

Research article

Available online www.ijsrr.org

International Journal of Scientific Research and Reviews

Implementation of Testing Methods for VLSI Circuits

A .Ooha¹ and V. Leela Rani²

¹M.Tech. Student, Department of Electronics and Communication Engineering, G.V.P. College of Engineering (A), Visakhapatnam, AP-India, Email: <u>a.ooha9@gmail.com</u>.

²Associate professor, Department of Electronics and Communication Engineering, G.V.P college of Engineering (A), Visakhapatnam, AP-India, Email: lee_rani@gvpce.ac.in

ABSTRACT:

An ASIC chip designed may not meet the functionality requirements. There may be lot of conditions which may cause damage to the designed circuit and effect its functionality. Those conditions include processing faults (such as missing contact windows, oxide breakdown), material defects (such as cracks, crystal imperfections), packaging failures etc. These problems may cause faults in the digital logic circuits and effects circuit functionality. Therefore, the circuit must be tested in order to know whether it is working properly or not.

Several testing methods and algorithms have been proposed such as D-algorithm, PODEM algorithm, Built-in-self-test etc in literature. In this paper, Built-in-logic-block-observer (BILBO) method of testing is implemented. A test pattern compaction method, STAR-EDT is combined with BILBO to derive minimum number of test data for detecting all faults that are injected into the circuit. The combined method called BILBO with STAR-EDT methodology is proposed. Existing and proposed methods are implemented and applied to various test circuits. After that, a Scheduling method is applied to BILBO with STAR-EDT method for all test circuits at a time. The proposed method requires less number of test patterns than the number of patterns required in existing BILBO methodology. Scheduling concept applied to proposed method results in reduced timing requirements for test procedures.

KEYWORDS: BILBO, STAR-EDT, Scheduling, Test data, Faults.

*Corresponding author

Ms. A. Ooha

M.Tech. student,

G.V.P. College of Engineering (A)

Visakhapatnam, AP, India.

Email: a.ooha9@gmail.com.

INTRODUCTION:

An ASIC chip designed may not meet the functionality requirements. Several testing methods and algorithms have been proposed such as D-algorithm, PODEM algorithm, Built-in-self-test etc to test the designs in order to define whether it is faulty or fault-free. The D-algorithm and PODEM (Path-Oriented-Decision-Making) algorithm are external methods of test data application. Whereas, BIST (Built-In-Self-Test) architecture is designed on-chip and the entire testing process is done on chip. BIST is categorized into several types namely Logic BIST (LBIST), Memory BIST (MBIST), Scan-based BIST, Programmable BIST, BILBO etc^{3,4,5,8}. In this paper, BILBO (Built-In-Logic-Block-Observer) method of testing is considered. BILBO needs more number of patterns to detect faults. A test data compression methodology STAR-EDT is used which derives minimum number of test patterns to detect more number of faults^{1,6}. Testing multiple test circuits requires more time. Scheduling methodology is applied to test reduce the time consumption and hence, test circuits can be tested in a parallel way.²

BILBO METHODOLOGY:

BILBO (Built-In-Logic-Block-Observer) is one of the methodologies of BIST. It uses a register that operates in different modes based on its control inputs. The structure of BILBO register consists of flip-flops and a combinational logic comprising of XOR, NAND and NOR gates between every two flip-flops. Two control inputs are present in order to decide the mode of operation. The structure of BILBO register is shown in Figure 1.

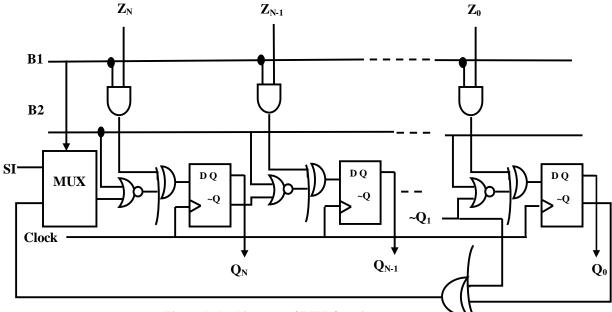


Figure 1: Architecture of BILBO register

MODES OF BILBO REGISTER:

The inputs B1 and B2 are control inputs to BILBO register which decides the modes of operation of BILBO register.

If B1=B2=0, then BILBO register will be in RESET mode.

If B1=0, B2=1, then BILBO register will be in Scan mode.

If B1=1, B2=0, then BILBO register will be in PRPG or MISR mode.

If B1=B2=1, then BILBO register will be in Register mode.

The BILBO register in PRPG mode generates test patterns in order to give them as inputs to the test circuit. The response of the test circuit is given to BILBO register in MISR mode as input to generate a signature. This signature of test circuit is compared with reference signature. If both signatures match with each other, then the circuit is said to be fault-free.

