VLSI

Mini Project 1

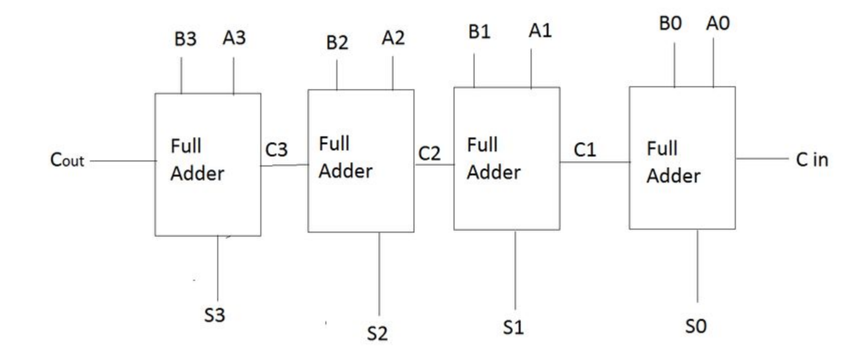
Name	Sec	BN
Bemoi Erian Ayad	1	18
Peter Atef	1	19
Rufaida Kassem	1	26
Mark Yasser Nabil	2	12

Explanation of each adder design

1. Verilog ('+') version of adders

```
module VerilogAdder (a,b,Cin,sum,Cout,of);
input [31:0] a;
   input [31:0] b;
   input Cin;
   output [31:0] sum;
   output Cout;
   output of;
   assign {Cout,sum} = a + b+ Cin; // concatination
   assign of = (a[31] == b[31])? (a[31] ^ sum[31]) : 'b0; //if the 2 signs of the operands are equal
so check if the sum has the same sign else of =0
endmodule
```

2. Ripple Carry Adder



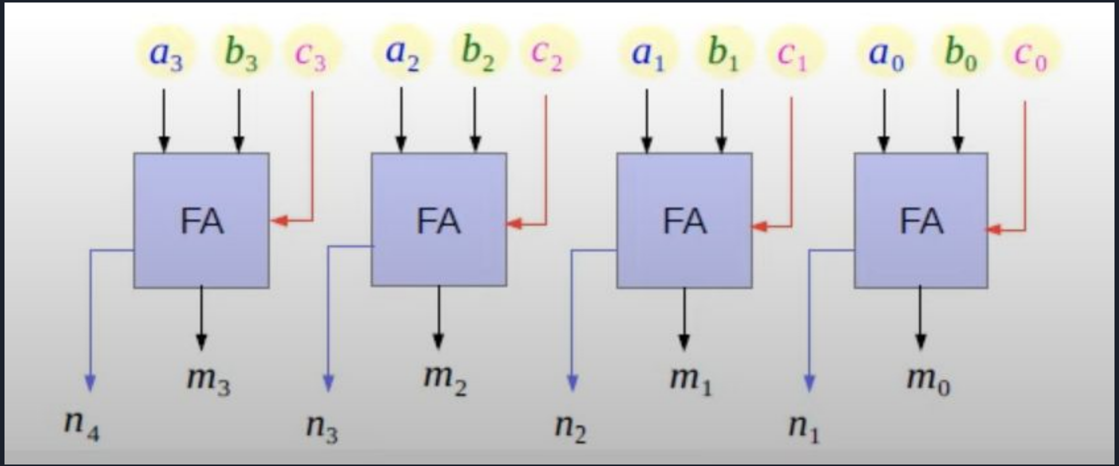
The **ripple carry adder** is constructed by cascading full adders (FA) blocks in series. One full adder is responsible for the addition of two binary digits at any stage of the ripple carry. The carry-out of one stage is fed directly to the carry-in of the next stage. It is not efficient when large bit numbers are used. One of the most serious **drawbacks** of this adder is that the delay increases linearly with the bit length. The **worst-case delay** of the RCA is when a carry signal transition ripples through all stages of adder chain from the least significant bit to the most significant bit.

3. Carry Save Adder

The carry-save adder consists of n full adders, each of which computes a single sum and carries bit based solely on the corresponding bits of the three input numbers.

