Second 256 ccs	Third 256 ccs	Fourth 256 ccs
Multiply (000)	P=A×B (5×3)	P=A×B (3×5)	P=A×B+C (5×3)	P=A:B+C (3×5)
Adder (001)	P=Z(C) P=X(P)+Y(C)	P=Y(C) P=X(P)+Z(C)	P=Z(C) P=Y(C)+Z(P)	P=Y(C) P=Y(C)+Z(ShiftP)
ALU (010)	P=X(A:B)∇ Z(C) (∇ = •, +, ⊕, etc.)			
Pattern Detect (010)	P=X(A:B) == Z(C) ('==' indicates comparison)			
Cascade (100)	P_1=A:B+Z(PC) P_0=Z(C)	P_1=A:B+Z(ShiftPC) P_0=Z(C)	P_1=Z(C) P_0=A:B+Z(PC)	P_1=Z(C) P_0=A:B+Z(ShiftPC)
MACC extend (101)	P_0=Z(P)+Y(A×B)+X(A×B) P_1=Z(P) (P=PC+MULTSIGNIN+CARRYCASCOUT)		P_0=Z(P) (P=PC+MULTSIGNIN+CARRYCASCOUT) P_1= Z(P)+Y(A×B)+X(A×B)	
SIMD/cascade (110)	P_0=Z(C) P_1=Z(PC)+X(A:B)		P_1=Z(PC)+X(A:B) P_0=Z(C)	

Table 4.10 Variables in Table 4.8

Variables	Explanation
A, B & C	Input ports to the DSP
P	Output port of the DSP
Z(C),Y(C)	C input fed to the adder through the Z and Y multiplexers respectively
Z(P), X(P)	P output of the DSP fed back to the adder through the Z and X multiplexers respectively
Z(ShiftP)	Indicates P output shifted by 17 bits and fed back through the Z multiplexer
PC	PC indicates the cascaded P output from the DSP slice below
Z(PC)	Indicates the cascaded P output from the DSP slice below fed through the Z multiplexer
Z(ShiftPC)	Indicates the cascaded P output from the DSP slice below shifted by 17 bits and fed through the Z multiplexer
P_1	Output P corresponding to slice 1
P_0	Output P corresponding to slice 0

4.4 BIST Generation

The TPG written in VHDL is synthesized for an LX30T device and the location of the TPG in the chip array is determined. The TPG location for all other devices is offset with respect to the location of the TPG in the LX30T device based on the number of rows and columns in individual devices. The synthesized TPG in NCD format is converted to XDL format. Two programs written in C generate the DSP BIST configurations for any size or family of Virtex-5 devices. The *V5DSPBIST.exe* program extracts the TPG from the synthesized TPG file in XDL format (using the same *TPGXDLEXT.exe* program developed for Virtex-4) and instantiates the TPGs, the ORAs and the DSPs under test and generates a DSP BIST template in XDL format. The DSP BIST template in XDL format is converted to NCD format for routing the BIST architecture. The routed DSP BIST template is converted back to XDL format to be used by the modification program, *V5DSPMOD.exe*. This modification program, written in C, can be used to modify the routed DSP BIST template to generate the BIST configurations in XDL format. These BIST configuration files are converted back to NCD format to generate the download configuration bit files. The NCD files of the DSP BIST template and the BIST configuration can be viewed in FPGA Editor.

4.5 Timing Analysis of BIST

Timing analysis was performed on some of the Virtex-5 devices to determine the slowest BIST configuration. From the timing analysis it is observed that the position of the TPGs and the ORAs has an impact on the BIST clock frequency. Figure 4.13 illustrates the maximum BIST clock frequency for all Virtex-5 devices based on the position of the TPGs and the ORAs when the full array of the chip was tested. Figure 4.13 shows that the clock frequency is higher when the TPGs are placed at the middle of the chip array compared to positioning the TPGs at the

bottom of the chip array and for all devices except the LXT155, LXT330, SXT50 and SXT95 devices, where the clock frequency is higher when the ORAs are placed to the right of the DSPs compared to the position of the ORAs to the left of the DSPs. The lower clock frequency when the ORAs are placed on the left of the DSP is because of fewer routing resources between the TPGs and the DSPs. Moving the ORAs to the right of the DSPs reduces the routing congestion in all devices except the LXT155, LXT220, SXT50 and the SXT95 devices. According to these results, the BIST clock frequency is higher when the TPGs are placed at the middle of the array and the ORAs are placed to the right of the DSPs. Hence, the TPGs are placed at the middle of the array and the ORAs are placed to the right of the DSPs. For the bigger devices like the LXT110, LXT155, LXT330, SXT50 and SXT95 devices, the BIST clock frequency is less than 50 MHz. For these devices sub-array testing is done, where each half of the array is tested separately.