FAULTS INJECTED INTO TEST CIRCUITS:

The test circuits considered in this paper are 1-bit full-adder, 32-bit ALU, 32-bit ripple carry adder and 32-bit magnitude comparator. In 1-bit full-adder, 2 faults (stuck-at-0 and stuck-at-1) are injected. In 32-bit ALU, 10 faults (8 s-a-1 and 2 s-a-0) are injected and in 32-bit magnitude comparator, 7 faults (4 s-a-1 and 3 s-a-0) are injected. In 32-bit ripple carry adder, 5 faults (2 s-a-1 and 3 s-a-0) are injected during test process.

PROPOSED METHOD:

BILBO with STAR-EDT methodology is proposed in this paper. STAR-EDT is a test data compression scheme which derives minimum number of test data that is able to detect all faults in the test circuit. STAR-EDT consists of a ring generator that produces test patterns that are applied to the test circuit. As BILBO register has an advantage of high fault coverage, ring generator is replaced with BILBO register. The responses of the test circuit helps to recognize parent pattern (a pattern that is able to detect at least one fault in the fault-list). By applying that parent pattern to a phase shifter, children patterns can be derived, which together forms a test cluster. The derived test clusters are again fault simulated in order to get minimum number of test clusters. The flow of proposed method implemented is shown in Figure 2.

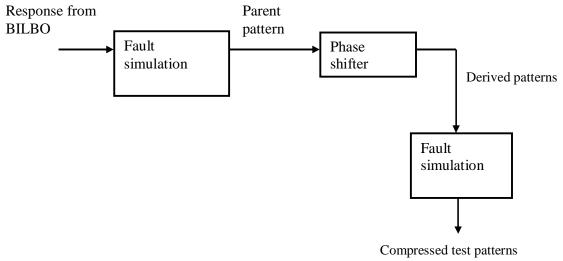


Figure 2: Block diagram of proposed met

SCHEDULING METHODOLOGY:

Scheduling is applied to above test procedure in order to test four test circuits at a time. The scheduling method considered in this paper consists of three sessions. In first session, ripple carry adder and 1-bit full adder are tested. In second session, magnitude comparator and in third session, 32-bit ALU are tested respectively. The structure of scheduling methodology is shown in Figure 3.

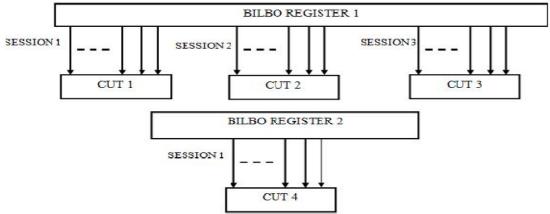


Figure 3: Architecture of Scheduling methodology

RESULTS AND DISCUSSIONS:

The simulation results of above methodologies implemented on all test circuits are shown below. The proposed method derived less number of test patterns for all test circuits compared with number of patterns required in BILBO methodology.

BILBO METHODOLOGY:

Figure 4 presents simulation result of BILBO methodology for 1-bit full adder. In the simulation, the signals "sig_faulty" and "sig_nofault" represents the signatures of both faulty and

fault-free test-circuits respectively. The signal "fault" compares both signatures and it is enabled if both signatures differ with each other indicating that the circuit is faulty.

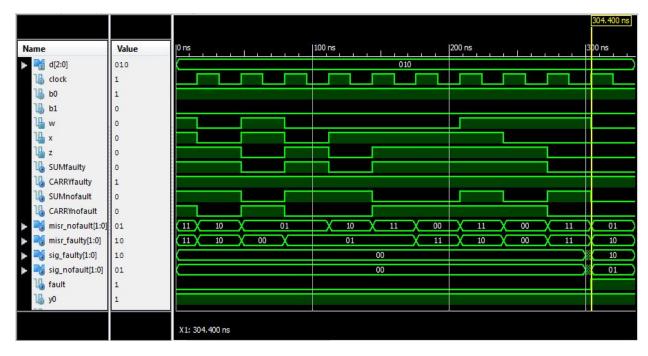


Figure 4: Simulation result of BILBO methodology for 1-bit full adder

Console				
	nature 0x7708f090)			
This is a Full ver	sion of ISim.			
# run 1000 ns				
	ng circuit initialization prod	cess.		
	nitialization process.			
circuit is faulty! circuit is faulty!				
ISim>				
Console	Compilation Log	Breakpoints	Find in Files Results	Search Results

Figure 5: Console window of BILBO methodology for 1-bit full adder

Figure 6 to Figure 11 shows the simulation results of BILBO methodology for test circuits 32-bit ALU, 32-bit Ripple carry adder and 32-bit Magnitude comparator respectively.