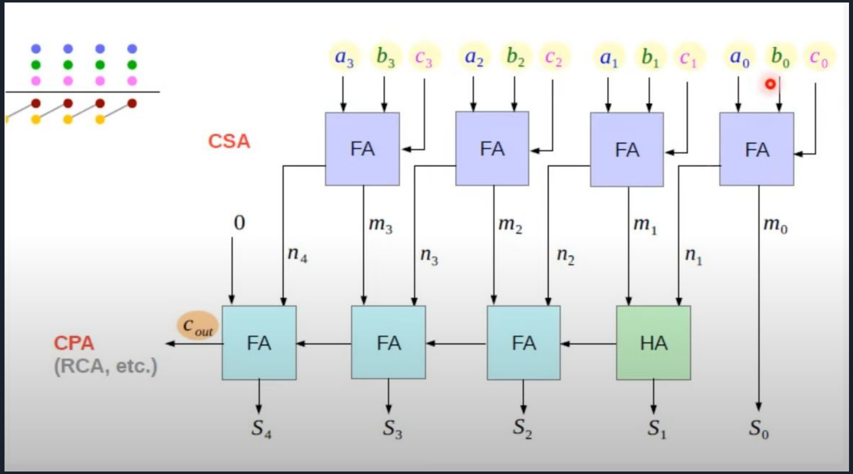
Unlike ripple carry adder, there is no intermediate carry propagation.

So, each full adder is able to compute 3 different numbers (carry-in can be used as a 3rd input).


a_3 b_3 c_3 a_2 b_2 c_2 a_1 b_1 c_1 a_0 b_0 c_0
FA
FA
FA
FA
m_3 m_2 m_1 m_0
n_4 n_3 n_2 n_1

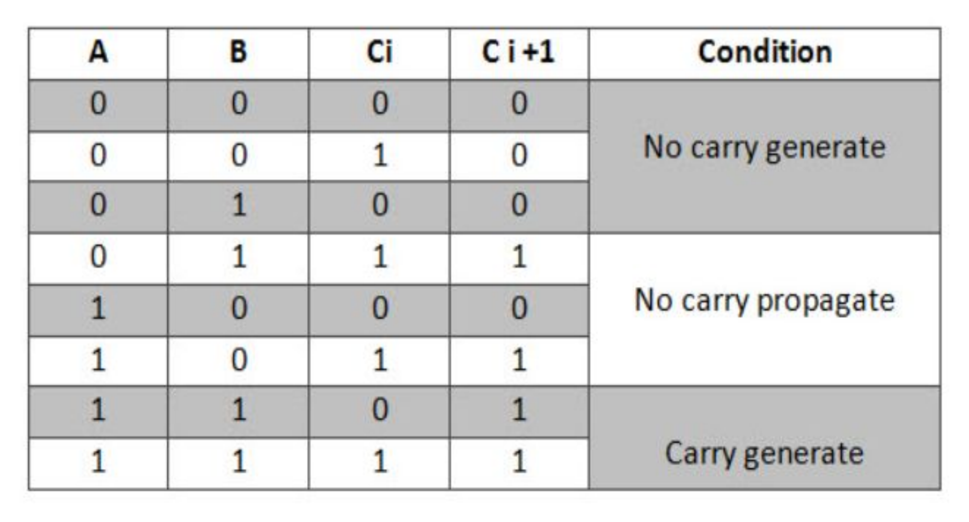
3. Carry Save Adder

The entire sum can then be computed by shifting the carry sequence left by one place and appending a 0 to the most significant bit of the partial sum sequence and adding this sequence with the carry-out sequence.

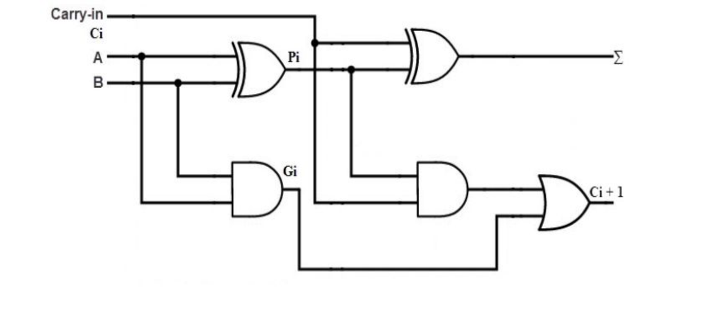


A carry save adder is typically used in a binary multiplier, since a binary multiplier involves addition of more than two binary numbers after multiplication