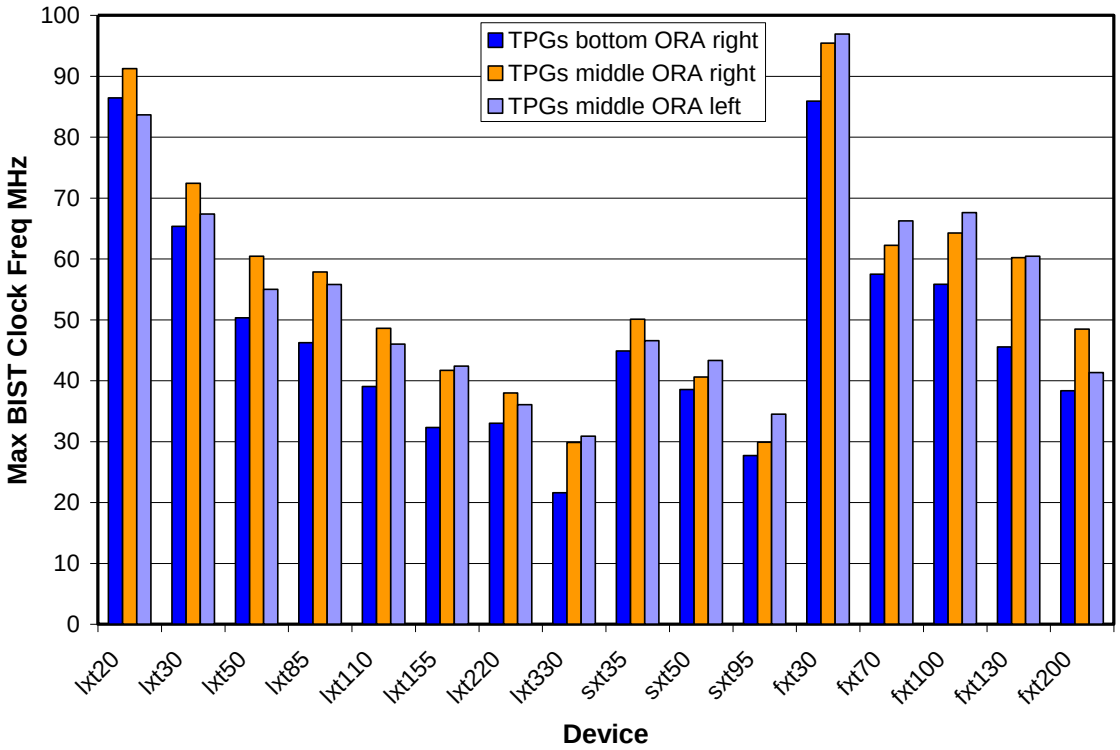


Figure 4.13 Clock Frequency Based on the Position of the TPGs and the ORAs

Figure 4.14 shows the impact of the position of the TPGs, based on sub-array that is being tested, on the BIST clock frequency. In Figure 4.14 bottom half and top half indicate the bottom half and top half arrays of the chip. Figure 4.14 shows that moving the TPGs to the bottom of the array while testing the bottom-half of the array improves the BIST clock frequency and raises the BIST clock frequency to over 50MHz for the LXT110, LXT155, LXT220 and SXT50 devices. However, the BIST clock frequency for at least one of the sub-arrays is still less than 50MHz for the LXT330, SXT50 and SXT95 devices. So for these devices each quarter of the array can be tested separately to improve the BIST clock frequency. The position of the TPG is at the middle of the array while testing all quarters of the array. Figure 4.15 shows the BIST clock frequency for each quarter of the array for the LXT330, SXT50 and the SXT95 devices.

