Hindawi Publishing Corporation
VLSI Design
Volume 2016, Article ID 1260879, 8 pages
http://dx.doi.org/10.1155/2016/1260879

Research Article

Implementation, Test Pattern Generation, and Comparative Analysis of Different Adder Circuits

Vikas K. Saini, Shamim Akhter, and Tanuj Chauhan

Jaypee Institute of Information Technology, Noida 201307, India

Correspondence should be addressed to Shamim Akhter; shamim.akhter@jiit.ac.in

Received 30 December 2015; Revised 14 March 2016; Accepted 6 April 2016

Academic Editor: Chang-Ho Lee

Addition usually affects the overall performance of digital systems and an arithmetic function. Adders are most widely used in applications like multipliers, DSP (i.e., FFT, FIR, and IIR). In digital adders, the speed of addition is constrained by the time required to propagate a carry through the adder. Various techniques have been proposed to design fast adders. We have derived architectures for carry-select adder (CSA), Common Boolean Logic (CBL) based adders, ripple carry adder (RCA), and Carry Look-Ahead Adder (CLA) for 8-, 16-, 32-, and 64-bit length. In this work we have done comparative analysis of different types of adders in Synopsis Design Compiler using different standard cell libraries at 32/28 nm. Also, the designs are analyzed for the stuck at faults (s-a-0, s-a-1) using Synopsis TetraMAX.

1. Introduction

In any processing unit, adders are basic building block. We look for an area and power efficient fast adders to increase system performance [1, 2]. Ripple carry adders (RCA) [3] are area and power efficient, but with drawback of being slow. Carry-select adders (CSA) [2, 4–10] are one of the fastest adders among traditional adders, but they are not power and area efficient. Common Boolean Logic (CBL) [7] adders are area-power-delay efficient adders. Carry Look-Ahead Adder (CLA) [11] is among the fastest adders with area overhead.

The organization of this paper is as follows. Section 2 deals with the analysis of adder using different topologies. Section 3 presents logic for CBL adder design. Section 4 deals with comparative analysis based on delay, area, and power among different adder architectures using Synopsis Design Compiler. The fault testing concept (s-a-0, s-a-1) for various adders is also discussed. Section 5 gives the conclusion followed by the references.

2. Different Topologies for Adder

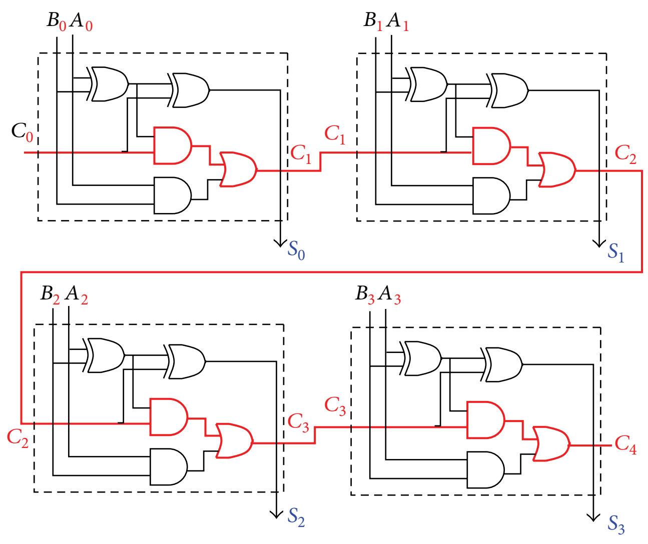
In this section, we have discussed in brief the different architectures of adder, namely, ripple carry adder (RCA), Carry Look-Ahead Adder (CLA), CSA, SQRT-CSA, and Common Boolean Logic (CBL) adder.

2.1. Ripple Carry Adder (RCA). It is basic parallel adder where a chain of adders is cascaded with carry rippled from one stage to another. Block diagram for 4-bit RCA is shown in Figure 1(a). Delay is due to the rippling of carry since the consequent blocks of adder have to wait for the carry generated from the previous adder. Delay of N-bit RCA adder is

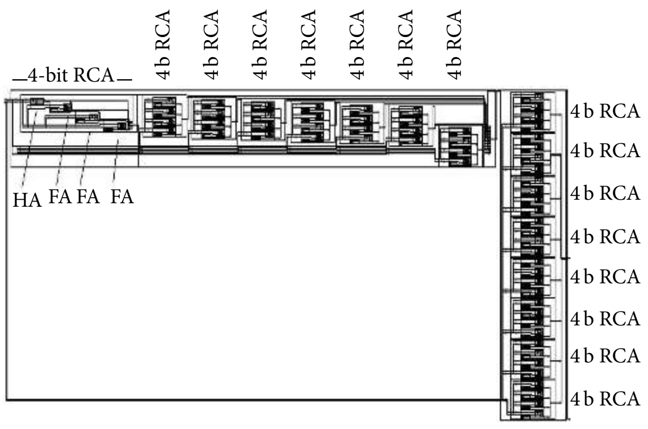
$$T_{\text{RCA}} = T_{\text{set-up}} + (N-1)\,T_{\text{gate}} + T_{\text{sum}}, \tag{1}$$

where $T_{\text{set-up}}$ is input setup time (for generating XOR and AND of basic inputs), T_{gate} is the delay of two gates, that is, AND gate and OR gates, and T_{sum} is the delay of the XOR gate.

A 64-bit RCA is shown in Figure 1(b). There are 16 blocks and each block is a 4-bit RCA. Each block has 4 full adders except the first block which has 3 full adders and 1 half adder as the lowest bit of addends can be added using the half adder instead of a full adder.



