Carry look ahead adder

Why another adder when ripple-carry-adder is there

As we have implemented ripple carry adder, we saw that carry-in of each one bit full adder is generated by the previous one bit adder. This causes a lot of delay especially when we increase the number of bits. To overcome this problem, a new type of adder i.e. Carry look ahead adder was needed.

Carry look ahead Adder does not wait for intermediate carries to be generated because it generates the carry-in for each bit simultaneously.

Working

From the following truth table, we can get an idea of how we can design it.

a	b	cin	cout
0	0	0	0
0	0	1	0
0	1	0	0
1	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	1

From this, we can conclude that

cout= (a & b) + ((a ^ b) & cin)

For our simplicity, lets define
- g = a & b
- p = a ^ b

Now for an 8 bit adder, we can take a 8 bit carry-in register and assign it values using the above designed logic.

Let's say it is carries[8:0] where carries[i] will be carry-in for i^{th} bit of inputs i.e a and b.

Then,

carries[0] will be carry-in itself

carries[1] = g[0]+(p[0]&carries[0]) and we can substitute for carries[0]
So,
carries[1]= g[0]+(p[0]&cin)

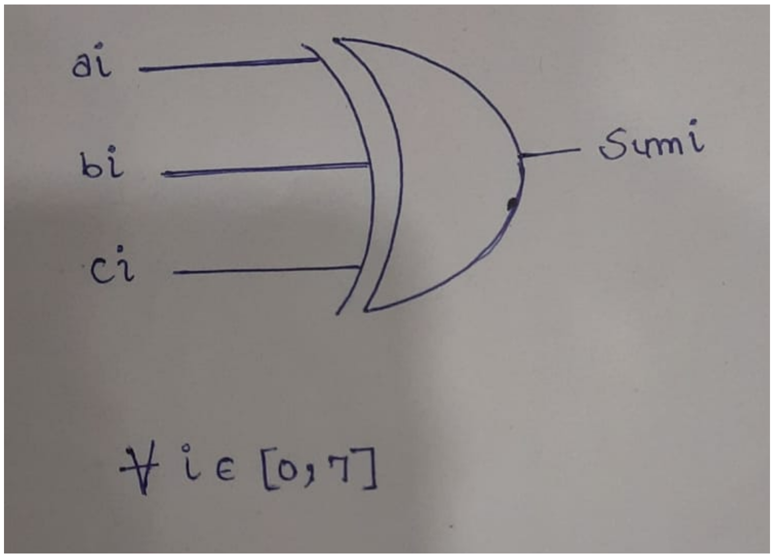
carries[2] = g[1]+(p[1]&carries[1]) , again we can substitute for carries[1]
So,
carries[2] = g[1]+(p[1]&(g[0]+(p[0]&cin)))

And so on till carries[7].

USing this method, we can also find the carry-out.

From this we can see intermediate carries can be calculated with no such propagation delay as it depends only on:
i) Bits being added
ii) Initial Carry-in

