Implementation of High Speed and Low Power Carry Select Adder Using BEC

Report

MASTER OF ENGINEERINGING

Microelectronics

By

Shivam Kumar (2022H1230198H)
Dikshant Bhatt (2022H1230169H)
Nikhil Devkar (2022H1230166H)

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE,PILANI (HYDERABAD)

INTRODUCTION –>

Power consumption is an important efficiency factor in designing Very Large Scale Integrated (VLSI) circuit. In many computers and other kinds of processors, adders are used in arithmetic logic units or ALU. They are also used in other parts of the processor, such as to calculate addresses, table indices, increment and decrement operators, and similar operations. Although adders can be constructed for various number representations, such as binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where two's complement or one's complement is being used to represent negative numbers, it is trivial to modify an adder.

into an adder–subtractor. Other signed number representations require more logic around the basic adder. Hybrid adders combine several addition schemes to achieve implementation delay/area constraints. Hybrid adders use carry-lookahead and carry-select schemes. Adders are the most used of the more complex operators in a typical design. In certain cases, ASIC designers sometimes employ special versions using combinations of half-adders and full-adders. This may work very efficiently in the case of a gate array device, for example, but it will typically result in a very bad FPGA implementation. The simplest possible adder circuit for binary digits is called a half-adder, and it allows two bits to be added, with a main output and a carry output. The half-adder can be constructed by using a combination of an exclusive-OR gate and an AND gate. The carry bit is 0 except when both inputs bits are 1, which is as required by the rules of binary arithmetic. This half-adder is such a useful circuit that it is made in IC form in its own right, and it can, in turn, be used to create other circuits. The name 'half-adder' arises because it can be used only as a first stage in an adder circuit. If we need to add only two bits, the half-adder is sufficient, but if we need to add, say, eight pairs of bits, as when we add two bytes, then the other adders will have three inputs: the bits that are to be added plus the carry bit from the previous stage. For example, adding 1011 and 0011 can use a half-adder for the lowest order pair, giving a 0 output and a carry bit. The addition of the next bits is 1 + 1 + 1 and the carry bit. This gives a 1 output and a 1 carry. The next addition uses the carry from the previous stage, and the last addition uses no carry.

ADDER CIRCUITS-

Carry select adder-

The basic building block of a carry select adder, with the block size of 4 is shown in Fig. 1. Two 4-bit ripple carry adders are multiplexed, where the resulting carry and sum bits are selected by the carry-in. Since one ripple carry adder assumes a carry-in of 0, and the other assumes a carry-in of 1, selecting which adder had the correct assumption via the actual carry-in yields the desired result. CSAs use multiple narrow adders to create fast wide adders. A CSA breaks the addition problem into smaller groups. It is one of the fast types of adder. The adder consists of two independent units. Both units implement the addition operation in parallel.To increase the speed of addition, the whole operation is divided into smaller groups, like 8-bit groups and then for each group four additions are performed in parallel, one assuming carry-in is 0 (CIN=0) and the other assuming the carry-in is 1 (CIN=1).

When the carry-in is eventually known, the correct sum is simply selected through an N-bit 2-to-1 mux. The adder based on this approach is known as carry select adder (CSA). The application CSA is used in data processing processor to perform fast arithmetic function.