4. Carry Look-Ahead Adder



A	B	Ci	C i +1	Condition
0	0	0	0	No carry generate
0	0	1	0	
0	1	0	0	
0	1	1	1	No carry propagate
1	0	0	0	
1	0	1	1	
1	1	0	1	Carry generate
1	1	1	1	

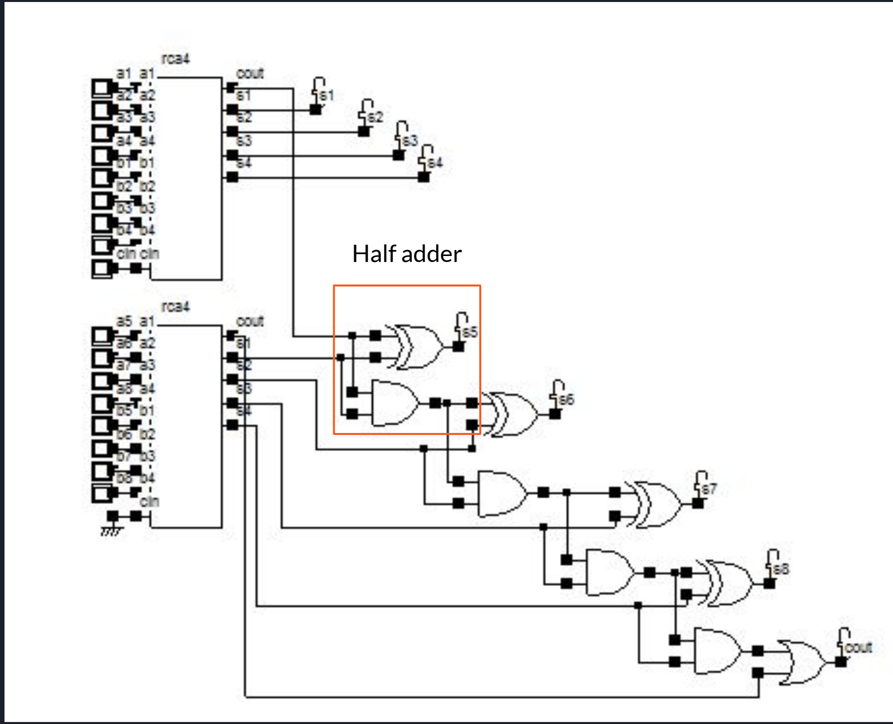


```
G[i] = A[i] & B[i]
P[i] = A[i] | B[i]
Cin[i+1] = G[i] | (P[i] & Cin[i]);
```

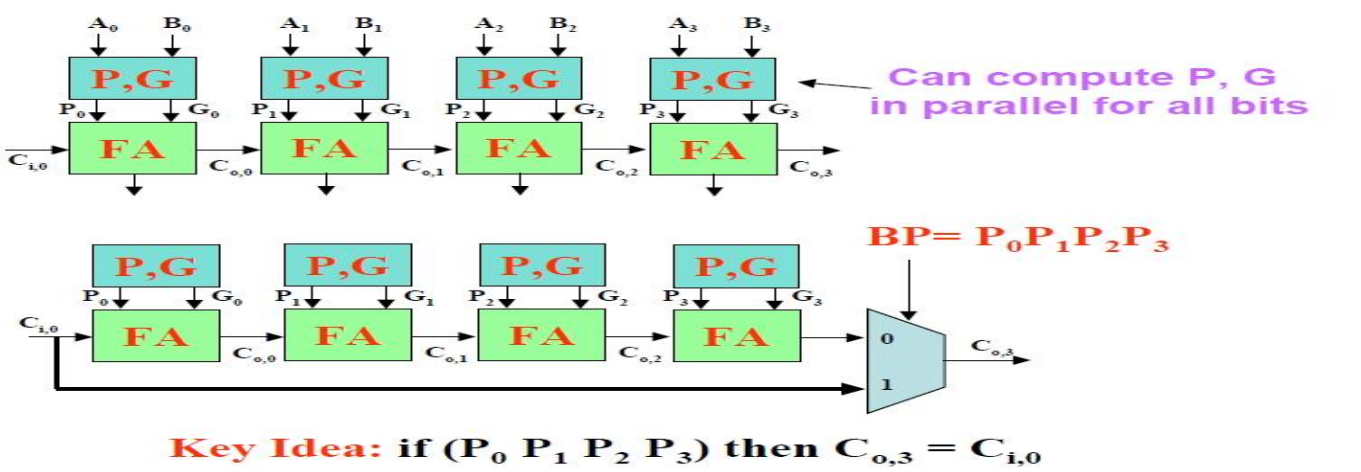
Carry out is calculated separately which increases its speed very much