W.		520.000 ns	
Name	Value	460 ns 470 ns 480 ns 490 ns 500 ns 510 ns 520 ns	530
🕨 📷 d[63:0]	0011010101010	0011010101010010011001100110010101010101	
16 clock	1		
w[31:0]	1000110011001	1000110	
x[31:0]	111010101010101	1019101 <u>X1111111111111110110000100 X0101010101000001101111 X100000000</u>	
ALU_SEL1[1:0]	10	10 11 00 01 01 01 11 00 01 01 01 00 01 01	X OD X
ALU_nofault[31:0]	0111011101110	<u> (001_)011_)011_/100_)100_)111_/001_/111_/11</u>	X110.X
ALU_FAULTY[31:0]	0111011101110	011_011_011_011_000_000_0111_001_011_01	(110 Y
sig_faulty[31:0]	0100101001111	000000000000000000000000000000000000000	11100001
sig_nofault[31:0]	1111100011010	000000000000000000000000000000000000000	01011011
11 fault	1		
misr_faulty[31:0]	0100101001111	000 0011111000011011. X 101100001110 0000010000 X0111101110000010110110 X100000011000011101101 X01001001	11100001
▶ 式 misr_nofault[31:0]	1111100011010	00111001000001101001	01011011
final[31:0]			Yood Y
			Y

Figure 6: Simulation result of BILBO methodology for 32-bit ALU

Console		
ISim P.40xd (signature 0x7708f090) This is a Full version of ISim. Time resolution is 1 ps Simulator is doing circuit initialization process. Finished circuit initialization process. circuit is faulty! circuit is faulty! ISim>		
Console Compilation Log Breakpoints	Find in Files Results	s Search Results

Figure 7: Console window of BILBO methodology for 32-bit ALU

											522.747 r	15	
Name	Value	460 ns	n ľ	170 ns	480 ns	490	ns	500 ns	510 ns	520	s Lilii	530 r	ns
📡 📑 d[63:0]	01001110010101			(01001110010101	011101010101	01010101010101	1100110011001100	101010101011				
16 clock	1			1								1	
16 во	1												
1 <mark>6</mark> b1	0												
w [31:0]	11001100111110												
x[31:0]	10011001100100												
rcanofaultout[31:0]	11011101111001	110111 X1	100110010	110100101001:	1 1000 1000 1	0000101001	0110 11001	100110101100000	01X110111	0111100100:	000100	1100110	0011
rcafaultout[31:0]	11011101111001	110111X1	100110010	1101101010101	1 1000 1000 1	0000101001	0110	100110101100000	001X110111	0111100100:	000 100	1100110	0011.
misrfaulty[31:0]	10110000100101	00001101000	01101110	101011010	1010011100001	101001	011001110101	10001 0101010	1000011101101	101 101	0000100	101100100	0101.
misrnofault[31:0]	11011011111111	01011011111	10111101	. 000001100	00111101111010		000011101111	01101 0110001	1110111011000	011 110	1011111	11111111110	0010
sig_faulty[31:0]	10110000100101				00000000000	000000000000000000000000000000000000000	00000000000			X 101	0000100	101100100	0101
sig_nofault[31:0]	11011011111111				000000000000000000000000000000000000000	000000000000000000000000000000000000000	00000000000			110	1011111	11111111110	0010
1 fault	1												
		X1: 522.747	ns										

Figure 8: Simulation result of BILBO methodology for 32-bit Ripple carry adder

Console	
ISim P.40xd (signature 0x7708f090)	
This is a Full version of ISim.	
# run 1000 ns	
Simulator is doing circuit initialization process.	
Finished circuit initialization process.	
circuit is faulty!	
circuit is faulty!	
ISim>	
Console Compilation Log Breakpo	points 🦝 Find in Files Results 🖬 Search Results

Figure 9: Console window of BILBO methodology for 32-bit Ripple carry adder

				Wit											79	92.770	ns	
Name	Value		500 ns		550 ns		600 ns		650 ns		700 n		750 n	S		800 n	S	. 85
b 📑 d[63:0]	0011010101010101				001	10101010	1010011	001100110	00111010	10101010101	010101	110101010	011100					
1 clock	1	ΠП	ΓП	П						ЪГШ		п	ЛГ	П		Г	Г	
116 во	1																	
116 b1	0																	
🗓 si	0																	
w [31:0]	00010101010101		01	01)01)	00)00		(11)(01		10	. X01	.)00)(1	0 11	X01X	00	10	(11))1X
x[31:0]	11011001101011		10X11X	01)(00)00)	10 (1)	X 11	01)00		01	X01	. 10	1 10	X11X	11	01	X10X	01X
COMPfaulty[2:0]	010	010	110 010	1	10	0	0	110	X	010 11	αX	010	110	χ	010		110	010
COMPnofault[2:0]	010		010	10	0	0	.0	100	X010	X 100	X	010 X	100	X 010			00 X	010
misr_faulty[2:0]	101	100 000	010 111	101	000 1	10 101	000 0	10 011	111 (1	101		100		000 110)1	01 🗙 0	00 110	101
😽 misr_nofault[2:0]	111	111 01	1 111 10	010	101 1	10 001	010 0	11 101	010 0	011 101	010	011	X001 X0	010 010)C	111	(01	1/111
🗓 fault	1																	
sig_faulty[2:0]	101						00	0						X 110)(1	01 🗙 0	00 110)(101)
sig_nofault[2:0]	111						00	0						010)C	111	(011	1/111
		X1: 792.7	70 ns															