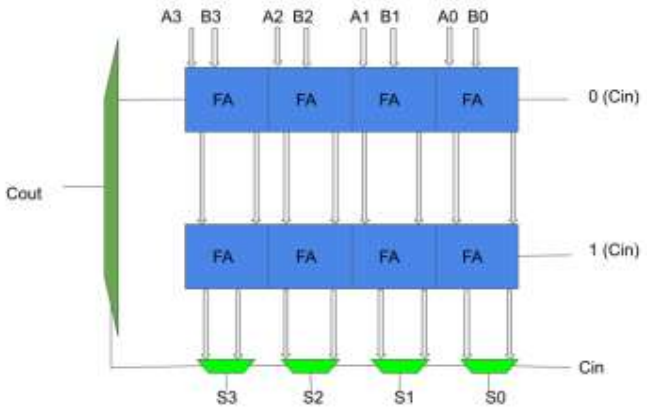


Fig-1

Ripple carry adder-

A ripple carry adder(RCA) is a logic circuit in which the carry-out of each full adder is the carry-in of the next, significant full adder. It is called a ripple carry adder because each carry bit gets rippled into the next stage. In a ripple carry adder the sum and carry-out bits of any half adder stage is not valid until the carry-in of that stage occurs. Propagation delays inside the logic circuitry causes invalid sum and carry-out bits. Propagation delay is time elapsed between the application of an input and occurrence of the corresponding output. Consider a NOT gate, when the input is “0”, the output will be “1”, and vice versa. The time taken for the NOT gate’s output to become “0” after the application of logic “1” to the NOT gate’s input is the propagation delay here. Similarly the carry propagation delay is the time elapsed between the application of the carry-in(Cin) signal and the occurrence of the carry-out (Cout) signal. In 4-bit RCA sum out S0 and carry out Cout of the Full Adder 1 is valid only after the propagation delay of Full Adder 1. In the same way, Sum out S3 of the Full Adder 4 is valid only after the joint propagation delays of Full Adder 1 to Full Adder 4. In simple words, the final result of the ripple carry adder is valid only after the joint propagation delays of all full adder circuits inside it.

PROPOSED METHOD-

In the proposed design, Carry Select Adder (CSA) is designed by using Ripple Carry Adder (RCA) with Binary to Excess-1 Converter (BEC). Because RCA would incur more delay and more power consumption, to reduce the propagation delay and power consumption, we use BEC-based Carry Select Adder. The goal of fast addition is achieved using BEC together with a multiplexer (mux). One input of the 2:1 mux, as shown in Fig. 2, has inputs (B3, B2, B1, and B0), The other input of the mux is the BEC output.

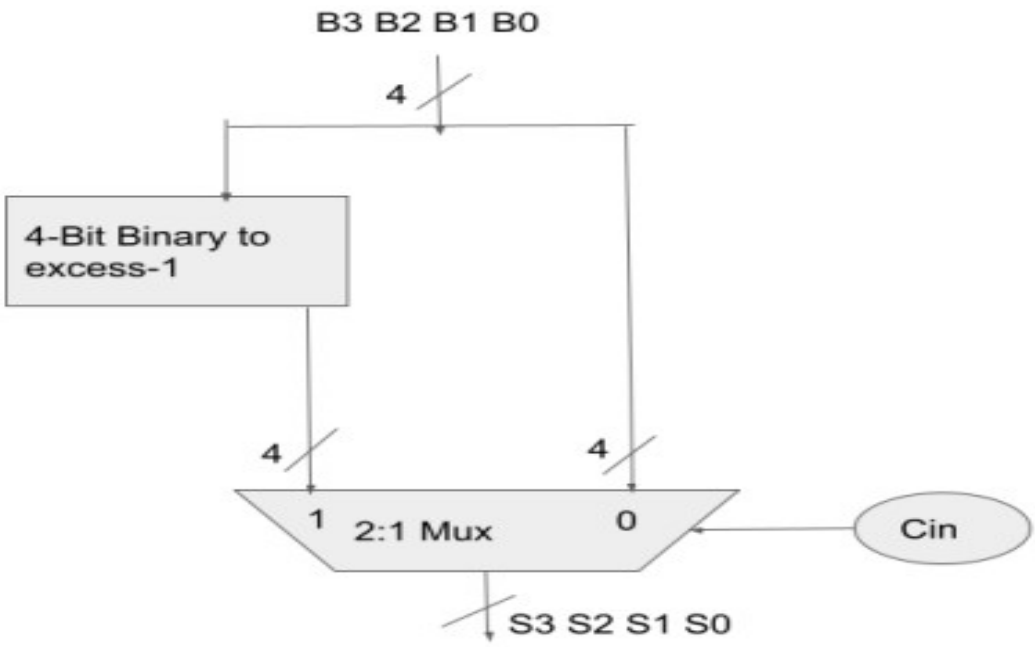


Fig -2- 4-bit BEC with 2:1 mux.