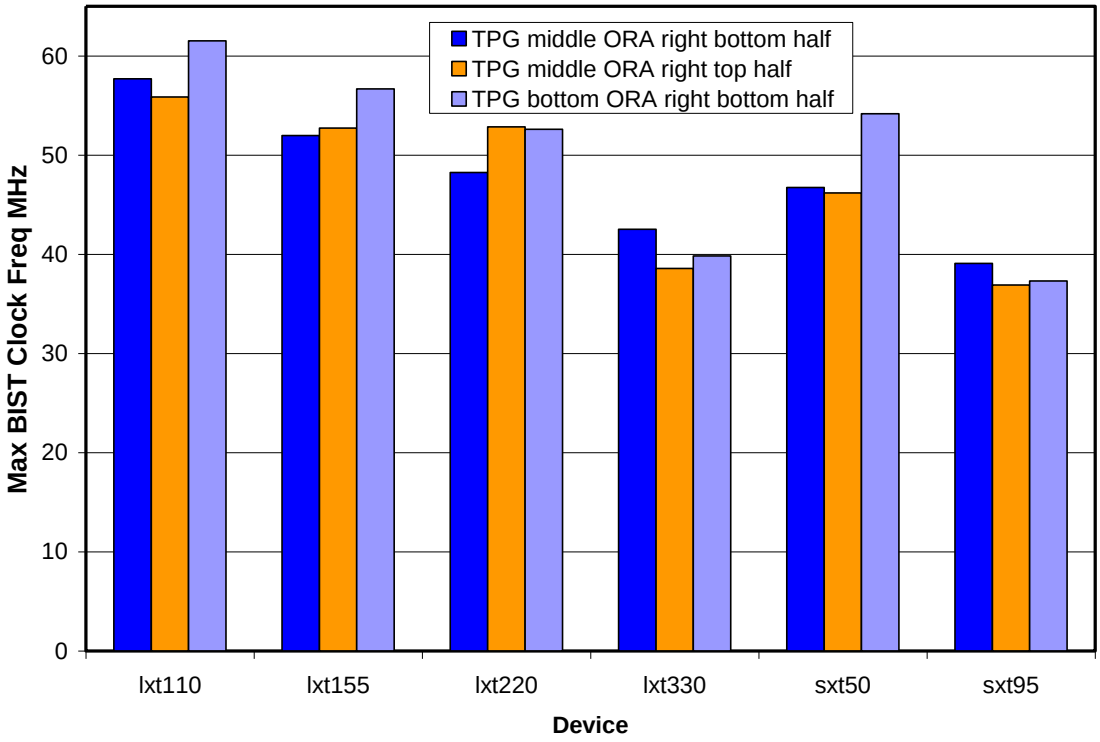


Figure 4.14 Clock Frequency for the Sub-Arrays Based on the Position of the TPGs

Figure 4.15 shows that the BIST clock frequency is higher for the quarter-arrays that have the TPGs located close to the arrays. Figure 4.16 shows the position of the TPGs for each of

the quarter arrays for an LXT330 device. For the LXT330 and the SXT50 devices, the clock frequency is well above 50MHz when the TPGs are located close to the DSPs under test. So for these two devices, two sets of two TPGs can be used, where one set of TPGs is placed at the middle of the array to test the middle two quarter-arrays and the other set of TPGs is placed at the top of the array to test the top-most quarter-array or at the bottom of the array to test the bottom-most quarter of the array.