(a)



(b)

FIGURE 1: (a) Block diagram for 4-bit RCA. (b) Block diagram for 64-bit RCA.

2.2. Carry Look-Ahead Adder (CLA). CLA use the concept of generating carry and propagating carry. We know that

$$C_{i+1} = x_i y_i + x_i C_i + y_i C_i = x_i y_i + (x_i + y_i) C_i. \quad (2)$$

$g_i = x_i y_i$ is generating function which generates the carry regardless of the value of C_i and $p_i = (x_i + y_i)$ is propagating function. Carry propagates if either of x_i or y_i is 1:

$$\begin{aligned} C_{i+1} &= g_i + p_i C_i = g_i + p_i (g_{i-1} + p_{i-1} C_{i-1}) \\ &= g_i + p_i g_{i-1} + p_i p_{i-1} C_{i-1} \\ &= g_i + p_i g_{i-1} + p_i p_{i-1} (g_{i-2} + p_{i-2} C_{i-2}) \\ &= g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + p_i p_{i-1} p_{i-2} C_{i-2}. \end{aligned} \quad (3)$$

On simplifying, we have

$$\begin{aligned} C_{i+1} &= g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \cdots \\ &\quad + p_i p_{i-1} p_{i-2} \cdots p_2 p_1 p_0 g_1 \\ &\quad + p_i p_{i-1} p_{i-2} \cdots p_2 p_1 p_0 C_0. \end{aligned} \quad (4)$$

Expressions shown above are implemented using a two-level AND-OR circuit so that C_{i+1} can be evaluated very quickly. Due to independent computation of carry, the CLA based adder is fast as compared to RCA but at a cost of increased hardware. Block diagram for 4-bit CLA is shown in Figure 2(a).

Since circuit of CLA Adder for higher order bit is complex to implement directly, we have realized it by cascading 4-bit CLA blocks. Figure 2(b) shows 64-bit adder based on CLA blocks. There are 16 blocks. Each block is of 4 bits and has circuit as shown in Figure 2(a). The carry generated from any block is rippled to its successor block. This is not a pure 64-bit adder. It has been implemented just for study purpose.

2.3. Carry-Select Adder (CSA). The carry-select adder consists of two ripple carry adders and a set of multiplexers. The block diagram for 4-bit addition using CSA is given in Figure 3(a). For adding two 4-bit numbers using CSA, we require two 4-bit full adders and that can be ripple carry adder (RCA) or Carry Look-Ahead Adder (CLA). 1st stage computes the 4-bit addition using $C_{\text{in}} = 0$ and 2nd stage computes the same with $C_{\text{in}} = 1$. After the two results are available, the correct sum and carry-out are then decided by the multiplexer once the correct carry is known [4]. For designing 64-bit adder, we can cascade the structure shown in Figure 3(a). It means we need 16 stages of CSA to create 64-bit CSA adder. The number of bits in each CSA block is uniform.

Figure 3(b) shows block diagram of 64-bit CSA. There are 2 blocks, each having 8 blocks consisting of 4-bit RCA. All blocks other than RCA0 are in pair whose C_{in} is connected to logic 0 or 1 which acts as C_{in} and the results of whom block is to be considered depend upon the carry propagating from the previous block which is controlled by the MUX which can be seen after every pair of blocks (smaller). Depending upon the carry the sum is also considered to be taken. The bunch of MUX can be seen at the end of both blocks; these control the sum. The RCA block can be of any bit not necessarily of 4 bits and the number of adders in each RCA block controls the speed and power dissipation. The more the number of multiplexers in the circuit, the more the area and power dissipation.

A CLA-CSA can be designed in the same fashion. The only difference will be that the RCA block will be replaced by the CLA blocks. The block diagram of 64-bit CLA-CSA is shown in Figure 3(c).

2.4. Square-Root Carry-Select Adder (SQRT-CSA). A 16-bit CSA with variable size is created by cascading four CSA with variable input size. The block diagram for the same is given in