The CSA functions by using the 4-bit BEC along with the multiplexer. One input of the 2:1 multiplexer gets as is input (B3, B2, B1, and B0) and another input of the multiplexer is the BEC output. This produces the two possible sectional results in parallel, and according to the control signal Cin, the multiplexer is used to select either the BEC output or the direct inputs. The importance of the BEC logic stems from the large silicon area reduction when the CSLA with large number of bits are designed. The Boolean implementation of the 4-bit BEC is shown as-

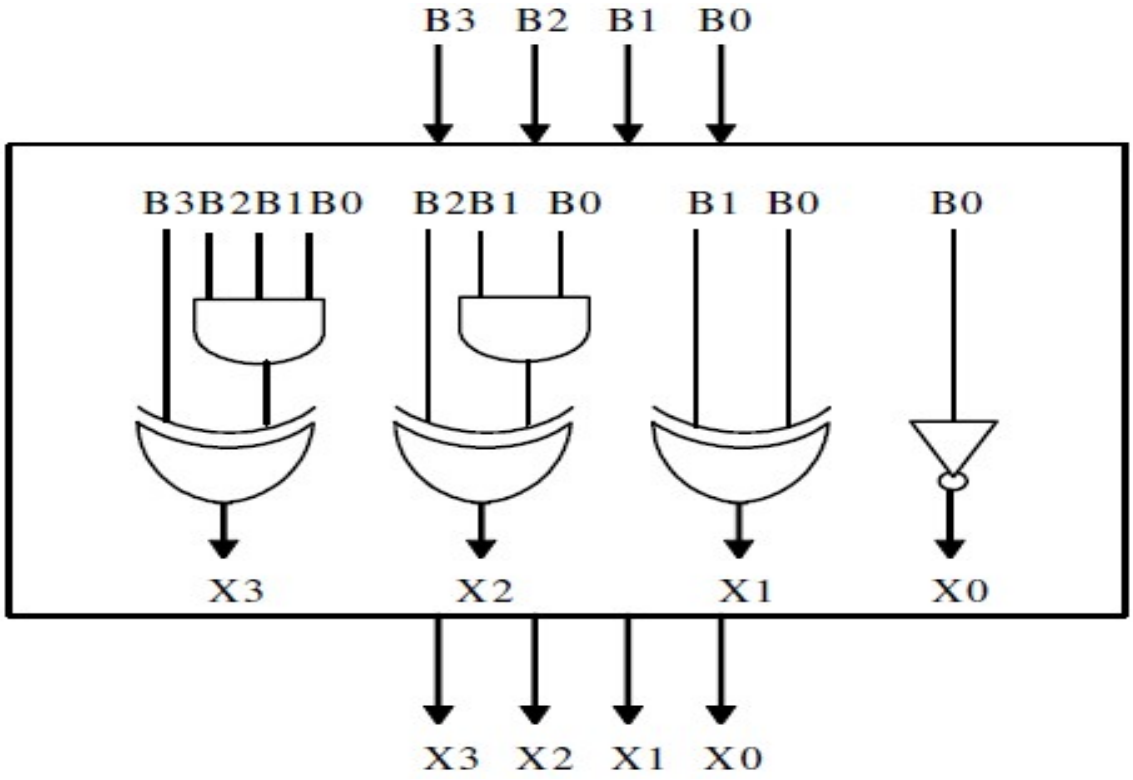


Fig-3 implementation of 4 bit BEC

The basic idea of this work is to use BEC instead of RCA with Cin=1. CSLA BEC uses fewer number of logic gates than N-bit full adder structure in order to optimize area and power. N-bit RCA is replaced by (N+1)-bit BEC. Therefore, the modified CSLA has low power and less area than conventional CSLA. CSLA has been picked for correlation with modified design using BEC as it has more stabilized delay, less area and low power. To reduce the delay, area and power, the design is modified by using BEC instead of RCA with Cin=1.