Figure 10: Simulation result of BILBO methodology for 32-bit magnitude comparator

Console
This is a Full version of ISim.
run 1000 ns
Simulator is doing circuit initialization process.
Finished circuit initialization process.
circuit is faulty!
ISim>
🖀 Console 🔝 Compilation Log 🔎 Breakpoints 🕅 Find in Files Results 🖬 Search Results

Figure 11: Console window of BILBO methodology for 32-bit magnitude comparator

PROPOSED METHOD:

Figure 12 to 19 presents the simulation results of all test circuits when BILBO with STAR-EDT methodology is applied. In the simulation result, signals 'I1', 'I2', 'I3' and 'I4' represents the derived cluster of patterns. The signal 'numberoffaultsdetected' indicates the number of faults detected by that particular test cluster.

			80.000 ns
Name	Value	0 ns	100 ns 200 ns 300 ns
🕨 📷 d[2:0]	010		010
16 clock	1		
l <mark>l₀</mark> w	0		
Ц _о х	0		
l <mark>la</mark> z	1		
numberoffaultsdetected[1:	1	1 (1)(2)(1)	
퉪 faulta1	0		
堝 faulta2	1		
1 ADDER_SUMnofault	1		
U ADDER_CARRYnofault	0		
1 ADDER_SUMfaulty	1	A	
U ADDER_CARRYfaulty	1		
😽 final[1:0]	10	00 X 10 X 00	
i 1[2:0]	100	(000 X 100 X 000	<u>100 010 110 000 101 001 000</u>
12[2:0]	000	(000 X 000 X 000	000 110 010 000 001 101 100
😽 B[2:0]	000	000 000	000 110 100 000 001 011 010
N 14[2:0]	000	(000 X 000	000 101 001 000 010 110 100
30			
		X1: 80.000 ns	

Figure 12: Simulation result of proposed method for 1-bit full adder

Cons	ole								
	80 T	he nu	mber of faults dete	ected b	y this cluster	s:1			
	96 T	he nu	mber of faults dete	ected b	y this cluster	s:2			
	104	The n	umber of faults det	ected l	by this duster	is:1			
	112	The n	umber of faults det	ected I	by this cluster	is:0			
	120	The n	umber of faults det	ected l	by this cluster	is:2			
	136	The n	umber of faults det	ected I	by this cluster	is:0			
	144	The n	umber of faults det	ected l	by this cluster	is:2			
	152	The n	umber of faults det	ected l	by this cluster	is:1			
	160	The n	umber of faults det	ected l	by this cluster	is:0			
	168	The n	umber of faults det	ected l	by this cluster	is:2			
	184	The n	umber of faults det	ected	hy this cluster	is 1			
2-	Console		Compilation Log	•	Breakpoints	M	Find in Files Results	-	Search Results