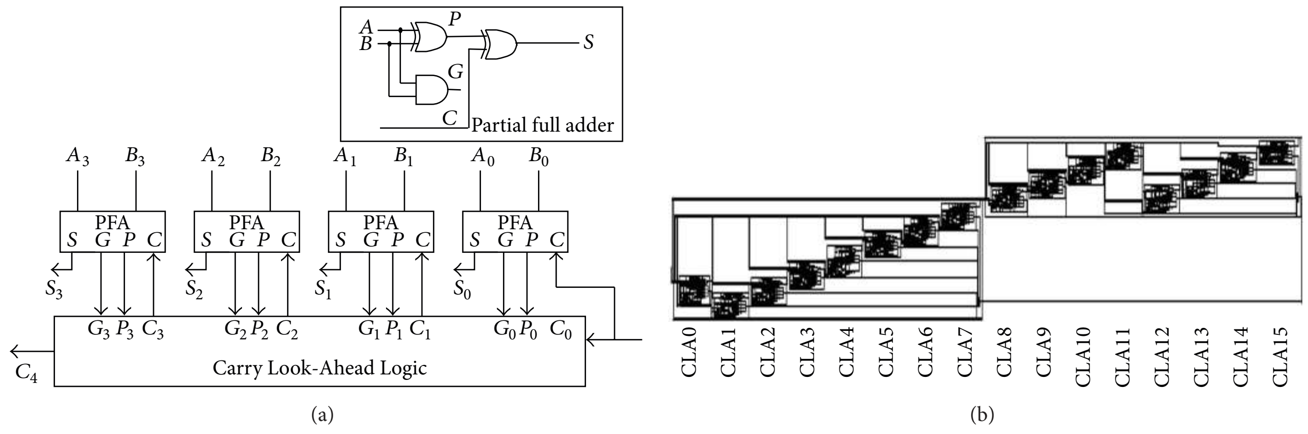

A
B
P
G
S
C
Partial full adder
A_3 B_3 A_2 B_2 A_1 B_1 A_0 B_0
PFA
S G P C
S_3 S_2 S_1 S_0
G_3 P_3 C_3 G_2 P_2 C_2 G_1 P_1 C_1 G_0 P_0 C_0
C_4
Carry Look-Ahead Logic
(a)
CLA0 CLA1 CLA2 CLA3 CLA4 CLA5 CLA6 CLA7 CLA8 CLA9 CLA10 CLA11 CLA12 CLA13 CLA14 CLA15
(b)

Figure 2: (a) Block diagram for 4-bit CLA. (b) Block diagram for 64-bit CLA based adder.

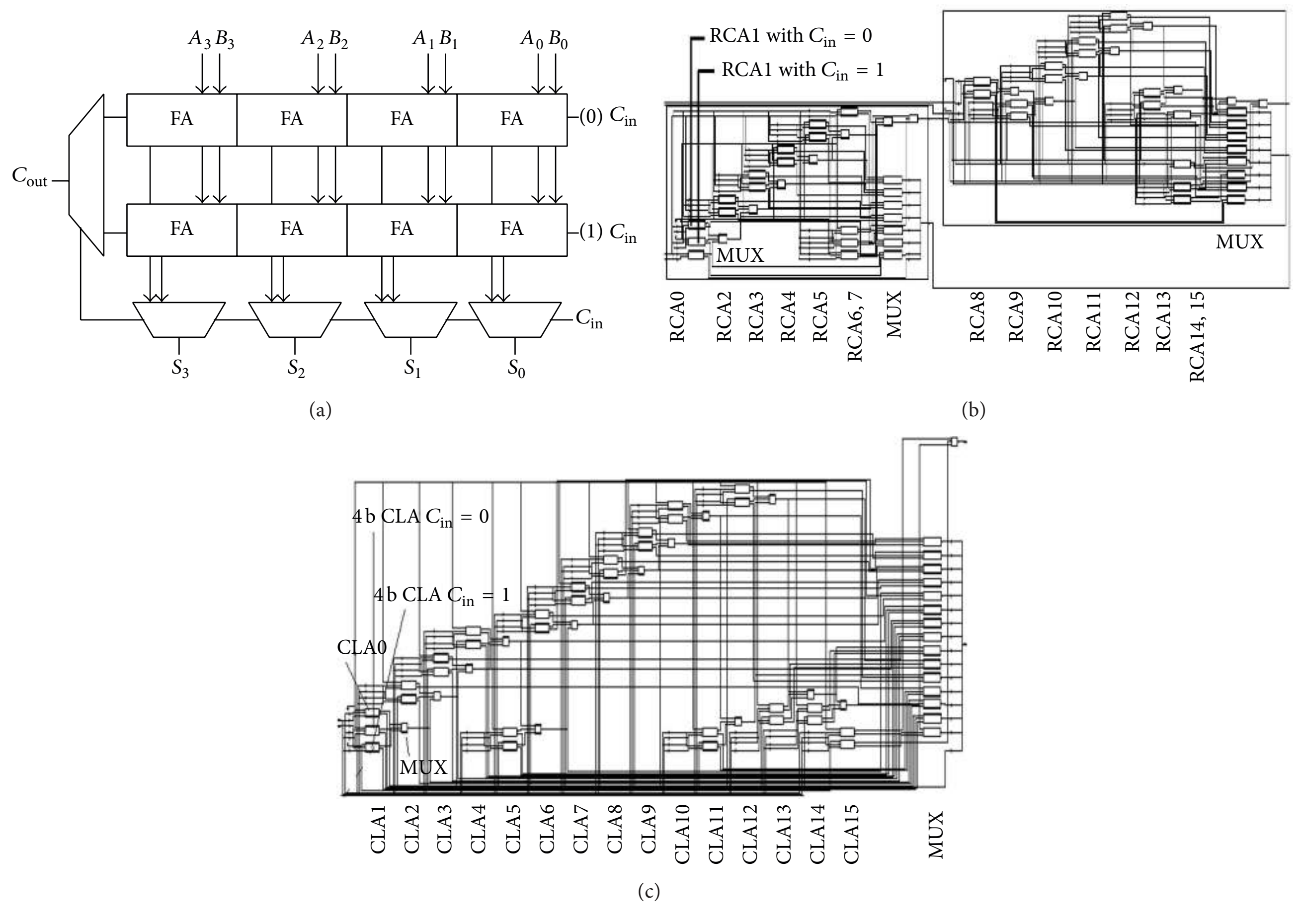

A_3 B_3 A_2 B_2 A_1 B_1 A_0 B_0
FA
(0) C_{in}
(1) C_{in}
C_{out}
C_{in}
S_3 S_2 S_1 S_0
(a)
RCA1 with C_{in} = 0
RCA1 with C_{in} = 1
MUX
RCA0 RCA2 RCA3 RCA4 RCA5 RCA6, 7 MUX RCA8 RCA9 RCA10 RCA11 RCA12 RCA13 RCA14, 15
(b)
4 b CLA C_{in} = 0
4 b CLA C_{in} = 1
CLA0
MUX
CLA1 CLA2 CLA3 CLA4 CLA5 CLA6 CLA7 CLA8 CLA9 CLA10 CLA11 CLA12 CLA13 CLA14 CLA15 MUX
(c)

Figure 3: (a) Block diagram for 4-bit CSA. (b) Block diagram for 64-bit RCA-CSA. (c) Block diagram for 64-bit CLA-CSA.