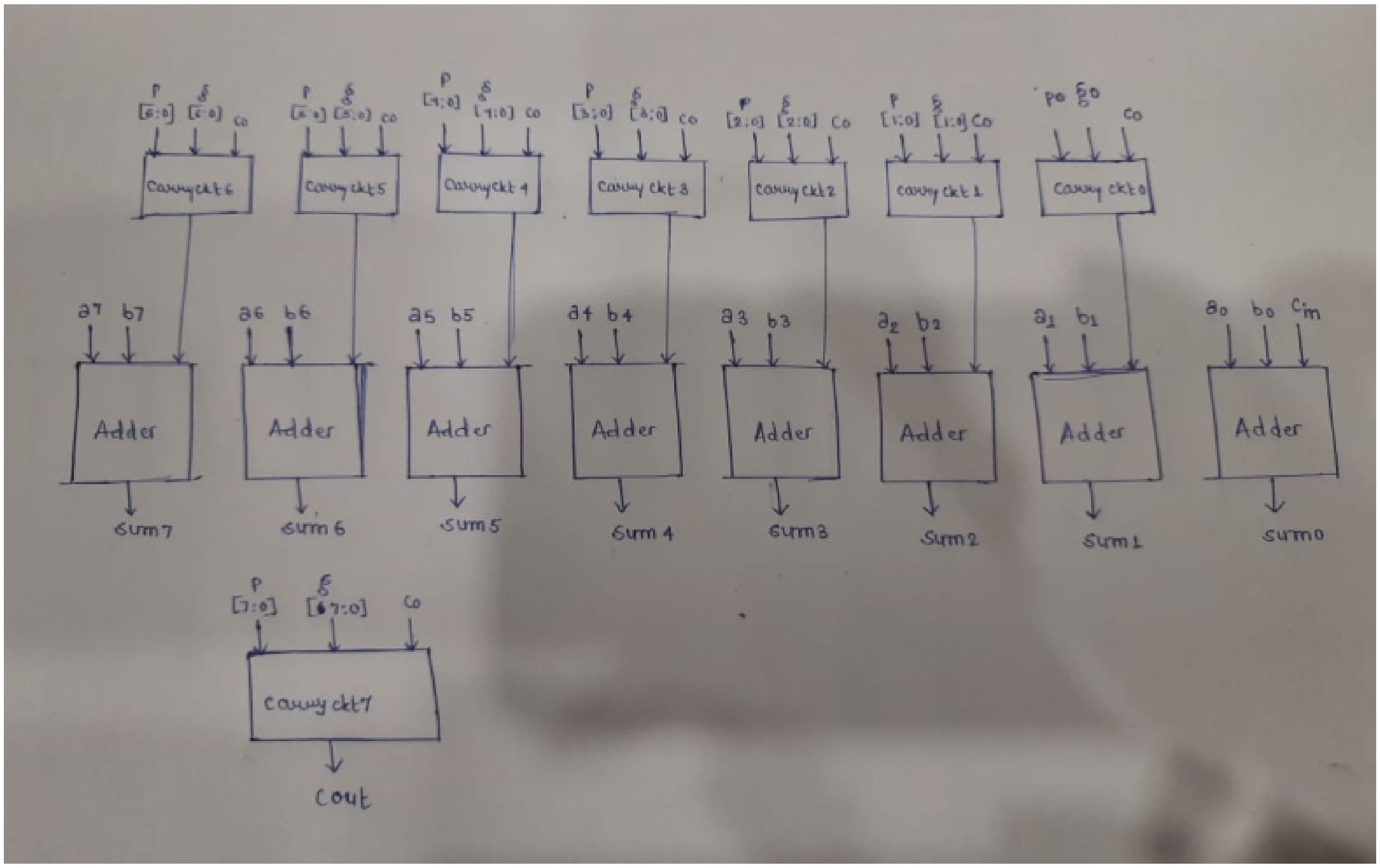
Then we can easily get the sum by using xor gates as described below:



sum[i]=a[i]^b[i]^carries[i]