Figure 13: Console window of proposed method for 1-bit full adder

lame	Value	890 ns		895	ns	900 ns		905 ns		910 ns	i a a i	915 r	IS	920 ns			925 n
d[63:0]	0011100101010				0011	1001010101110	1010	1010101010	101110	0110011	10011001010	1010	101100				
16 clock	0																
w[31:0]	0111000000000	100010	10101010111		011	1000000000001	000	0101110111	110			0001	101010101000	00011	0001000	01	
x[31:0]	0110011011100	1001100	00100110010		011	0011011100110	011	0 10000 1 100	001			0011	001110110011	01000	0100001	00	
ALU_SEL[1:0]	00	10	11		00	01		10)		1	00	X	01		10	X	11
ALU_OUTnofault[31	1101011011100	0111	000 100 100	0)	(1101011011)	0000100100	100	1111111)	000101	1011	001001110	LΧ	1111001101)	01110	11010	00	01010
ALU_OUTfaulty[31:0	1101011011101	0111	000 100 100	0)	(1101011011)	0000100101	100	1111111	000101	1011	001001110	X	1111001110)	01110	11010	X 00	0101
💐 final2[31:0]	0000000000000	0000	000000000	o)	(0000000000)	0000000001	000	0000000)	000000	0000	0000000000	ΞX	0000000011)	00000	00000	X 00	0000
🗓 or1	1															L	
11 [63:0]	1111000000000	0000	000010101	0)	(1111000000000	0010001010111	011	11001100110	0111001	1001	000011010	1010	1000100011)	00001	10101	00	0000
12[63:0]	1111000000000	0000	000010101	0)	(1111000000000	0010001010111	011	11001100110	0111001	1001	000011010	1010	1000100011)	00001	10101	X 00	0000
B[63:0]	1111000000000	0000	000010101	0)	(1111000000000	0010001010111	011	11001100110	0111001	1001	000011010	1010	1000100011)	00001	10101	X 00	0000
14[63:0]	1111000000000	0000	000010101	Q)	(1111000000000	0010001010111	011	11001100110	0111001	1001	000011010	1010	1000100011)	00001	10101	00	0000
ALU_OUTnofault1[3	0101011011100	₫XX	000000	XX	00000000	00000000	00	000000	0000	0000	000000	XX	0000000	XXX	XXXX	Σ	000
ALU_OUTfaulty1[31:	0101011011101	XXXX	XXXXXXX	XX	00000000	00000000	00	000000	0000	CXXX	000000	XX	XXXXXXXXX	XXX	XXXX	XX	(XX
💐 final[31:0]	0000000000000	XXXX	0000000	XX	00000000	00000000	00	000000	0000	0000	000000	XX	00000000	XXX	XXXX	XX	ŝ
numberoffaultsdet	10	5	X 7				0				(6			6	X	5

Figure 14: Simulation result of proposed method for 32-bit ALU

Console	
00	so the number of faults detected is 7
89	96 The number of faults detected is 10
91	12 The number of faults detected is 6
92	28 The number of faults detected is 5
94	44 The number of faults detected is 5
96	50 The number of faults detected is 8
97	76 The number of faults detected is 9
99	92 The number of faults detected is 6
ISim>	
Consol	le 🔲 Compilation Log 🔎 Breakpoints 🙀 Find in Files Results 📊 Search Results

Figure 15: Console window of proposed method for 32-bit ALU

1 (1) (1)	N.I.		160 pg		190 pg		n	100 pg		120 ns		140	20	1160 pg		190 pg	
Name	Value		60 ns	<u>i Li i i i i</u>	80 ns			100 ns						160 ns		180 ns	
d [63:0]	00110101010101					0011010	10101	010011001	1001100	1110101	010101010101	0101	110101010011	00			
1 clock	0				_												
▶ 📷 w[31:0]	00011011001000																
> 📷 x[31:0]	11000100010001	E.F.															
rcanofaultout[31:0]	11000100110011	011000100	0	01010010	1000	011001	01	1011110	111100	0110)	110001001	1)	1110110011	0100101111		0101110	10111
🔊 📑 rcafaultout[31:0]	11000100010010	11100001	ιφ <mark>Χ</mark> οοα	01010000	0000	011001	X11	1011110	111100	0110	110001000	1)	0110110011	1100101101		0101100	10111.
rcafinal[31:0]	00000000100001	10000011	10 100	00000010	1000	000000	X100	0000000	000000	0000)	000000001	0)	1000000000	1000000010		0000010	00000.
🗓 or1rca	1					-											
🔊 📑 numberoffaultsdeted	5	X3X 4		3	XIX	2	XΞ	2	X		5			2		3	
I1rca[63:0]	00001001010010	01001011	φ <mark>X</mark> 110	D1110010	1001	011100	X001	1001011	111000	0000)	000010010	1)	1111110111	0000011110		1101010	00000.
🔰 📑 I2rca[63:0]	00001001010010	10001011	10X110	01110010	0101	011100	X001	1001011	111000	0000)	000010010	1)	1111110111	0000011110		1101010	00000.
🗲 📑 Brca[63:0]	00001001010010	10001011	10 10	11110010	0011	011100	010	1001011	111000	0000)	000010010	1)	1111110111	0000011110		1101010	00000.
🔉 📑 I4rca[63:0]	00001001010010	10001011	10 10	11110010	0011	011100	01	.0001011	110100	0000)	000010010	1)	1111110111	0000011110	11	1101010	00000.
🔉 式 rcafinal11[31:0]	10000000100001	XX10000	000	00000010	X	000000	000	0000000	100000	0000)	100000001	0)	0000000000)	0000000010		0000010	10000.
rcanofaultout1[31:0]	01000100110011	XX00100	00	0	X	. 101	X11		011100		010001001	1)	0110110011	1100101111		0101110	00111.
rcafaultout1[31:0]	11000100010010	X	00	0	X		X11		111100		110001000		0110110011	1100101101		0101100	X 10111
1 rcafaulta1	1																
30														8			