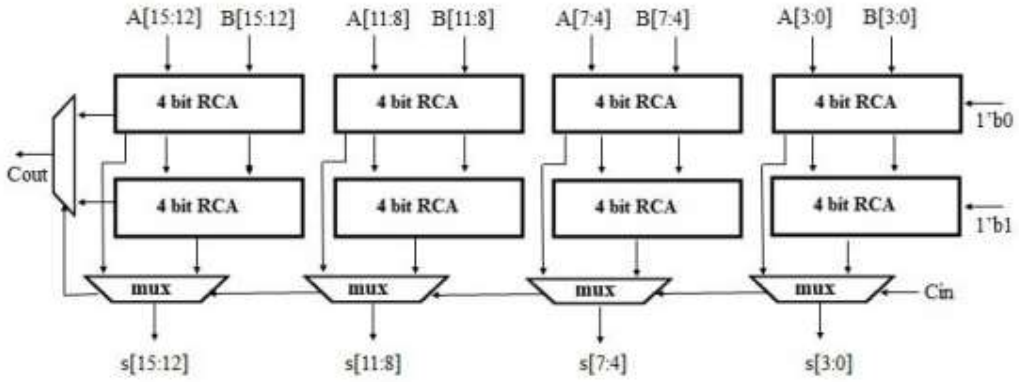


Fig. 4 Block diagram of CSA with RCA

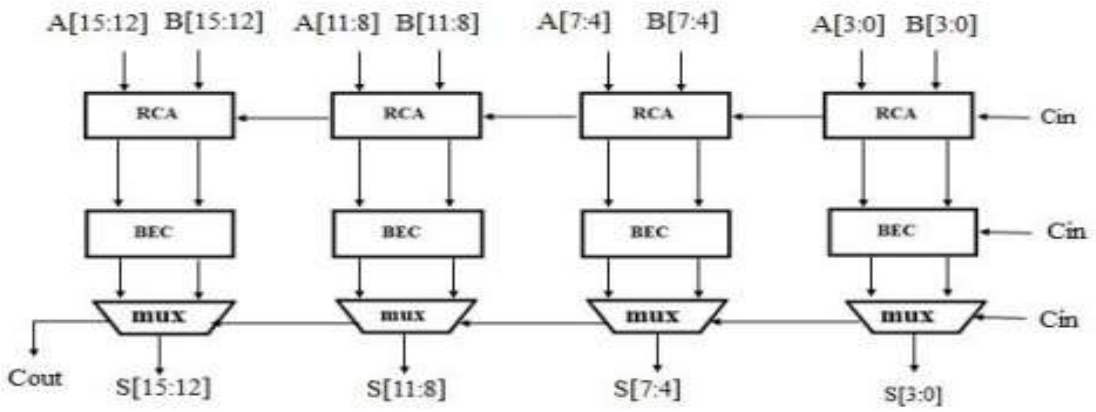


Fig 5 Block diagram of proposed 16-bit CSA with BEC.

The entire work has been performed by using of Binary to Excess-1 Converter (BEC) instead of RCA with Cin = 1 in the regular CSLA to achieve lower power consumption The main advantage of this BEC logic comes from the fewer number of logic gates than the n-bit Ripple Carry Adder (RCA). The structure and the truth table of 4-bit BEC is shown in below Fig. 6 and Table 1, respectively. FIg.4 shows the block diagrram of CSA with RCA, where few of the full adders from a conventional CSA have been replaced by 4-bit RCA. This architecture is used to reduce area and lower the power when compared to converntional CSA.