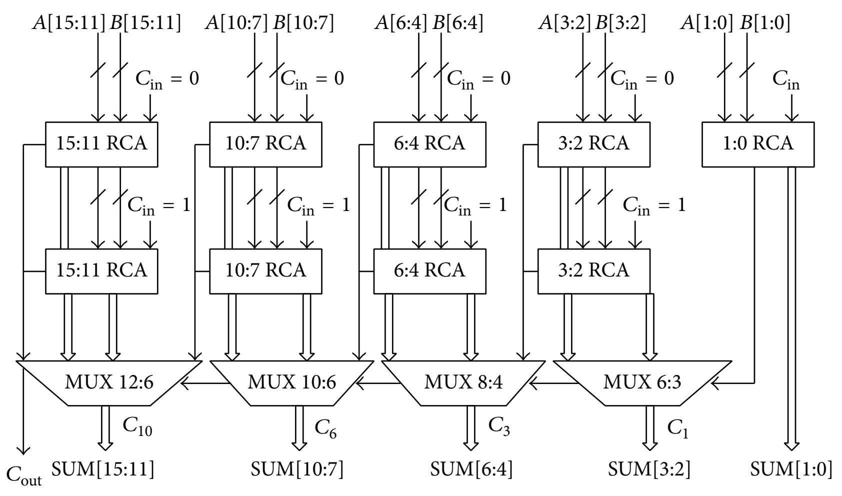
Figure 4(a). We have five stages of CSA with different inputs bits to each stage. We have an adder with block sizes of 2, 2, 3, 4, and 5, respectively. In the modified SQRT-CSA, the block of CSA with $C_{in} = 1$ is replaced with BEC (Binary to Excess Converter) [5].

Figure 4(b) shows the block diagram of 64-bit SQRT-CSA. We have an adder with block sizes of 2, 3, 4, 5, 6, 7, 8, 9, 10, and 10, respectively.

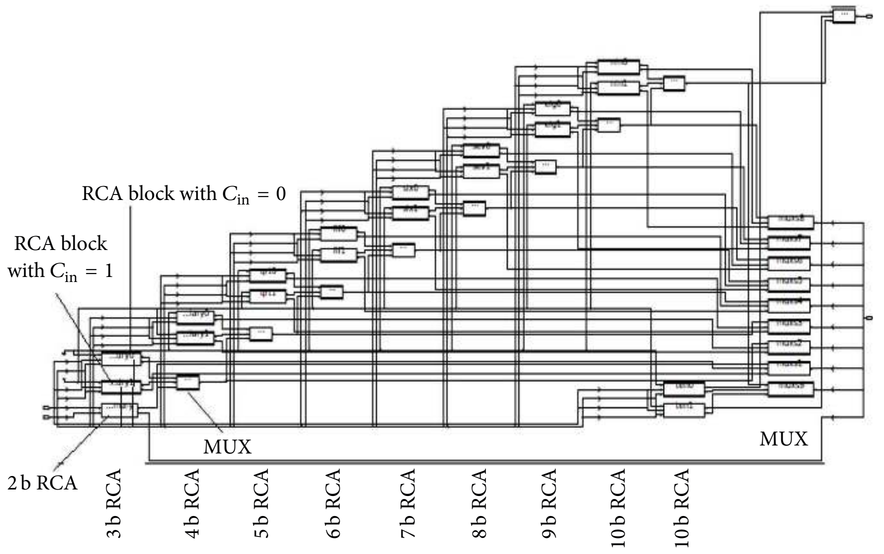
Another interesting adder is based on CBL. It is explained in Section 3 in detail.

3. Common Boolean Logic Adders

Table 1 shows the output pattern of 1-bit full adder. From the table we conclude that if the carry propagating from previous adders is "0," the current sum (S) and the carry-out bit (C)



(a)



(b)

FIGURE 4: (a) Block diagram for 16-bit SQRT-CSA [5]. (b) Block diagram for 64-bit SQRT-CSA.

TABLE 1: Design requirement for the design of full adder.

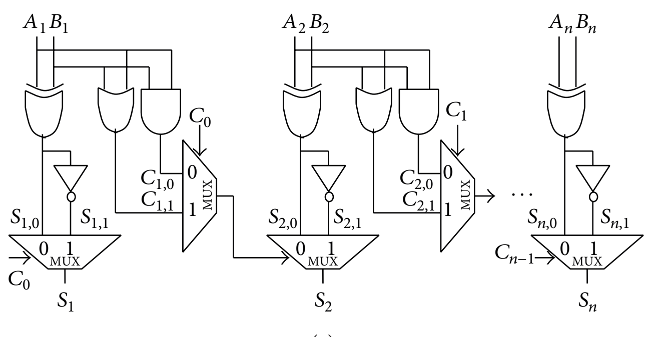
C_{in}	A	B	S	C
0	0	0	**0**	0
0	0	1	**1**	0
0	1	0	**1**	0
0	1	1	**0**	1
1	0	0	**1**	0
1	0	1	**0**	1
1	1	0	**0**	1
1	1	1	**1**	1

are the XOR and AND of the two input bits, respectively. On the other hand, if the carry propagating from previous adder is “1,” the current sum (S) bit is the XNOR and the carry-out bit (C) is OR of the two input bits, respectively. Hence,

$$S = (A \oplus B) C'_{in} + (A \odot B) C_{in}$$
$$C = (AB) C'_{in} + (A + B) C_{in}. \tag{5}$$

The implementation of the CBL based adder is shown in Figure 5(a).