Figure 16: Simulation result of proposed method for 32-bit Ripple carry adder

Console								
96	The number of faults detected by this cluster is:2							
104	The number of faults detected by this cluster is:2							
112	The number of faults detected by this cluster is: 3							
120	The number of faults detected by this cluster is: 3							
128	The number of faults detected by this cluster is: 5							
136	The number of faults detected by this cluster is:5							
144	The number of faults detected by this cluster is:2							
152	The number of faults detected by this cluster is:2							
160	The number of faults detected by this cluster is:2							
168	The number of faults detected by this cluster is:2							
175	The number of faulte detected by this duater is 2							
Console	Compilation Log 🛛 Breakpoints 🙀 Find in Files Results 📊 Search Results							

Figure 17: Console window of proposed method for 32-bit Ripple carry adder

											160.000 ns			
Name	Value	6	i0 ns	80 ns		100 ns	144.4	120 ns	ana kaa	140 ns	160 ns		180 ns	2
🕨 📷 d[63:0]	01001110010101			01001110	0101	010111010	1010101	0101010	1110011001	100110010101010101	11			
16 clock	0				L									
▶ 📷 w[31:0]	11001100111110													
🕨 📷 x[31:0]	10011001100100													
COMPnofault[2:0]	100	X	100	010	X	1	.00		X	010	1	.00		X 010
Discourse Compfaulty[2:0]	110	X	110		X	010	χ_1	0	X	010	1	.10		X 010
🕨 式 final1[2:0]	010	X	010	100	X	110	χ0	0	χ	000	0	10		X 000
🕨 😽 I1[63:0]	11011111011110	X0110101010		1100010101	01	1011111	00101	1010	00000000	000000000000000	1101111101	01	1101011	00000.
▶ 📑 I2[63:0]	11011111011110	X1010101010		1100010101	10	1011111	00101	1010	00000000	00000000000000	(1101111101	X101	1101011	X00000.
ь 📑 В [63:0]	10111111011110	X1100101010		1010010101	110	1011111	01001	11010	00000000	000000000000000	1011111101	X110	1101011	X00000.
▶ 📑 I4[63:0]	10111111011110	X1100101010		1001010101	11	0011111	01001	11010	000000000	000000000000000	1011111101	X11	0101011	X00000.
🕨 😽 numberoffaultsdeted	6	X2X 4	(6) 4	2	3	5	χ	2	χ	3	6	X	3	χ 3
COMPnofault1[2:0]	100	X	100	010		100	X o	0	X	001	100	Σ	100	X 001
COMPfaulty1[2:0]	110	XX 010	110	X 010	X	010	χ_1	0	X	010	110 010	X	110	X 010
🗓 faulta1	1										2			1
🗓 faulta2	1													
🗓 faulta3	1	1												
30		-							-					
		X1: 160.000 m	ns											

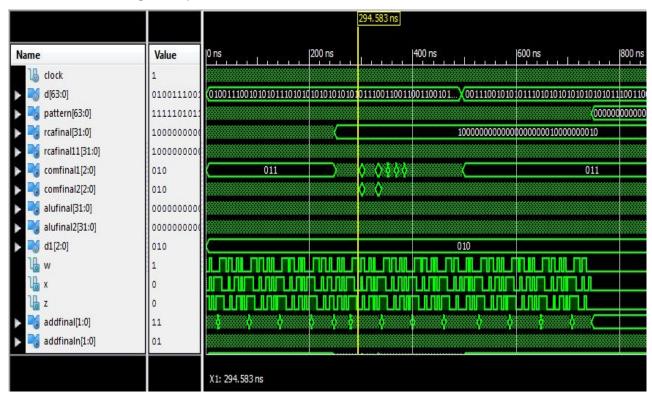
Figure 18: Simulation result of proposed method for 32-bit magnitude comparator

Console								
112	2 the number of faults detected by this cluster is 2							
128	3 the number of faults detected by this cluster is 3							
144	4 the number of faults detected by this cluster is 3							
160) the number of faults detected by this cluster is 6							
176	5 the number of faults detected by this cluster is 3							
192	2 the number of faults detected by this cluster is 3							
208	3 the number of faults detected by this cluster is 3							
224	4 the number of faults detected by this cluster is 3							
240) the number of faults detected by this cluster is 3							
Console	📄 Compilation Log 🛛 🖲 Breakpoints 🕅 Find in Files Results 🚮 Search Results							

Figure 19: Console window of proposed method for 32-bit magnitude comparator

SCHEDULING METHODOLOGY:

The test procedure considered in this paper is scheduled to test different test circuits at a time. The simulation results of scheduling methodology are presented from Figure 20 to Figure 23. The simulation result of all sessions is shown in Figure 20. Figure 21 to Figure 23 represents the testing process in session-1, session-2 and session-3 respectively. In the first session, 1-bit full-adder and 32-bit ripple carry adder are tested. In session-2, 32-bit magnitude comparator and in session-3, 32-



bit ALU are tested respectively.