5. Carry Increment adder

An 32-bit increment adder includes two Ripple carry adder of 16 bit each. The first ripple carry adder adds a desired number of first 16-bit inputs generating a plurality of partitioned sum and partitioned carry. Now the carry out of the first block RCA is given to Cin of the conditional increment block. Thus the first 16-bit sum is directly taken from the ripple carry output. The second RCA block regardless of the first RCA output will carry out the addition operation and will give out results which are fed to the conditional increment block. The input Cin to the second RCA block is given always low value. The conditional increment block consists of half adders. Based on the value of Cout of the 1st RCA block, the increment operation will take place. Here the half adder in carry increment block performs the increment operation. Hence the output sum of the second RCA is taken through the carry increment block.



6. Carry Bypass Adder



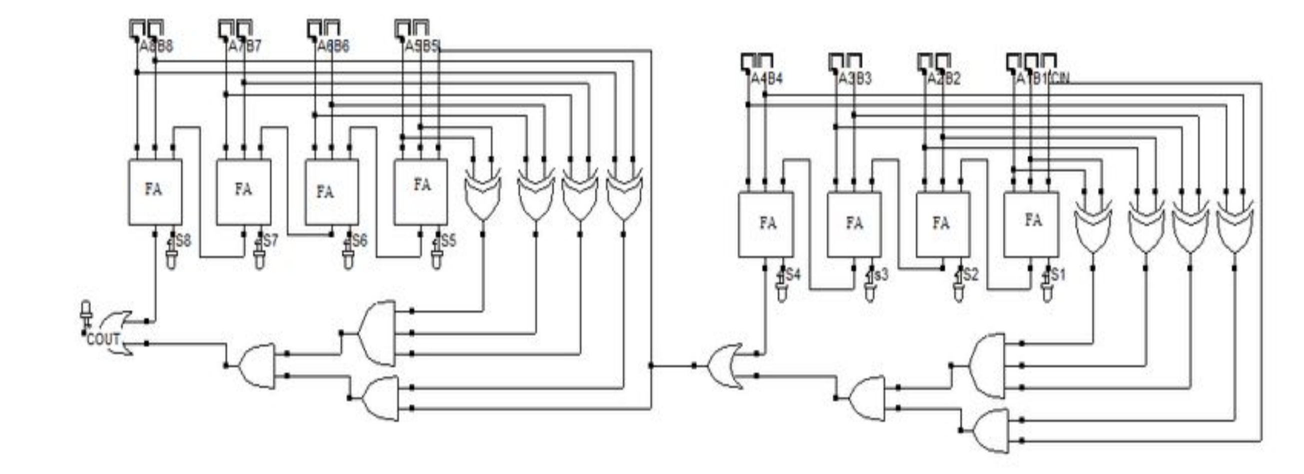
A_0 B_0 A_1 B_1 A_2 B_2 A_3 B_3
P,G
P_0 G_0 P_1 G_1 P_2 G_2 P_3 G_3
FA
$C_{i,0}$ $C_{o,0}$ $C_{o,1}$ $C_{o,2}$ $C_{o,3}$
Can compute P, G in parallel for all bits
$BP= P_0P_1P_2P_3$
0
1
Key Idea: if (P_0 P_1 P_2 P_3) then $C_{o,3} = C_{i,0}$

A carry-bypass adder is an adder implementation that improves on the delay of a ripple-carry adder with little effort compared to other adders.

The worst case for a simple one level carry-ripple-adder occurs, when the propagate-condition is true for each digit pair (ai,bi). Then the carry-in ripples through the n-bit adder and appears as the carry-out after τCRA(n)=n.τVA.

For each operand input bit pair (ai,bi) the propagate-conditions pi=ai⊕bi are determined using an XOR-Gate. When all propagate-conditions are true, then the carry-in bit c0 determines the carry-out bit.