From Figure 5(a), it can be seen that MUX decides the final sum depending upon the carry propagating from previous logic cell. The carry which will propagate will decide the final sum and the carry for the next logic cell. If the carry propagating from previous adder (C_0) is “0” then the sum (S_0) will be $S_{1,0}$; otherwise S_0 would be $S_{1,1}$. Similarly if C_0 is “0” then carry which will propagate to next cell would be $C_{1,0}$; otherwise propagating carry would be $C_{1,1}$.



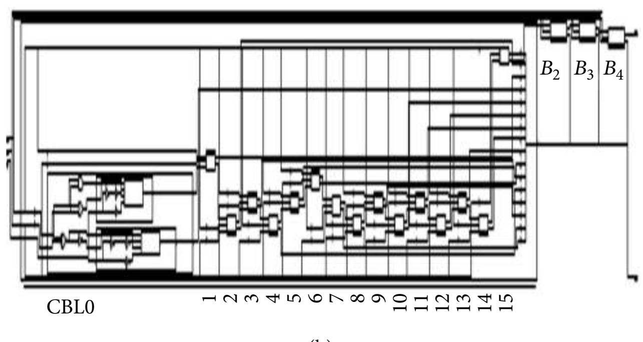


FIGURE 5: (a) Common Boolean Logic based adder. (b) 64-bit Common Boolean Logic based adder.

Delay of N-bit CBL adder is

$$T_{\text{CBL}} = T_{\text{set-up}} + (N-1)\,T_{\text{mux}} + T_{\text{sum}}, \qquad (6)$$

where $T_{\text{set-up}}$ is as defined earlier, T_{mux} is the delay of a 2×1 MUX, and T_{sum} is the delay of the XOR gate.

Figure 5(b) shows the block diagram of 64-bit Common Boolean Logic based adder. There are 4 blocks, having 16 blocks each, which means 64 blocks of Common Boolean Logic.

4. Comparative Analysis of Different Adder Architecture

In this section, we have presented the comparison of delay, power, and area of various adders using 32/28 nm digital standard cell library using Synopsis Design Compiler.

4.1. Design Compiler. Design Compiler is developed by Synopsis. Important aspects of any digital circuit like area, power, delay, and so forth can be calculated by Design Compiler just providing the HDL code of the design and digital standard cells library of the technology on which you want to work. Different standard cells libraries are provided by the Synopsis for the educational purpose. The libraries are in three formats .db, .lib, and .v. In Design Compiler we need .db format; .lib format is readable format of the .db format.

4.2. Area-Delay Estimation of Basic Circuit Using Design Compiler. We have considered 2 input AND, OR, and XOR gates. The area, delay, Dynamic Power Dissipation, and leakage are given in Table 2. Leakage power is considered for $V_{\text{dd}} = 1.05$ V DC at room temperature as per the data sheet. Operating frequency is 500 MHz.

For the understanding purpose, we are analyzing a simple circuit of full adder as shown in Figure 6. The critical path of the full adder is highlighted in the figure. It should be the double of the XOR gate, that is, .16 ns, which is exactly the same as the propagation delay as per the Design Compiler.

The area of the circuit should be the area of the 2 XOR gates (i.e., 8.640896 μm^2), 2 AND gates (i.e., 4.066308 μm^2), one OR gate (i.e., 2.033152 μm^2), and the area of the nets

TABLE 2: Area, delay, and dynamic and leakage power of SAED32.28 HVT digital standard cell libraries.

Parameters	AND	OR	XOR
Area (μm^2)	2.033154	2.033152	4.320448
Delay (ns)	0.1	0.06	0.08
Leakage (nW)	297	393	344
Dynamic (nW/MHz)	3	4	2

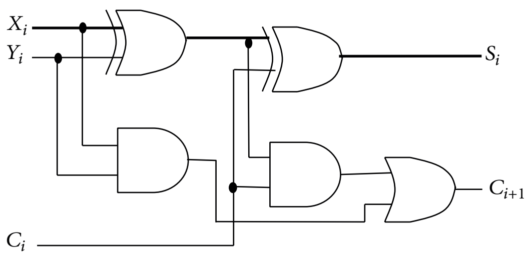


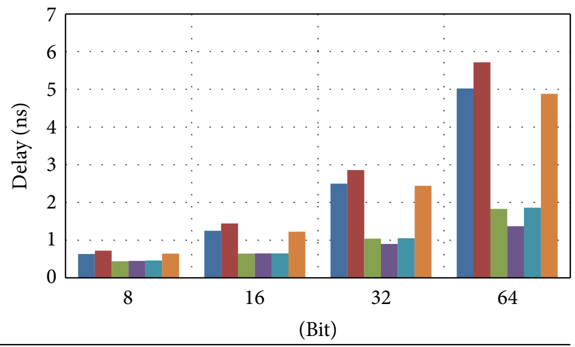
FIGURE 6: Full adder.

(i.e., 1.070661 μm^2) which is equal to 15.811017 μm^2. The area calculated by Design Compiler is 15.811013 μm^2 which is approximately the same.