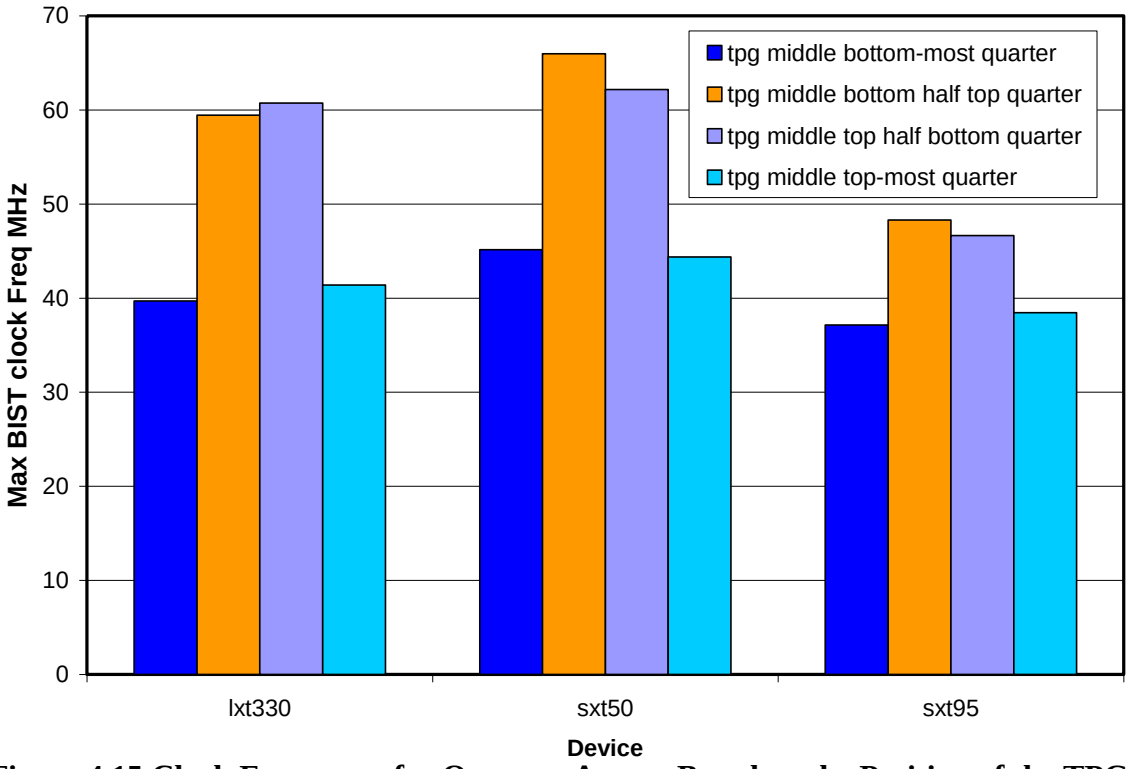
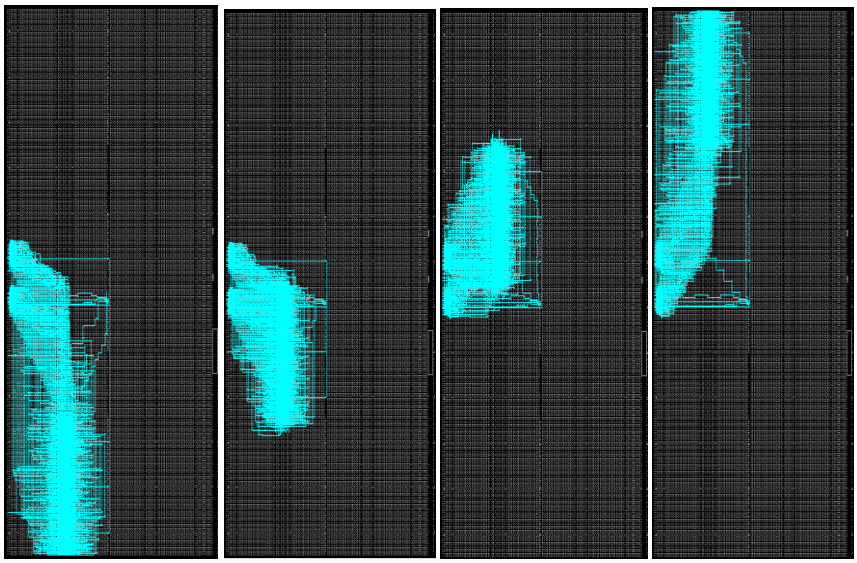


Figure 4.15 Clock Frequency for Quarter -Arrays Based on the Position of the TPGs

Figure 4.17 shows the BIST clock frequencies for the SXT95 devices based on different locations of the TPGs and the ORAs in the chip array for sub-arrays, quarter-arrays and quadrants of the array. For example, in Figure 4.17, the phrase “tpg bottom ORA right” indicates that the TPG is located at the bottom of the array and the ORAs are located on the right side of the DSPs. From Figure 4.17 it is seen that in all the cases the frequency is below 50MHz. So for the SXT95 device each quadrant of the quarter-array can be tested separately and four sets of two