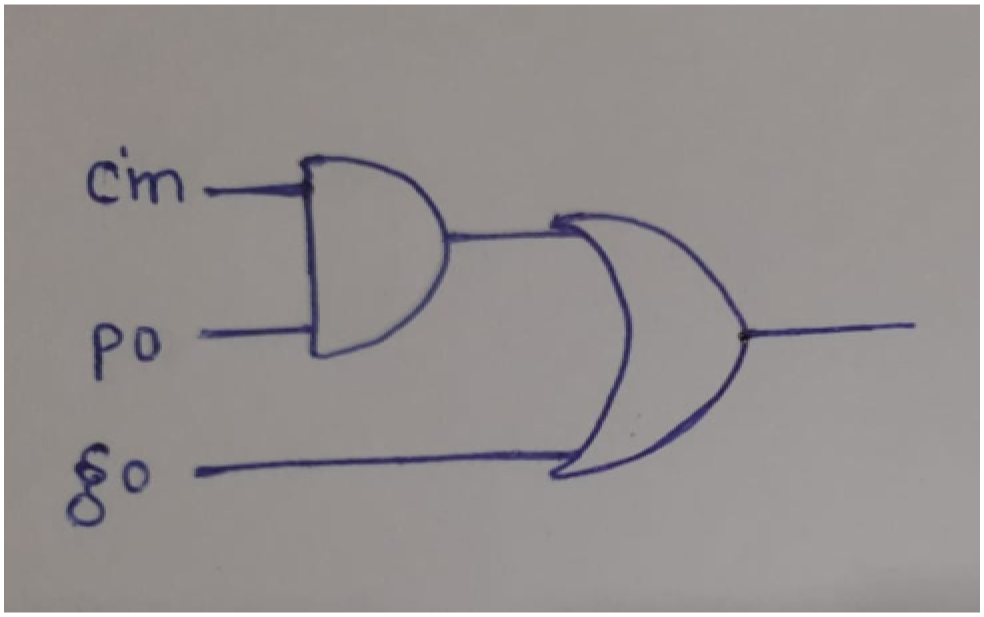
Let's also talk about some of the drawbacks of this adder:

- It gets more and more complex as we increase the number of bits.
- It involves more hardware as compared to ripple carry adder, so it is bit costly.