We have computed the area, power, or delay for the different adder circuit. We have developed HDL code for each adder module for different input bit length. Figures 7, 8, and 9 show the delay for the various adders for 32/28 nm technology for HVT (High Voltage Threshold), RVT (Regular Voltage Threshold), and LVT (Low Voltage Threshold), respectively, for different adder architecture for 8-bit, 16-bit, 32-bit, and 64-bit adder. RVT is equivalent to standard Voltage Threshold. Units are in nanoseconds. EDK libraries for 32 and 28 nm technology are the same as provided by the Synopsis.

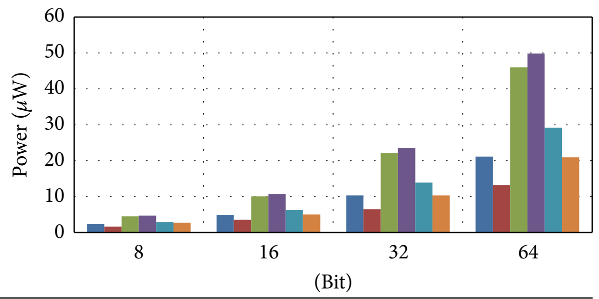
Figures 10, 11, and 12 show the Dynamic Power Dissipation for 32/28 nm technology for HVT, RVT, and LVT, respectively. Units are in microwatt. Figure 13 shows silicon area required to design hardware for various adders of different length. Units are in square micrometer.

Figure 14 shows leakage power dissipation for different adder architecture for 8-bit, 16-bit, 32-bit, and 64-bit adders. Units are in microwatt.



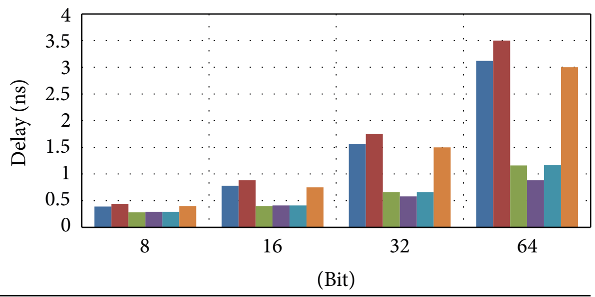
		8	16	32	64
🟦	RCA	0.63	1.25	2.5	5.02
🟥	CLA	0.72	1.44	2.86	5.72
🟩	RCA-CSA	0.44	0.64	1.04	1.83
🟪	SQRT-CSA	0.45	0.65	0.9	1.37
🟦	CLA-CSA	0.46	0.65	1.05	1.86
🟧	CSA-CBL	0.64	1.22	2.44	4.88

FIGURE 7: Delay of various adders for HVT.



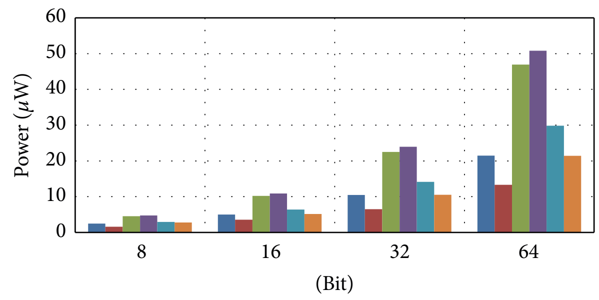
		8	16	32	64
🟦	RCA	2.4269	4.9154	10.3088	21.1339
🟥	CLA	1.6152	3.5606	6.4583	13.2288
🟩	RCA-CSA	4.4749	10.0452	22.0536	46.0029
🟪	SQRT-CSA	4.68	10.7039	23.4799	49.8229
🟦	CLA-CSA	2.903	6.2889	13.8798	29.1991
🟧	CSA-CBL	2.7384	5.0282	10.3098	20.9074

FIGURE 10: Dynamic Power Dissipation of various adders for HVT.



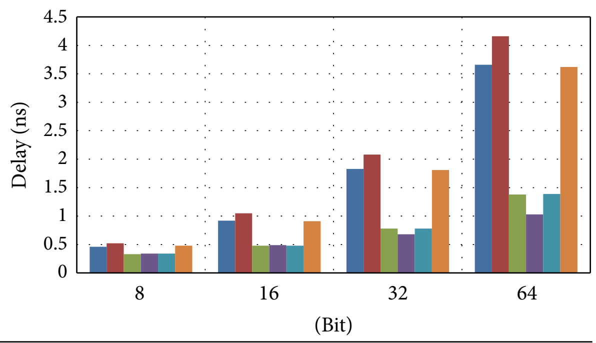
		8	16	32	64
🟦	RCA	0.39	0.78	1.56	3.12
🟥	CLA	0.44	0.88	1.75	3.5
🟩	RCA-CSA	0.28	0.4	0.66	1.16
🟪	SQRT-CSA	0.29	0.41	0.58	0.88
🟦	CLA-CSA	0.29	0.41	0.66	1.17
🟧	CSA-CBL	0.4	0.75	1.5	3

FIGURE 8: Delay of various adders for LVT.



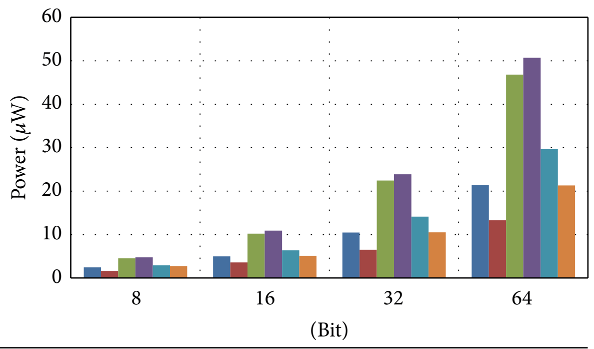
		8	16	32	64
🟦	RCA	2.463	4.9958	10.4781	21.4816
🟥	CLA	1.6308	3.5906	6.5164	13.3439
🟩	RCA-CSA	4.552	10.2441	22.4957	46.9318
🟪	SQRT-CSA	4.7734	10.9195	23.9477	50.818
🟦	CLA-CSA	2.9524	6.421	14.1767	29.8279
🟧	CSA-CBL	2.7976	5.15	10.56	21.4152

FIGURE 11: Dynamic Power Dissipation of various adders for LVT.