TPGs can be used, where two sets of TPGs will be placed at the middle of the array with one set on the left of the chip array and one set on the right of the chip array. The remaining two sets will be placed on either side of the chip at the bottom/top of the array based on the array that is being tested.



a) bottomost quarter **b) bottom-half top quarter** **c) top-half bottom quarter** **d) topmost quarter**

Figure 4.16 Quarter Arrays for LXT330 device

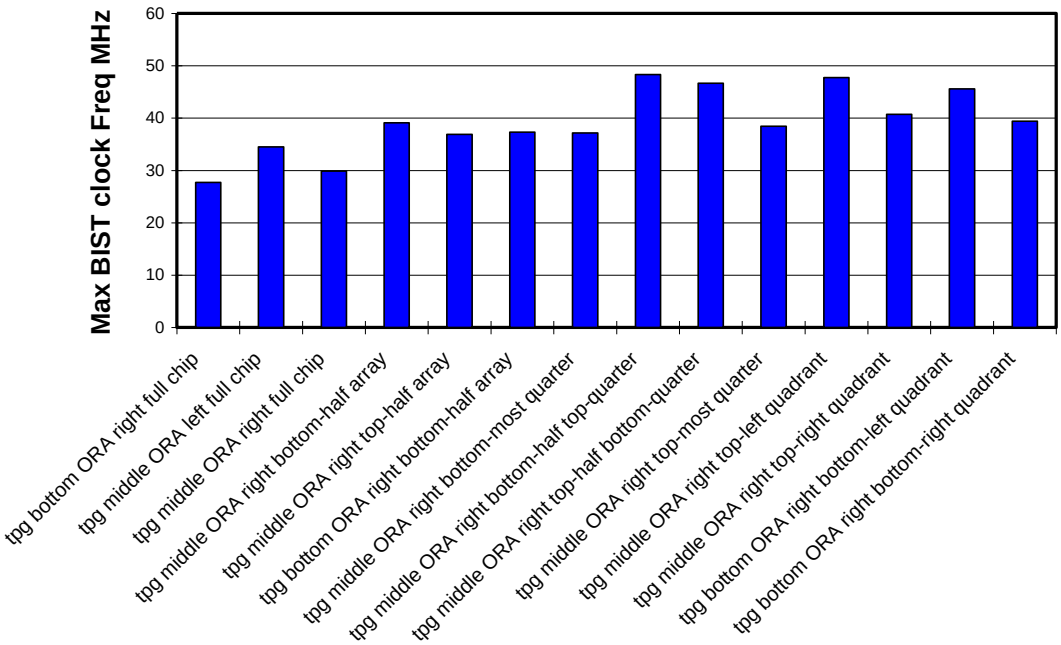


Figure 4.17 Clock Frequencies for all Arrays of the SXT95 Device Based on Positions of the TPGs and ORAs

4.6 Fault Inject Analysis and Fault Coverage

Fault-inject analysis was done by injecting a total of 604 faults of which 564 faults were detected. Of the 40 undetected faults, 30 faults are associated with the test circuitry that only the manufacturer has access to. These faults are not a concern since they do not affect the functioning of the DSP. Six of the remaining ten faults are faults associated with non-functional configuration bits that are not mentioned in the data sheet and hence are not of concern. The remaining four undetected faults are associated with the PATDET and NO_PATDET configuration bits of the USE_PATTERN_DETECT attribute. The PATDET and NO_PATDET configuration bits select between combinational paths of different speed. Hence these faults will be detected when tested at speed. These faults are not detected because the boundary scan interface is used for fault inject analysis, which is slow. Figure 4.18 illustrates the individual and cumulative fault coverage for the 20 test sequences. The bar graph in the figure illustrates individual fault coverage and the line graph illustrates the cumulative fault coverage. Cumulative fault coverage of 99.3% is achieved.