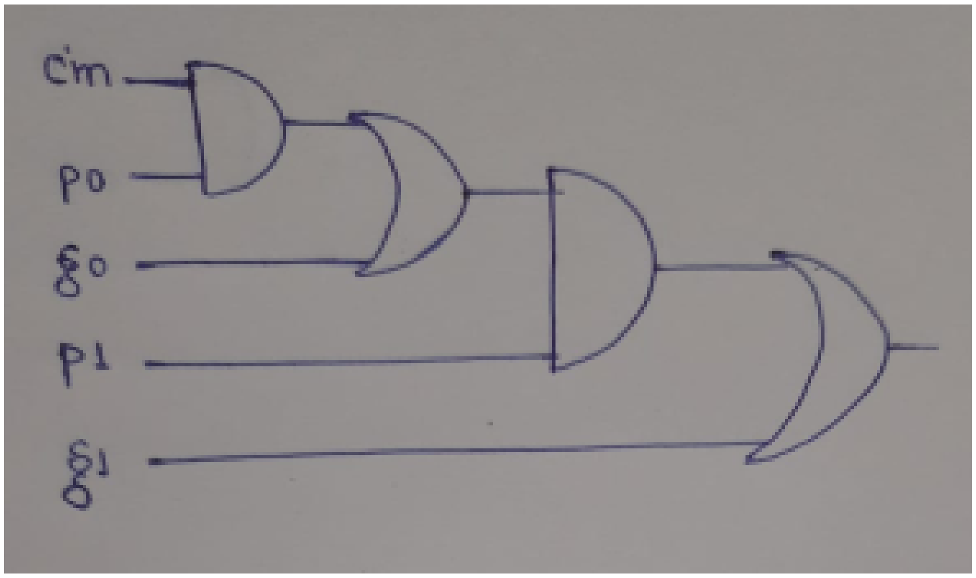


Circuit diagram

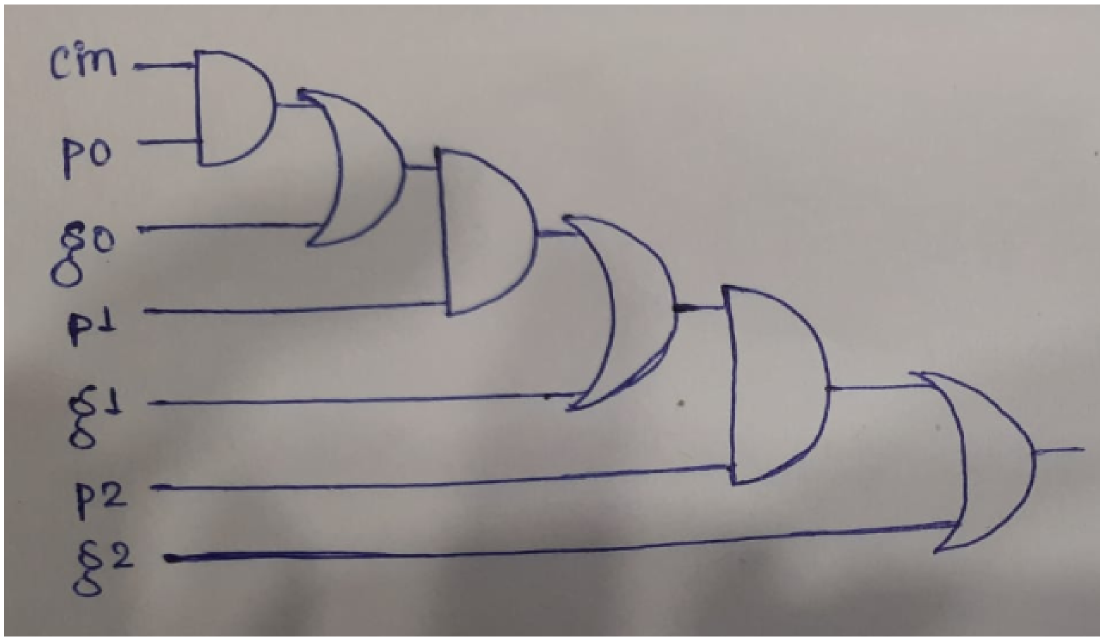
Gate level circuit diagram of different components are shown below