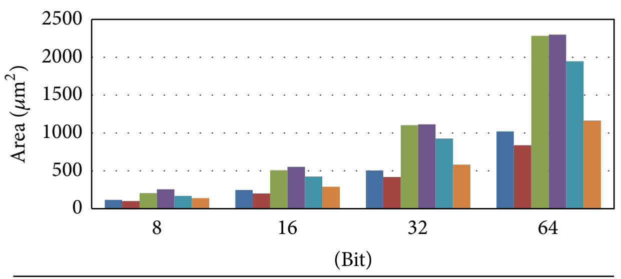
		8	16	32	64
🟦	RCA	0.46	0.92	1.83	3.66
🟥	CLA	0.52	1.05	2.08	4.16
🟩	RCA-CSA	0.33	0.48	0.78	1.38
🟪	SQRT-CSA	0.34	0.49	0.68	1.03
🟦	CLA-CSA	0.34	0.48	0.78	1.39
🟧	CSA-CBL	0.48	0.91	1.81	3.62

FIGURE 9: Delay of various adders for RVT.



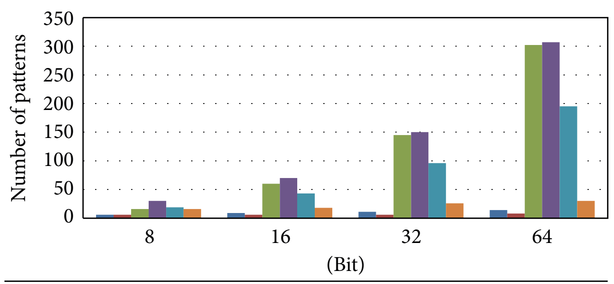
		8	16	32	64
🟦	RCA	2.4608	4.9908	10.4672	21.4586
🟥	CLA	1.6247	3.5768	6.4875	13.2844
🟩	RCA-CSA	4.544	10.2224	22.4461	46.8255
🟪	SQRT-CSA	4.7634	10.8954	23.8951	50.7042
🟦	CLA-CSA	2.9405	6.3897	14.1069	29.6805
🟧	CSA-CBL	2.7847	5.1227	10.5039	21.3014

FIGURE 12: Dynamic Power Dissipation of various adders for RVT.



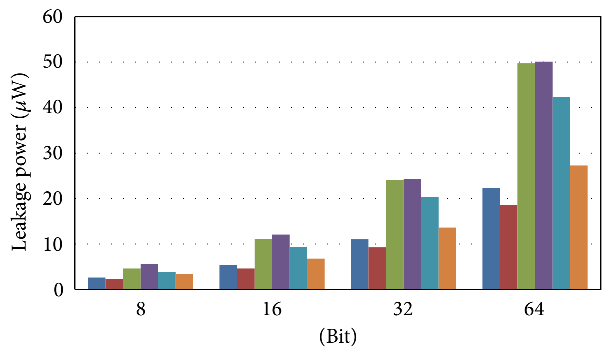
		8	16	32	64
🟦	RCA	116.99	248.44	505.79	1020.49
🟥	CLA	101.01	202.3	419.04	838.11
🟩	RCA-CSA	206.51	508.22	1104.24	2282.76
🟪	SQRT-CSA	255.93	552.59	1115.38	2299.4
🟦	CLA-CSA	169.49	426.34	928.67	1946.84
🟧	CSA-CBL	140.41	290.97	581.98	1164

FIGURE 13: Area of various adders.



		8	16	32	64
🟦	RCA	6	9	11	14
🟥	CLA	6	6	6	8
🟩	RCA-CSA	16	60	145	302
🟪	SQRT-CSA	30	70	150	307
🟦	CLA-CSA	19	43	96	195
🟧	CSA-CBL	16	18	26	30

FIGURE 15: Minimum numbers of patterns.



		8	16	32	64
🟦	RCA	2.61	5.43	11.06	22.32
🟥	CLA	2.32	4.64	9.28	18.56
🟩	RCA-CSA	4.64	11.12	24.06	49.74
🟪	SQRT-CSA	5.6	12.07	24.33	50.11
🟦	CLA-CSA	3.9	9.38	20.34	42.27
🟧	CSA-CBL	3.41	6.81	13.63	27.26

FIGURE 14: Leakage power of various adders.

4.3. Required Test Patterns to Test Various Adder Circuits. In the analysis of digital circuits, testing plays a very important role [12]. In the traditional testing we used to apply all the input and check the output corresponding to the applied inputs. If all the results used to be fine then we used to declare the hardware good only. As the size of the digital design increases the number of the input patterns increases. Then different methods were introduced to minimize the number of the input test patterns. Some of the methods like ATPG are discussed in [12] to save the time. As in the environment of such huge competition where the manufacture wants to launch the product as soon as possible, it requires lesser number of input test patterns. The hardware which will need lesser input test patterns to test will be better as per the testing point of view.