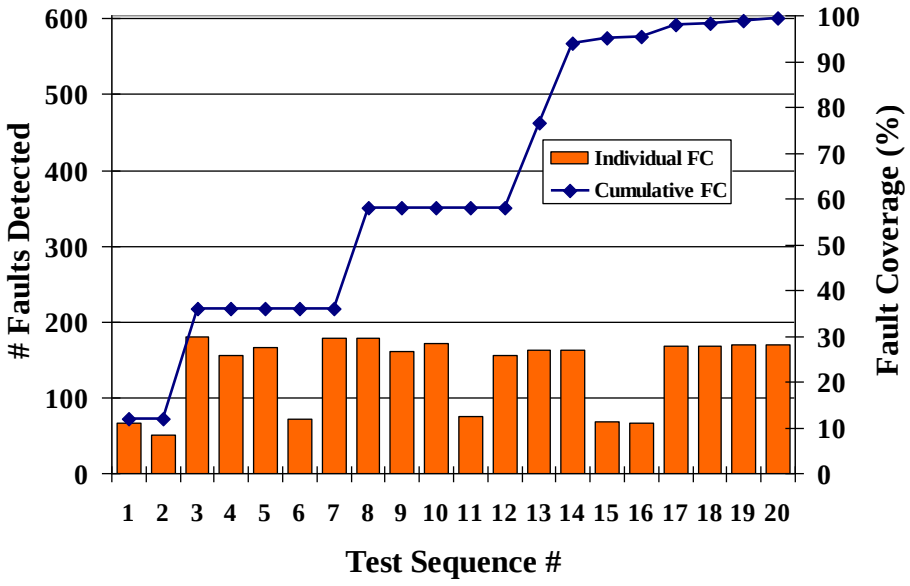


Figure 4.18 Fault Inject Results for Virtex-5 DSP BIST

4.7 Summary

A minimum set of BIST configurations was developed to test the DSP cores in Virtex-5 FPGAs. Fault detection capabilities and fault diagnosis were verified by injecting faults into the configuration memory bits controlling the DSP cores in an LXT-30 device. Fault coverage of 99.3% is achieved for the faults injected in the configuration memory of the DSP. The functional fault coverage, as determined by fault simulations, is much higher. Fault coverage for the faults injected in the configuration memory of the DSP can be improved to 100% by adding more BIST configurations if desired. Since these undetected faults are in nonfunctional modes of operation, the value of additional BIST configurations is questionable. Timing analysis was done to determine the maximum BIST clock frequency. Based on the timing analysis results the position of the TPGs and the ORAs in the chip array were changed to improve the maximum BIST clock frequency. For larger devices sub-array testing (where only half the chip is tested at a time) and quarter-array testing (where only one quarter of the chip is tested at a time) is done.

Chapter 5

Summary and Conclusion

This chapter highlights and summarizes the work presented in the thesis. Section 5.1 summarizes DSP BIST for Virtex-4 devices. Section 5.2 summarizes DSP BIST for Virtex-5 devices. Section 5.3 describes application of DSP BIST for DSPs in other FPGAs.

5.1 Summary of Virtex-4 DSP BIST

DSP BIST for Virtex-4 FPGAs presented in this thesis was developed by writing 8-bit and 48-bit models of the architectures of the logic cores in the DSP: multiplier and adder in ASL. Fault simulations were carried out using AUSIM to determine the correct set of test patterns and configurations for completely testing the cores for single stuck-at and bridging faults. Three test sequences were developed for the three test modes: multiplier, adder and cascade modes of operation. Five BIST configurations were developed and the FPGA is repeatedly reconfigured to run the three test sequences. Seven test sequences are run in five downloads to the FPGA to completely test the FPGA in all its functional modes of operations. Fault detection for the DSP BIST was evaluated by injecting faults in the configuration memory bits of the DSP in Virtex-4 FX12 device. Of the 154 faults injected, 150 faults were detected giving a fault coverage of 97.4%. The four undetected faults are in non-functional modes of the DSP but can be detected by adding additional BIST configurations. Timing analysis was done on all Virtex-4 devices to determine the maximum BIST clock frequency for each device. Based on this analysis the position of the TPGs for achieving a BIST clock frequency of at least 50MHz was determined.

Sub-array testing in larger Virtex-4 devices minimizes the power dissipation caused by concurrently testing a large number of DSPs in the device.