7. Carry Skip Adder



The improvement of the worst-case delay is achieved by using several carry-skip adders to form a block-carry-skip adder. It has optimum daly of O(n^0.5).

p[i , i + 3] = pi + 3 • pi + 2 • pi + 1• pi

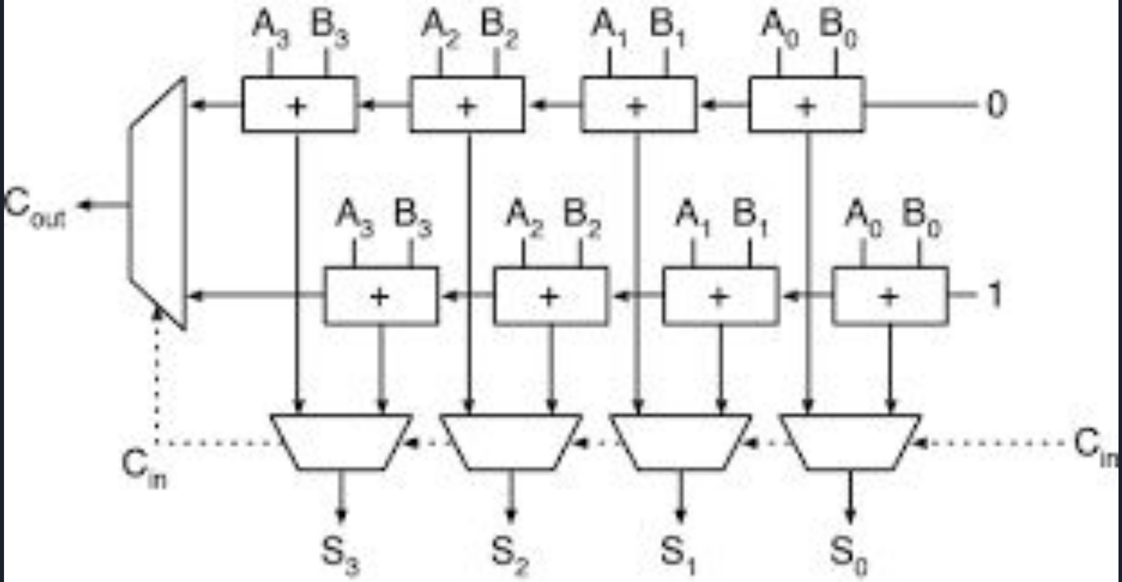
carry = ci + 4 + p[i, i + 3] • ci

If p[i, i + 3] =0, then the carry-out of the group is determined by the value of ci + 4 . However, if p[i , i + 3] =1 when the carry-in bit is ci =1, then the group carry-in is automatically sent to the next group of adders.

8. Carry Select Adder

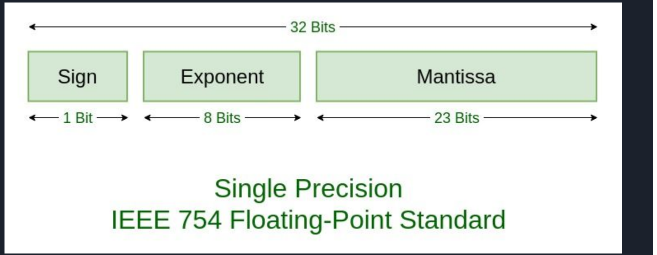
Adding two n-bit numbers with a carry-select adder is done with two adders (therefore two ripple-carry adders), in order to perform the calculation twice, one time with the assumption of the carry-in being zero and the other assuming it will be one. After the two results are calculated, the correct sum, as well as the correct carry-out, is then selected with the multiplexer once the correct carry-in is known.