Figure 20: Simulation result of Scheduling methodology for all sessions

				72.500 ns	11			
Name	Value	0 ns	50 ns	- 1 2 - 13	100 ns	150 ns	200 ns	250 ns
下 📑 d[63:0]	01001110010			010011100	1010101110101010	1010101010111001	10011001100101010	0101011
16 clock	0		ກັບການ					
🔈 📑 pattern[63:0]	10110011010	000000000000000000000000000000000000000	0000000	0000000	00000000000000000	000000000000000000000000000000000000000	000000000000000000000000000000000000000	000000
🕞 📑 rcafinal[31:0]	000000000000	000000000000000000000000000000000000000	0000000	0000000	0000000000000000	000000000000000000000000000000000000000	00000000000000000	100000
🕨 📑 I1rca[63:0]	00110011010	000000000000000000000000000000000000000	0000000	0000000	00000000000000000	000000000000000000000000000000000000000	00000000000000000	100000
🕨 式 I2rca[63:0]	00110011010	000000000000	0000000	0000000	00000000000000000	\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$	00000000000000000	0 10000
🕨 📑 Brca[63:0]	01010011010	000000000000000000000000000000000000000	0000000	0000000	00000000000000000	\$00000000000000000000000000000000000000	00000000000000000	001000
🕨 式 14rca[63:0]	01100011010	000000000000000000000000000000000000000	0000000	0000000	00000000000000000	\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$	00000000000000000	000100
🕨 式 rcafinal11[31:0]	10000000000	00 02 00 0		≫		00 0 2 2 00	002202	
🕨 📷 dalu[2:0]	010	\sim				010		
🗓 wadd	0							лл
🗓 xadd	0		urn					
1 zadd	1		Л					шп
🔈 📑 addfinal[1:0]	10	00000000000	000000	00000		000000000000000000000000000000000000000		
🕨 📑 I1add[2:0]	100	00000_00000	000000	00000		000000000000000000000000000000000000000	XD000000000	xxxxx
🕨 式 I2add[2:0]	000	00000000000	000000			000000000000000000000000000000000000000		XXXXXX
🕨 📑 Badd[2:0]	000	00000000		00000		000000000000000000000000000000000000000		XXXO
		Lougun	-					- ulu
		X1: 72.500 ns						

Figure 21: Simulation result of Scheduling methodology for session-1

				338.837 ns			
Name	Value	250 ns	300 ns	350 ns	400 ns	450 ns	500 ns
1 clock	1		הההההההההה				
🕞 📷 d[63:0]	01001110010101	01001	11001010101110101	0101010101010111	001100110011001010	010101011	001110010
🔈 📑 pattern[63:0]	00111011001100	00000000000	000000000000000000000000000000000000000	00000000000000000			000000000000000000000000000000000000000
🗓 comfaulta1	0						
1 comfaulta2	0						
🗓 comfaulta3	1						1
1 comfaulta4	1						
1 comfaulta5	0						
1 comfaulta6	0			n n n			
1 comfaulta7	1						
s 式 comfinal1[2:0]	000						Ŵ
🕞 📑 I1 com[63:0]	000000000000000000000000000000000000000			000000000000000			0000000
▶ 📑 I2com[63:0]	000000000000000000000000000000000000000			000000000			000000
▶ 📑 Bcom[63:0]	000000000000000000000000000000000000000		000000000000000000000000000000000000000				00100000
🕨 式 I4 com[63:0]	000000000000000000000000000000000000000						00010000
s 😽 comfinal2[2:0]	011	0.000					00
		X1: 338.837 ns					

Figure 22: Simulation result of Scheduling methodology for session-2

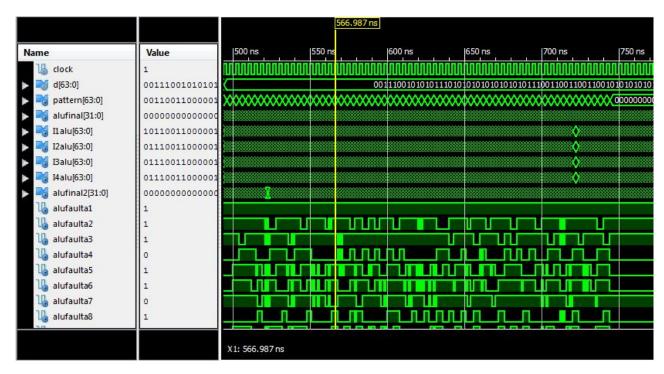


Figure 23: Simulation result of Scheduling methodology for session-3

Table 1 presents the comparison of existing methodology BILBO and proposed method called BILBO with STAR-EDT. BILBO requires more number of test patterns to detect all faults that

are injected into the test-circuits. Whereas, BILBO with STAR-EDT derived less number of test patterns compared to BILBO methodology to detect all faults that are injected into test-circuits.