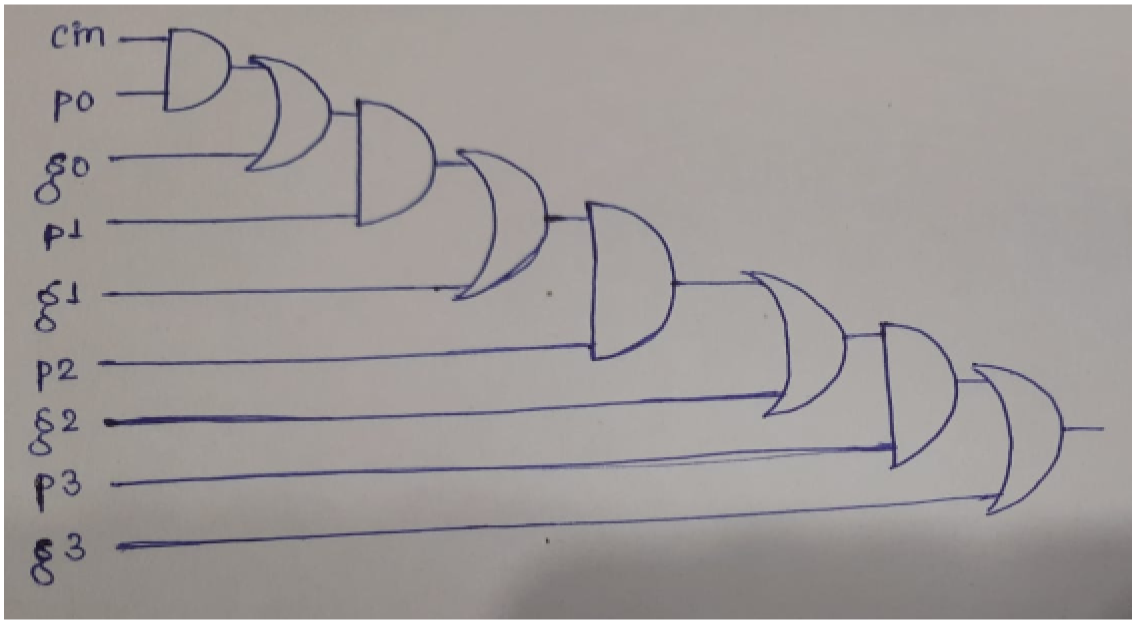
Implementation of Carry ckt 1



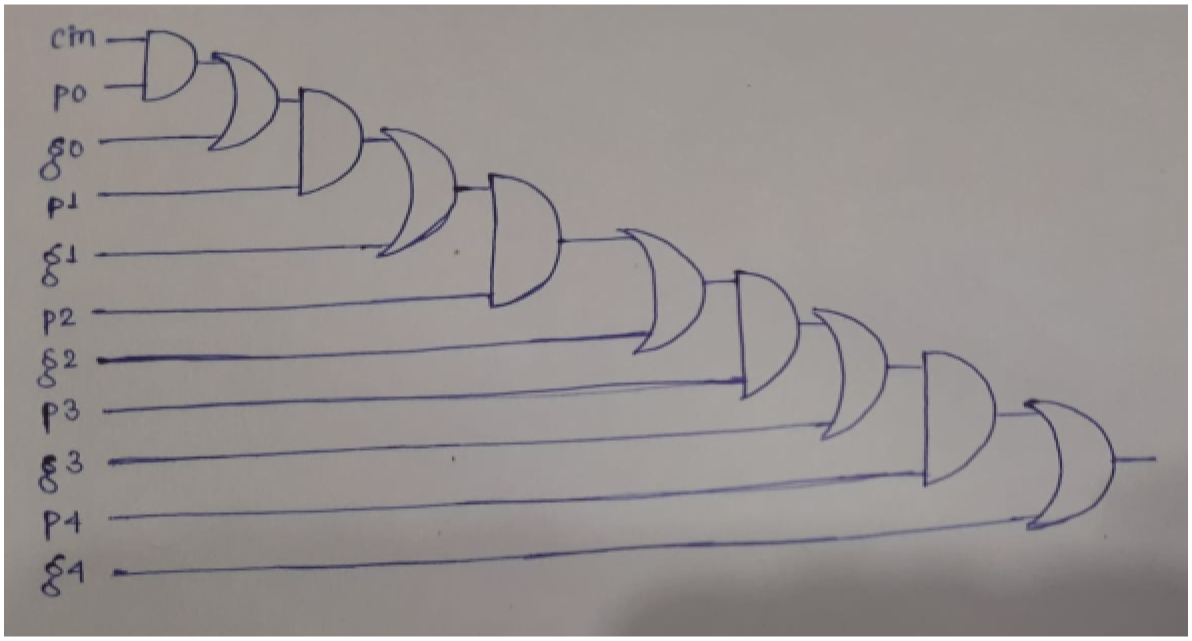
Implementation of Carry ckt 2



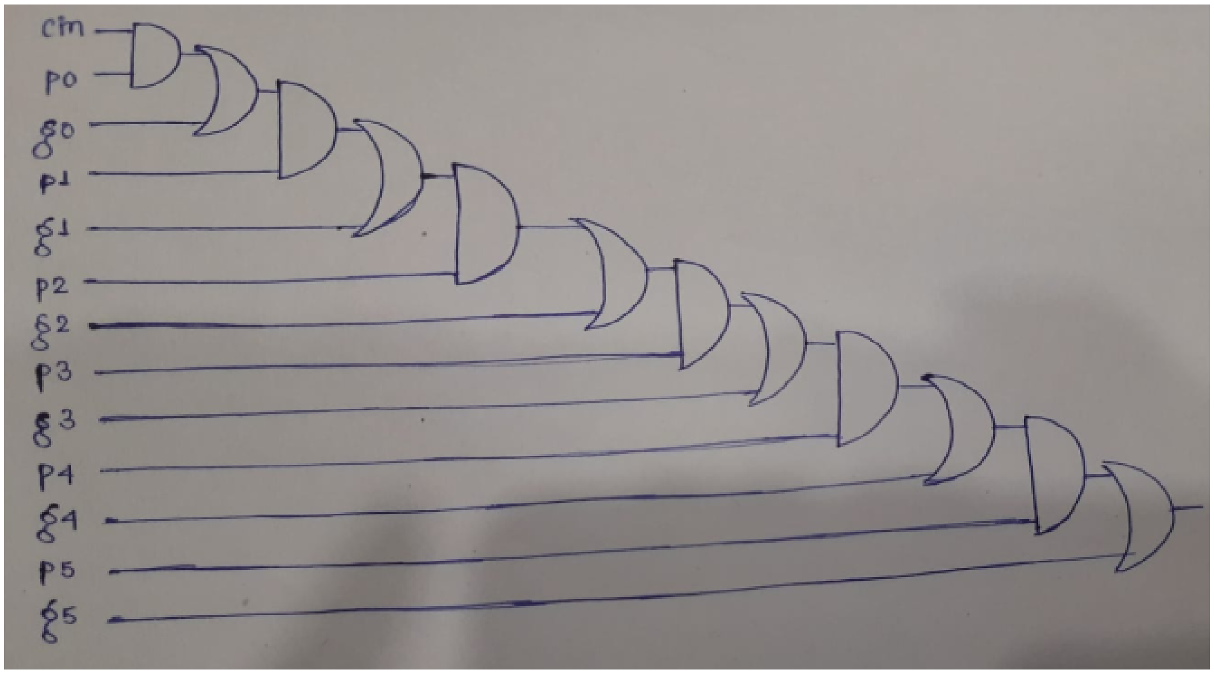
Implementation of carry ckt 3



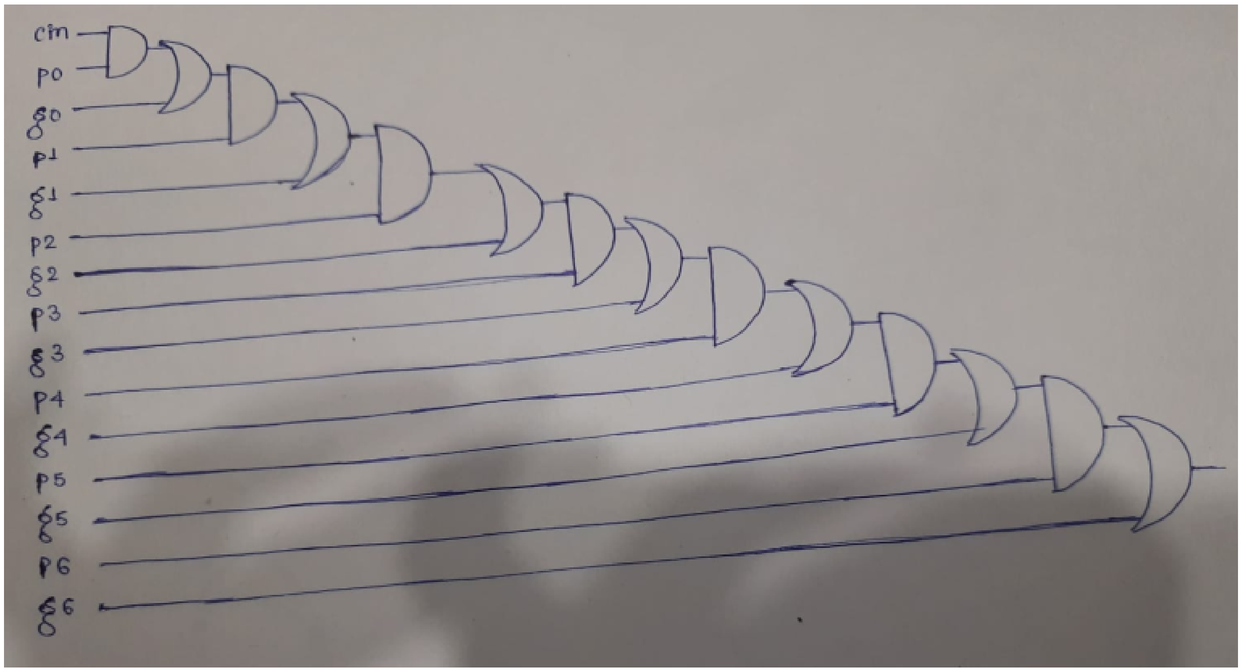
Implementation of carry ckt 4



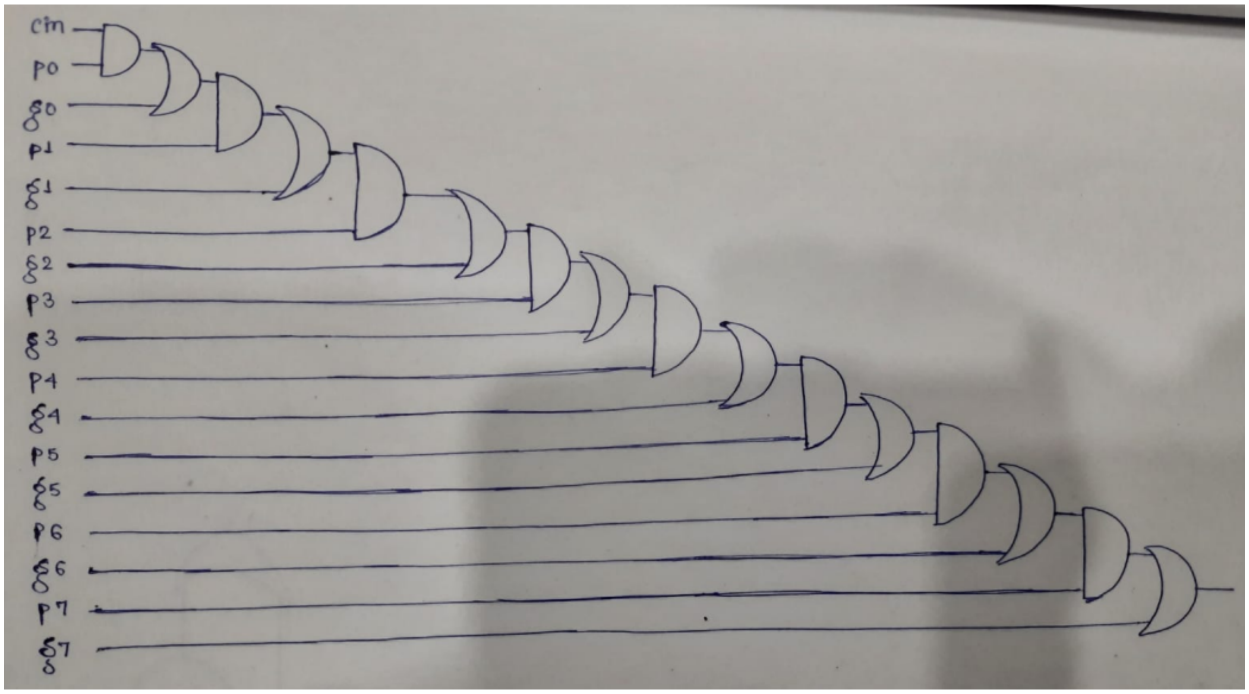
Implementation of carry ckt 5



Implementation of carry ckt 6



Implementation of carry ckt 7



Implementation of carry-out

Johnson Counter

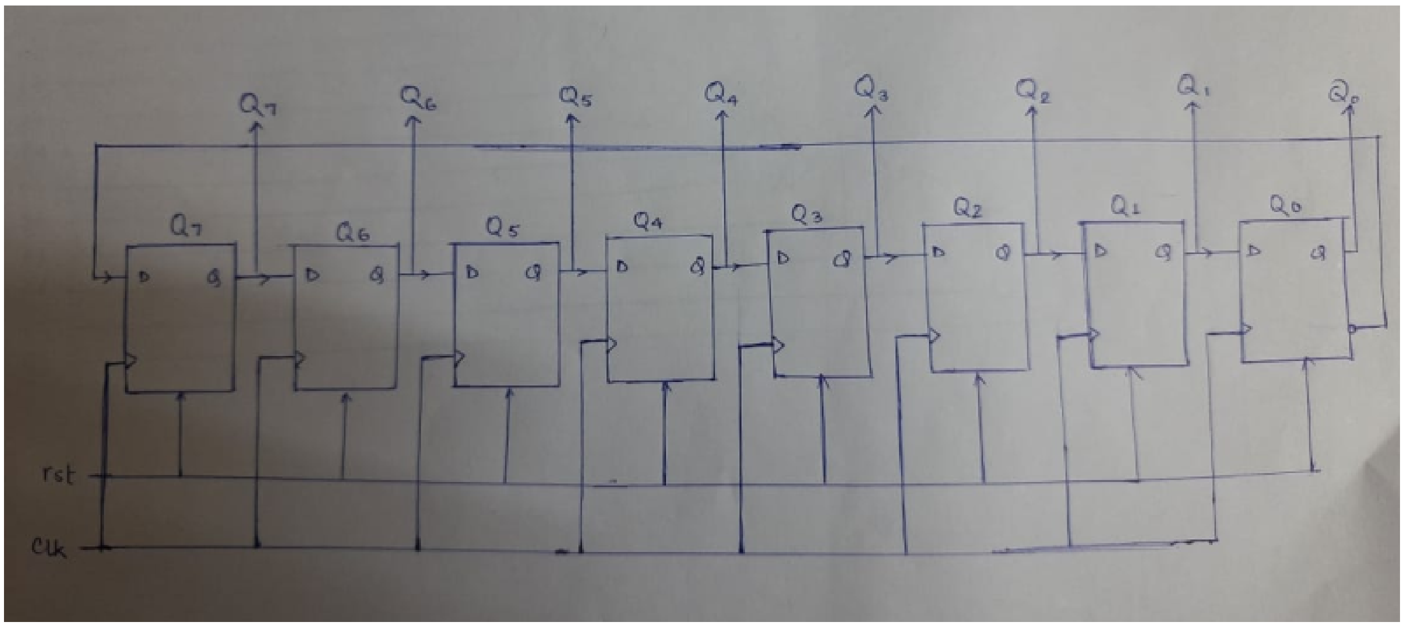