5.2 Summary of Virtex-5 DSP BIST

Since the architecture of DSPs in Virtex-5 FPGAs is similar to the architecture of DSPs in Virtex-4 FPGAs, the test algorithms used to test DSPs in Virtex-4 FPGAs are also applied to test the multiplier and the adder cores in DSPs of Virtex-5 FPGAs. Test patterns and BIST configurations for the additional circuits in Virtex-5 FPGAs were developed by writing ASL models of the circuits and doing fault simulation for these models in AUSIM. Five test sequences were developed for the five test modes: multiplier, adder, logic, pattern detector and cascade modes of operation. Eleven BIST configurations were developed and the FPGA is repeatedly reconfigured to completely test the FPGA in all its functional modes of operation. 20 tests are run in eleven downloads to the FPGA. Fault coverage for the DSP BIST developed was evaluated by injecting faults in the configuration memory bits of the DSP in Virtex-5 LXT30 device. Timing analysis was done on all Virtex-5 devices to determine the maximum BIST clock frequency for each device. Based on this analysis the position of the TPGs and the ORAs for achieving a BIST clock frequency of at least 50MHz was determined. Sub-array, quadrant and sub-quadrant testing in larger Virtex-5 devices minimizes the power dissipation caused by concurrently testing a large number of DSPs in the device.

5.3 Application to Other FPGAs and Architectures

The DSP BIST for Virtex-4 and Virtex-5 FPGAs presented in this thesis can be extended and applied to test DSPs in other FPGAs like Spartan-3A, Spartan-6 and Virtex-6 FPGAs. The architectures of DSPs in Spartan-3A and Spartan-6 FPGAs is similar to the architecture of DSPs in Virtex-4 FPGAs [26] [27]. However, the DSPs in Spartan-3A and Spartan-6 FPGAs have a

pre-adder stage in addition to the circuits present in DSPs of Virtex-4 FPGAs, but the test algorithms used to test the adder and the multiplier in Virtex-4 FPGAs can also be used to test the adder and multiplier cores in Spartan-3A and Spartan-6 FPGAs. The architecture of DSPs in Virtex-6 FPGAs is similar to the architecture of DSPs in Virtex-5 FPGAs [28], but the test algorithms used to test the logic circuits in Virtex-5 DSPs can also be used to test the logic circuits in Virtex-6 FPGAs.

DSP BIST for Virtex-4 devices is explained in [30]. [31] gives a detailed explanation of adder BIST and [32] gives a detailed explanation of multiplier BIST.

References

[1] L. T. Wang, C. Stroud and N. A. Touba, "*System On Chip Test Architectures*," Morgan Kaufmann Publishers, 2007.

[2] K. M. Thompson, "Intel and the Myths of Test," *IEEE Design & Test of Computers,* vol. 13, pp. 79-81, 1996.

[3] S. Hauck, "The roles of FPGAs in Reprogrammable Systems," *Proc. of the IEEE,* vol 86, pp. 615-638, 1998.

[4] M. B. Tahoori and S. Mitra, "Test Compression for FPGAs," *Proc. IEEE Intl. Test Conf.,* pp. 1-9, 2006.

[5] E. Chmelar, "Minimizing the number of test configurations for FPGAs," *IEEE/ACM Intl. Conf. on Computer Aided Design,* pp. 899-902, 2004.

[6] Xilinx Inc., "Virtex-4 FPGA User Guide," UG070 v2.5, 2008.

[7] Xilinx Inc., "Virtex-5 FPGA User Guide," UG190 v4.2, 2008.

[8] W. K. Huang, F.J Meyer and F. Lombardi, "Array-Based Testing of FPGAs: Architecture and Complexity," *Proc. IEEE Intl. Conf. on Innovative Systems in Silicon*, pp. 249-258, 1996.

[9] Xilinx Inc., "XtremeDSP for Virtex-4 FPGAs," User Guide UG073 (v2.7), Xilinx Inc., 2008.

[10] Xilinx Inc., "Virtex-5 XtremeDSP Design Considerations," User Guide UG193 (v3.1), Xilinx Inc., 2008.

[11] B. Dufort and G. H Chapman, "Test Vehicle for a Wafer-Scale Field Programmable Gate Array," *Proc. IEEE Intl. Conf. on Wafer Scale Integration,* pp.33-42, 1995.

[12] A. Orailoglu, "Microarchitectural Synthesis for Rapid BIST Testing," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* vol. 16, no. 6, pp. 573-586, 1997.