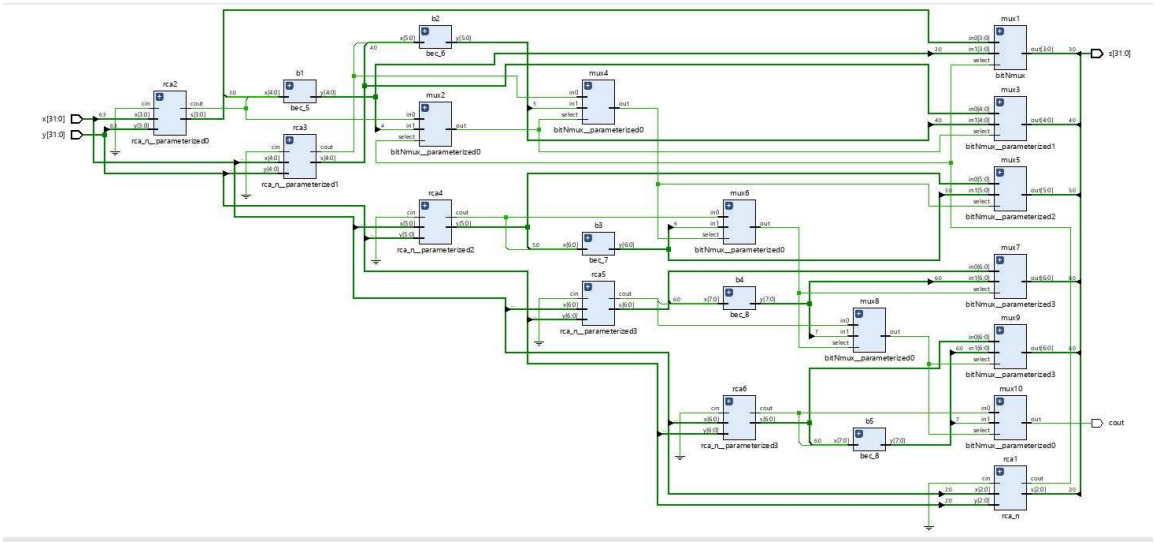
Binary Logic	*Excess-1 Logic*
0000	0001
0001	0010
0010	0011
0011	0100
0100	0101
0101	0110
0110	0111
0111	1000
1000	1001
1001	1010
1010	1011
1011	1100
1100	1101
1101	1110
1110	1111
1111	0000

IMPLEMENTATION OF CLSA USING BEC (VIVADO – XILINX)

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 07/07/2023 10:43:38 AM
7  // Design Name:
8  // Module Name: RCA
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22 // CSA moduele 16 bit
23
24 module CSA (input [15:0]A,
25             input [15:0]B,
26             input cin,
27             output [15:0]sum,
28             output carry
29             );
30
31              wire [15:0]temp_0,temp_1;
32              wire c1,c2,c3,c4;
33              wire c1_1,c2_1,c3_1,c4_1;
34              wire c1_a,c2_a,c3_a,c4_a;
35
36              assign c1_a=c1|(cin&c1_1);
37              assign c2_a=c2|(c1_a&c2_1);
38              assign c3_a=c3|(c2_a&c3_1);
39              //assign c4_a=c4|(c3_a&c4_1);
40
41              RCA x1(A[3:0],B[3:0],temp_0[3:0],c1);
42              BEC y1(temp_0[3:0],temp_1[3:0],c1_1);
43
44              RCA x2(A[7:4],B[7:4],temp_0[7:4],c2);
45              BEC y2(temp_0[7:4],temp_1[7:4],c2_1);
46
47              RCA x3(A[11:8],B[11:8],temp_0[11:8],c3);
48              BEC y3(temp_0[11:8],temp_1[11:8],c3_1);
49
50              RCA x4(A[15:12],B[15:12],temp_0[15:12],c4);
51              BEC y4(temp_0[15:12],temp_1[15:12],c4_1);
52
53              mux m1(temp_0[3:0],temp_1[3:0],cin,sum[3:0]);
54              mux m2(temp_0[7:4],temp_1[7:4],c1_a,sum[7:4]);
55              mux m3(temp_0[11:8],temp_1[11:8],c2_a,sum[11:8]);
56              mux m4(temp_0[15:12],temp_1[15:12],c3_a,sum[15:12]);
57
58              assign carry=c4|c4_1;
59
60 endmodule
```