Optimum delay : $O(\sqrt{n})$

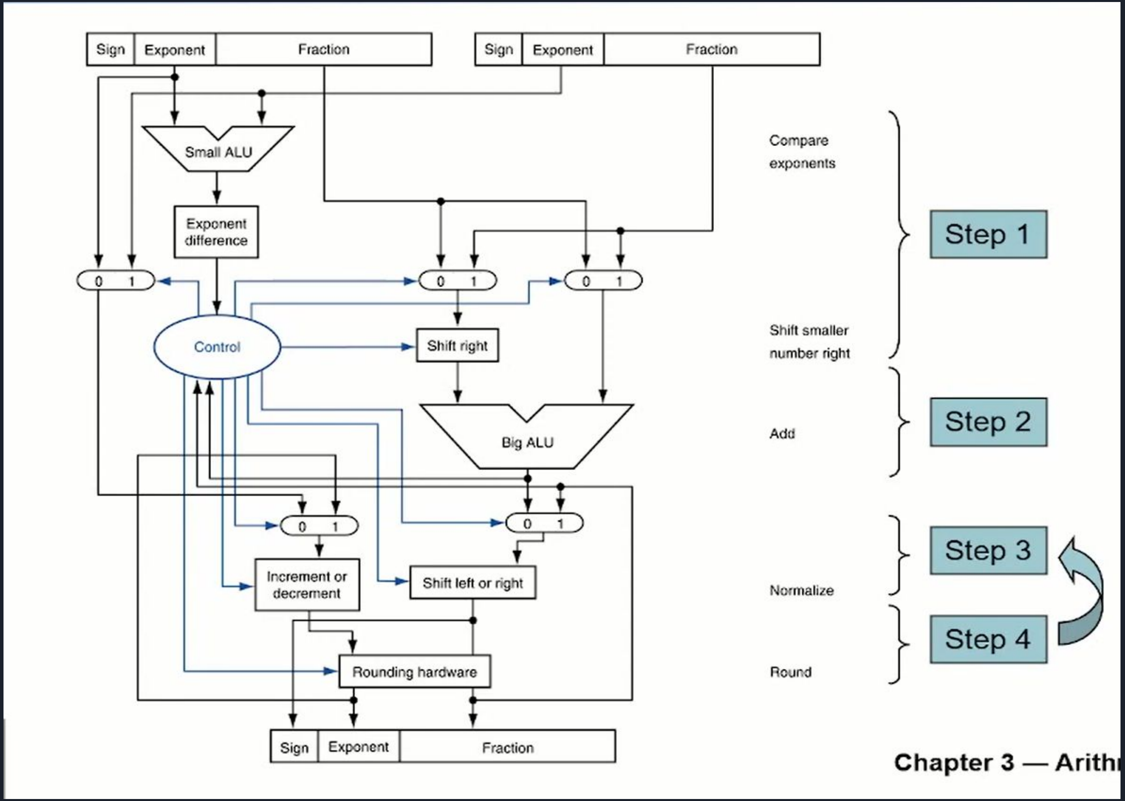


9. FLoating Point Adder

Input and
output format



Single Precision
IEEE 754 Floating-Point Standard



9. FLoating Point Adder

Step 1:

1. We Compare the exponents of the 2 numbers and get the difference between them
2. Shift right for the number with the smaller exponent by the difference of the two exponents

Step 2:

1. Add the mantissa of the 2 numbers together
2. Make the exponent of the result is the exponent of the largest number

Step 3:

1. Normalize the answer

Justification of our choice of adder used with floating point

We chose “**Carry Look-Ahead Adder**” because:

- It has the least max delay compared to other adders.
- Its total power is near an average value compared to other adders.
- It has an average area compared to other adders.

In conclusion, this adder offers the best trade-off between (time - speed - area).

Additional Challenges

1. It was so complicated to understand how the floating point adder works and to implement it.
2. Not so much resources are available that explains floating point adder implementation.
3. It was challenging to make synthesis and constraints because it is only available on lab machines, it was also hard to remember from the previous labs how to make the synthesis.

Roles of each team member and estimate time s/he worked

Name	Role	Estimated Time
Bemoi Erian Ayad	1. Ripple carry adder 2. Carry save adder 3. Synthesis of both adders 4. Floating point adder	25 hours
Peter Atef	1. Verilog (‘+’) version of adders 2. Carry increment adder 3. Synthesis of both adders 4. Testbench 5. Floating point adder	26 hours
Rufaida Kassem	1. Carry skip adder 2. Carry bypass adder 3. Synthesis of both adders 4. Constraints 5. Floating point adder	27 hours
Mark Yasser Nabil	1. Carry select adder 2. Carry look-ahead adder 3. Synthesis of both adders 4. Floating point testbensh	24 hours