Similarly, to find the stuck at faults we need some input patterns. We apply those patterns to the input of the design and check the output. If all the outputs corresponding to all the inputs are correct we can say that the hardware is free from stuck at faults.

TetraMAX is used to generate pattern using ATPG for different circuits. With TetraMAX, designers can generate high-quality manufacturing test patterns without compromising on high performance design techniques. While such techniques may impede other ATPG tools, TetraMAX is able to obtain coverage on the resulting complex logic. Input test patterns using ATPG for different digital circuits are generated by TetraMAX and details are given in Figure 15. These patterns can be used to test the circuit for various stuck at faults after manufacturing.

Figure 15 shows number of input test patterns required to test various adders for 8, 16, 32, and 64 bits. For example for 8-bit RCA we will need only 6 patterns to declare the hardware good while there can be total of 65536 input patters to add two 8-bit values.

5. Conclusion

A comparative analysis of various digital adders has been performed in this paper. The parameters of comparison were delay, area, and power (dynamic and leakage) using digital standard cell library. Test generation analysis on the adders was also performed.

With Respect to Delay. For 8-bit adders RCA-CSA is approximately 3% faster than SQRT-CSA and CLA-CSA, 42% faster than RCA and CSA-CBL, and 60% faster than CLA.

For 16-bit adders RCA-CSA is approximately 2% faster than SQRT-CSA and CLA-CSA, 90% faster than CSA-CBL and RCA, and 125% faster than CLA.

For 32-bit adders SQRT-CSA is approximately 14% faster than RCA-CSA and CLA-CSA, 170% faster than CSA-CBL and RCA, and 210% faster than CLA.

For 64-bit adders SQRT-CSA is approximately 33% faster than RCA-CSA and CLA-CSA, 260% faster than CSA-CBL and RCA, and 310% faster than CLA.

Adders with LVT cells are 20% and 60% slower than with RVT and HVT cells, respectively.

With Respect to Power Dissipation. CLA consumes power approximately 35% lesser than RCA, 40% lesser than CSA-CBL, 50% lesser than CLA-CSA, and 70% lesser than SQRT-CSA and RCA-CSA.

Adders with HVT cells are 1.5% and 2% lesser than with RVT and LVT cells, respectively.

With Respect to Area. CLA require area approximately 15% lesser than RCA, 30% lesser than CSA-CBL, 50% lesser than CLA-CSA, and 60% lesser than RCA-CSA and SQRT-CSA.

With Respect to Leakage Power. CLA have leakage power approximately 15% lesser than RCA, 30% lesser than CSA-CBL, 50% lesser than CLA-CSA, and 60% lesser than RCA-CSA and SQRT-CSA.

Considering the number of input test patterns required to test the adder CLA, RCA, and CSA-CBL architecture are far better than CSA. Also among non-CSA architectures CLA is the best and CSA-CBL is the worst while among CSA architectures CLA-CSA is better than RCA-CSA and SQRT-CSA.

So depending upon the requirements we can choose the adder accordingly.

Similar analysis can also be performed on other various adders for different standard digital libraries, that is, 360 nm and 90 nm.

Competing Interests

The authors declare that they have no competing interests.

References

[1] S. Akhter, "VHDL implementation of fast NxN multiplier based on vedic mathematic," in *Proceedings of the 18th European Conference on Circuit Theory and Design (ECCTD '07)*, pp. 472–475, Seville, Spain, August 2007.

[2] Y. He, C.-H. Chang, and J. Gu, "An area efficient 64-bit square root carry-select adder for low power applications," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 4, pp. 4082–4085, May 2005.

[3] N. H. E. Weste, D. Harris, and A. Banerjee, *CMOS VLSI Design: A Circuits and Systems Perspective*, Pearson Education, 3rd edition, 2005.

[4] B. K. Mohanty and S. K. Patel, "Area-delay-power efficient carry-select adder," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 6, pp. 418–422, 2014.

[5] O. J. Bedrij, "Carry-select adder," *IRE Transactions on Electronic Computers*, vol. 11, no. 3, pp. 340–346, 1962.

[6] S. Akhter, S. Chaturvedi, and K. Pardhasardi, "CMOS implementation of efficient 16-Bit square root carry-select adder," in *Proceedings of the 2nd International Conference on Signal Processing and Integrated Networks (SPIN '15)*, pp. 891–896, Noida, India, February 2015.

[7] B. Ramkumar and H. M. Kittur, "Low-power and area-efficient carry select adder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 2, pp. 371–375, 2012.

[8] G. Singh, "Design of low area and low power modified 32-BIT square root carry select adder," *International Journal of Engineering Research and General Science*, vol. 2, no. 4, pp. 422–431, 2014.

[9] I.-C. Wey, C.-C. Ho, Y.-S. Lin, and C.-C. Peng, "An area-efficient carry select adder design by sharing the common boolean logic term," in *Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS '12)*, pp. 1091–1094, Hong Kong, March 2012.

[10] V. Kokilavani, K. Preethi, and P. Balasubramanian, "FPGA-based synthesis of high-speed hybrid carry select adders," *Advances in Electronics*, vol. 2015, Article ID 713843, 13 pages, 2015.

[11] D. Yagain, V. Krishna A, and A. Baliga, "Design of high-speed adders for efficient digital design blocks," *ISRN Electronics*, vol. 2012, Article ID 253742, 9 pages, 2012.

[12] M. L. Bushnell and V. D. Agrawal, *Essential of Electronic Testing for Digital Memory and Mixed-Signal VLSI Circuits*, Kluwer Academic Publishers, 2000.