Johnson counter is like a ring counter with an advantage that a n bit johnson counter can have 2*n different states but a normal one has only n.

It is made up of n flip flops where the output of one flip flop acts as the input of next one and the input of first one is negation of the output of the last one.
The output of each flip flop makes up the total state of the counter.

Working

Initially clear is set to 1 and 8 bit output is set to 0.
Then clear is set to 0 so that we can start with the functioning of the counter. So the first flip flop get the input 1 (as output of last one is still 0).
As we are using D flip flop it will give output the same as input till the next rising edge of the clk. So we will get 10000000 as output.
Then in 2nd clock rise, we will get the 2nd flipflop's input also to be high so the output now will be 11000000
In similar fashion, after each rising edge of the clk, we will get different states depending on the previous state.

If we give the reset as 1, it will reset all the flip flop output to 0.



Circuit diagram

After keeping reset 0, this will be the truth table of this counter

Clock pulse	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0
3	1	1	1	0	0	0	0	0
4	1	1	1	1	0	0	0	0
5	1	1	1	1	1	0	0	0
6	1	1	1	1	1	1	0	0
7	1	1	1	1	1	1	1	0
8	1	1	1	1	1	1	1	1
9	0	1	1	1	1	1	1	1
10	0	0	1	1	1	1	1	1
11	0	0	0	1	1	1	1	1
12	0	0	0	0	1	1	1	1
13	0	0	0	0	0	1	1	1
14	0	0	0	0	0	0	1	1
15	0	0	0	0	0	0	0	1

From the truth table, we can see that at first, high flip flops increase from the left, then all the bits are high, after that they decrease towards the right .

Some of the drawbacks of it are:

- It does not give output in the serialized manner.
- Still many if the states are not being used.