[13] M. Abramovici, C. Stroud and M. Emmert, "Using Embedded FPGAs for SOC Yield Improvement," *Proc. ACM/IEEE Design Automation Conference,* pp. 713-724, 2002.

[14] S. Adham and S. Gupta, "DP-BIST: A Built-In Self-Test for DSP Data Paths - A Low Overhead and High Fault Coverage Technique," *Proc. IEEE Asian Test Symp*., pp. 205-212, 1996.

[15] H. Al-Asaad, J. Hayes, and B. Murray, "Scalable Test Generators for High-Speed Datapath Circuits," *J. Electronic Testing: Theory and Applications,* vol. no. 12, pp. 111-125, 1998.

[16] D. Gizopoulos, A. Paschalis and Y. Zorian, "Effective Built-In Self-Test for Booth Multipliers," *IEEE Design & Test of Computers,* vol. 15, no. 3, pp. 105-111, 1998.

[17] A. Paschalis, N. Kranitis, M. Psarakis, D. Gizopoulus and Y. Zorian, "An Effective BIST Architecture for Fast Multiplier Cores", *Proc. Design, Automation and Test in Europe Conf.*, pp. 117-121, 1999

[18] D. Bakalis, E. Kalligeros, D. Nikolos, H. Vergos and G. Alexiou, "Low Power BIST for Wallace Tree-based Fast Multipliers," *Proc. IEEE Int. Symp. on Quality of Electronic Design*, pp. 433-438, 2000.

[19] S. Kajihara and T. Sasao, "On the Adders with Minimum Tests," *Proc. IEEE VLSI Test Symp*., pp. 10-15, 1997.

[20] A. Sarvi and J. Fan, "Automated BIST-Based Diagnostic Solution for SOPC," *Proc. Int. Conf. on Design & Test of Integrated Systems in Nanoscale Technology*, pp. 263-267, 2006.

[21] C. Stroud and S. Garimella, "Built-In Self-Test and Diagnosis of Multiple Embedded Cores in SOCs," *Proc. Intl Conf. on Embedded Systems and Applications*, pp. 130-136, 2005.

[22] "Using Embedded Multipliers in Spartan-3 FPGAs," Application note XAPP467 (v1.1), Xilinx, Inc., 2003.

[23] C. Nagendra, M-J Irwin and R-M Owens, "Area-Time-Power Tradeoffs in Parallel Adders," *IEEE Trans. on Circuits and Systems II*, vol 43, no. 10, pp. 689-702, 1996.

[24] B.F Dutton and C. E Stroud, "Built-In Self-Test of Configurable Logic Blocks in Virtex-5 FPGAs," *Proc. IEEE Southeastern Symp. on System Theory*, pp. 230-234, 2009.

[25] P.H.W Leong, "Recent Trend in FPGA Architectures and Applications," *Proc. IEEE International Symp. on Electronic Design, Test and Applications*, pp. 137-141, 2008

[26] Xilinx Inc., "XtremeDSP DSP48A for Spartan-3A DSP FPGAs," User Guide UG431 (v1.3), Xilinx Inc., 2008.

[27] Xilinx Inc., "Spartan-6 Family Overview," DS160 v1.0, 2009.

[28] Xilinx Inc., "Virtex-6 Family Ovreview," DS150 v1.0, 2009.

[29] B.F. Dutton, M. Ali, J. Sunwoo and C. E. Stroud, "Embedded Processor Based Fault Injection and SEU Emulation for FPGAs," *Proc. Intl Conf. on Embedded Systems and Applications*, pp. 183-189, 2009

[30] M. D. Pulukuri and C. E. Stroud, "Built-In Self-Test of Digital Signal Processors in Virtex-4 FPGAs," *Proc. IEEE Southeastern Symp. On System Theory*, pp. 34-38, 2009

[31] M. D. Pulukuri and C. E. Stroud, "On Built-In Self-Test for Adders," *J. Electronic Testing: Theory and Applications*, vol. 25 no. 6 pp. 343-346, DOI 10.1007/s10836-009-5114-6, 2009.

[32] M. D. Pulukuri, G. J. Starr and C. E. Stroud, "On Built-In Self-Test for Multipliers," *Proc. IEEE Southeast Regional Conf.*, pp. 25-28, 2010.