TEST CIRCUIT	NUMBER OF FAULTS INJECTED	NUMBER OF PATTERNS REQUIRED IN BILBO METHODOLOGY	NUMBER OF TEST PATTERNS DERIVED BY BILBO WITH STAR-EDT METHODOLOGY
1-BIT FULL ADDER	2	5	2
32-BIT ALU	10	6	1
32-BIT MAGNITUDE COMPARATOR	7	5	3
32-BIT RIPPLE CARRY ADDER	3	9	1

 Table 1: Comparison between BILBO and proposed methodologies

The proposed method derived minimum number of test pattern required to detect all faults that are injected into the test-circuits. The number of compressed test patterns are given in Table 1.

SCHEDULING METHODOLOGY:

 Table 2: Scheduling applied to proposed methodology

TEST-CIRCUITS	SESSIONS	TIME REQUIRED TO TEST ALL TEST-CIRCUITS IN SCHEDULING
32-BIT RIPPLE CARRY ADDER	SESSION-1	
1-BIT FULL-ADDER	SESSION-1	
32-BIT MAGNITUDE COMPARATOR	SESSION-2	2.213 ns
32-BIT ALU	SESSION-3	

The scheduling methodology reduces the time required to test all test-circuits individually. Total time required to test all circuits is 2.213 ns only with three sessions. The process of scheduling and its sessions are given in Table 2.

CONCLUSION:

In this paper, BILBO methodology applied to four test-circuits is presented. In order to increase the fault coverage and to compress test data, STAR-EDT methodology is applied to BILBO method. Later, Scheduling methodology is applied in order to reduce the time required to test the circuits independently. Existed and proposed methodologies are verified on various test-circuits. The simulation results concludes that the proposed BILBO with STAR-EDT methodology requires minimum number of test patterns to detect all faults that are injected into the test-circuits and

scheduling methodology applied to BILBO with STAR-EDT reduces the time consumption by testing them parallelly.

REFERENCES:

- GrzegorzMrugalski, JanuszRajski, Lukasz Rybak, Je, drzejSolecki, and Jerzy Tyszer, "Star-EDT: Deterministic On-Chip Scheme Using Compressed Test Patterns", IEEE Transactions On Computer-Aided Design of Integrated Circuits and Systems, April 2017; 36: 4.
- Shaik Mohammed Waseem, Afroz Fatima, "Test Scheduling with Built in Logic Block Observer for NoC Architecture" International Conference on Innovative Mechanisms for Industry Applications (ICIMIA 2017).
- 3. S. Gayathri, V. SenthilKumaran, "Methodology to Detect and Diagnose Faults in Memories using BIST", International Journal of Advanced Research in Computer and Communication Engineering, August, 2015; 4: 8.
- Dong Xiang, Xiaoqing Wen and Laung-Terng Wang, "Low-Power Scan-Based Built-In Self-Test Based on Weighted Pseudorandom Test Pattern Generation and Reseeding", IEEE transactions on Very Large Scale Integration (VLSI) systems, March 2017; 25: 3.
- Alpanasingh, Manassinghal, Vijay kumar, Priyankagupta and Pallavisexena, "BIST, Built-inself- testing, A test technque" MIT International Journal of Electronics and Communication Engineering, August, 2012; 2(2): 83-88.
- Janusz Rajski, Jerzy Tyszer, Mark Kassab, and Nilanjan Mukherjee, "Embedded Deterministic Test", IEEE Transactions On Computer-Aided Design of Integrated Circuits and Systems, May 2004; 23: 5.
- Abdallatif S. Abu-Issa, "Energy-Efficient Scheme for Multiple Scan-Chains BIST Using Weight-Based Segmentation", IEEE Transactions On Circuits and Systems—II: Express Briefs, March 2018; 65: 3.
- Nandinipriya.M, Dr. (Mrs.) R.Brindha, "An Enhanced Architecture for High Performance BIST TPG", IEEE Sponsored 2nd International Conference on Innovations in Information, Embedded and Communication systems (ICIIECS)2015.
- G. Vamsi Krishna, G. SrinivasaRao, Y. Amar Babu, "Advanced Testing Methods for Reversible Logic", International Journal of Applied Engineering Research ISSN 0973-4562, 2018; 13(7): 5484-5490.
- 10. NamitaArya, AmitPrakash Singh, "Defect and Fault Detection in combinationalCircuits:Techniques and Analysis", IEEE, 2017; 332-337.

11. Yu Zhang, "Diagnostic Test Pattern Generation and Fault Simulation for Stuck-at and Transition Faults", Dissertation, August 4, 2012.