```
61
62
63
64  //4 bit 2:1 mux
65  module mux(input [3:0]x,
66             input [3:0]y,
67             input select,
68             output [3:0]out);
69
70   assign out=select?y:x;
71
72  endmodule
73
74
75  // 4 bit BEC module
76  module BEC(input [3:0]x,
77             output [3:0]y,
78             output c);
79  assign y[0]=~x[0];
80  assign y[1]=x[1]^x[0];
81  assign y[2]=x[2]^(x[1]&x[0]);
82  assign y[3]=x[3]^(x[2]&x[1]&x[0]);
83  assign c=~(y[0]|y[1]|y[2]|y[3]);
84
85  endmodule
86
87
88
89  // RCA module with carry=0
90  module RCA(
91  input [3:0]A,
92  input [3:0]B,
93  output [3:0]sum,
94  output carry
95  );
96  wire [2:0]c;
97  fulladd X1(A[0],B[0],0,sum[0],c[0]);
98  fulladd X2(A[1],B[1],c[0],sum[1],c[1]);
99  fulladd X3(A[2],B[2],c[1],sum[2],c[2]);
100 fulladd X4(A[3],B[3],c[2],sum[3],carry);
101 endmodule
102
103
104
105 // full adder module
106 module fulladd(
107     input a,
108     input b,
109     input c,
110     output S,
111     output Carry
112     );
113     assign S= (a&b&c)|(~a&~b&c)|(~a&b&~c)|(a&~b&~c);
114     assign Carry=(a&b)|(b&c)|(c&a);
115 endmodule
116
```

Output RTL Design



DELAY AND AREA EVALUATION METHOLOGY

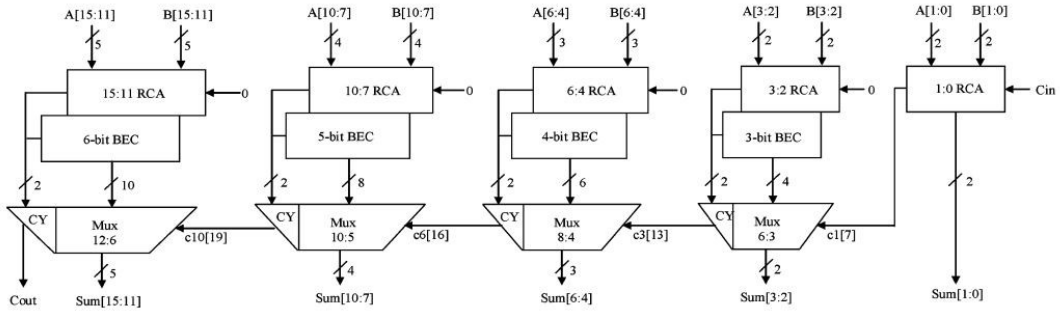


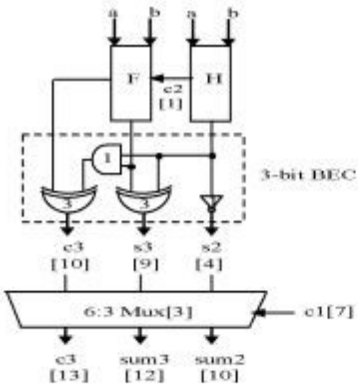
Figure given above depicts the structure of the proposed 16-b SQRT CSLA with BEC for RCA and optimised area and power. We divided the structure into five groups once more. table mentioned below displays the delay and area estimation for each group. The procedures involved in evaluating area & power.

One 2-b RCA with one FA and one HA is present in group 2 . A 3-b BEC is used in place of a second 2-b RCA to increase the output from the first by one.

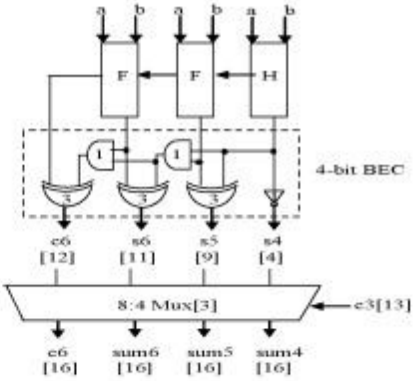
In accordance with the analysis of the delay values in Table , the arrival time of the chosen input for the 6:3 mux is earlier than the and later than the. Consequently, the final (output from mux) and sum3 depend on respectively, mux, partial (input to mux), and mux.

For the remaining groups, the mux selection input arrival time is always later than the data inputs from the BECs.
The arrival time of the mux selection input and the mux delay, therefore, determine the delay of the remaining groups.

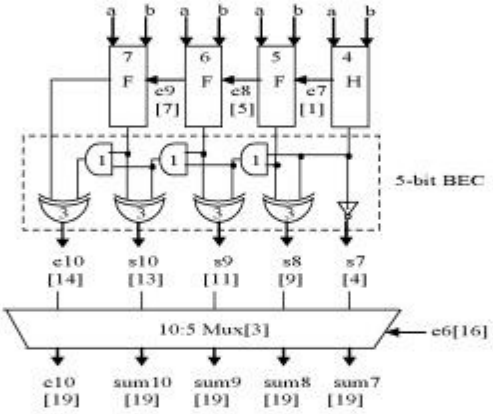
Group	Delay	Area
Group2	13	43
Group3	16	61
Group4	19	84
Group5	22	107


a b a b
F
H
c2
[1]
3-bit BEC
c3
[10]
s3
[9]
s2
[4]
6:3 Mux[3]
c1[7]
c3
[13]
sum3
[12]
sum2
[10]

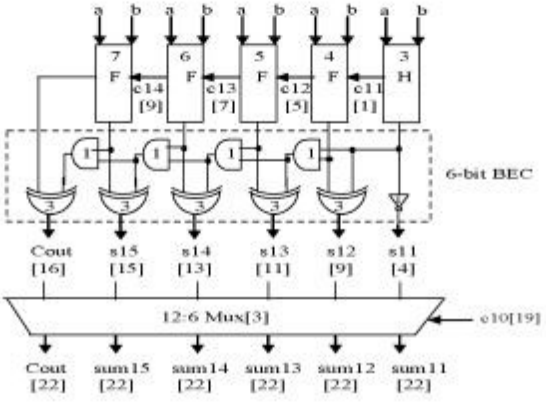
(a)


a b a b a b
F
F
H
4-bit BEC
c6
[12]
s6
[11]
s5
[9]
s4
[4]
8:4 Mux[3]
c3[13]
c6
[16]
sum6
[16]
sum5
[16]
sum4
[16]

(b)


a b a b a b a b
7
F
6
F
5
F
4
H
e9
[7]
e8
[5]
e7
[1]
5-bit BEC
c10
[14]
s10
[13]
s9
[11]
s8
[9]
s7
[4]
10:5 Mux[3]
e6[16]
c10
[19]
sum10
[19]
sum9
[19]
sum8
[19]
sum7
[19]

(c)


a b a b a b a b a b
7
F
6
F
5
F
4
F
3
H
e14
[9]
e13
[7]
e12
[5]
e11
[1]
6-bit BEC
Cout
[16]
s15
[15]
s14
[13]
s13
[11]
s12
[9]
s11
[4]
12:6 Mux[3]
e10[19]
Cout
[22]
sum15
[22]
sum14
[22]
sum13
[22]
sum12
[22]
sum11
[22]

(d)

comparision of different adders-

Adder	16-bit adder		
	Area	*Delay(ps)*	*Power(μw)*
CSA	241	103	57.941
CSA with RCA	235	198	58.931
CSA with BEC	169	113	42.986

CONCLUSION-

The architectures of Carry Select Adder(CSA) with Ripple Carry Adder (RCA) and carry select adder with Binary to Excess-1 Converter(BEC) is designed. The conventional carry select adder has the disadvantage of more power consumption. and occupying more chip area. With respect to delay time and power consumption, we can conclude that the implementation of CSA with BEC is more efficient as compared to existing methods. The main advantage of this BEC logic comes from the fewer number of logic gates than the 4-bit Full Adder (FA). With BEC, there is no carry propagation, thereby lower area and less dealy which in turn leads to low power consumption. Furthermore, the area and delay can be reduced by using 4-bit parallel prefix adders.