Addition

1. Half adder

One bit addition can be accomplished with an XOR gate

(a and b are the input to the half adder)

Sum a Xor b

0	**1**	**0**	**1**
+0	+0	+1	+1
0	**1**	**1**	**10**

Carry a $\wedge$ b

0	**1**	**0**	**1**
0	0	0	+1
0	**0**	**0**	**1**

2. Full Adder

cin	**a**	**b**	**sum**
0	**0**	**0**	**0**
0	**0**	**0**	**1**
0	**1**	**0**	**1**
0	**1**	**1**	**0**
1	**0**	**0**	**1**
1	**0**	**1**	**0**
1	**1**	**1**	**0**
1	**1**	**1**	**1**

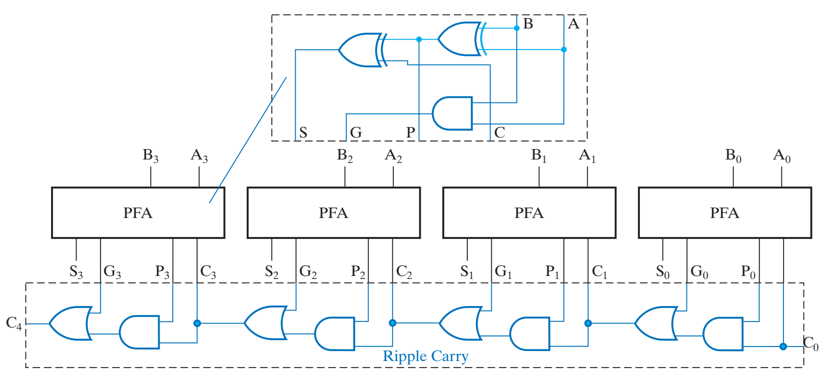
Sum = a (+) c (+) c

Cout = cin $\wedge$ (a $\vee$ b) $\vee$ (a $\wedge$ b)

You can also realize a Full Adder using two Half Adders.

3. Ripple Carry Adder

The Full adders above can be concatenated to add two n bit



(a)

From the Truth table of the the Full Adder

Cout = cin $\wedge$ (a $\vee$ b) $\vee$ (a $\wedge$ b)

In the above figure

P_i = a $\vee$ b (P indicates propagation)

and G_i = a $\wedge$ b (G indicates Generation)

With these definition we can intuitively understand $C_{out} = (P_i \wedge C_i) \vee G_i$

The Carry in from the previous stage (i -1) would be propagated to Stage i only if Pi is True.
The stage i would generate a carry irrespective of the the carry out from the stage (i -1).

Assuming each gate incurs a Unit delay (irrespective of the number of inputs to the gate and the logic it implements), the delay to get the final result Sum of all n-bits.
T(ripple-adder) = T_{FA}(x,y --> Cout) + (n-2) T_{FA}(Cin---> Cout) - T_{FA}(Cin-->Sn)

(The assumption above that each gate incurs a unite delay is is a simplified assumption. The XOR gate has more delay than a AND/OR gate. Also the delay depend on the number of inputs to the gate.)

4. Carry Look Ahead Adder.

The carry look-ahead adder is based on computing the carry bits C_i prior to the summation. The carry look-ahead logic makes use of the relationship between the carry bits C_i and the input bits A_i and B_i. We define two variables G_i and P_i, named as the generate and the propagate functions, as follows:

$$\begin{aligned} G_i &= A_i B_i \ , \\ P_i &= A_i + B_i \ . \end{aligned}$$

Then, we expand C_1 in terms of G_0 and P_0, and the input carry C_0 as

$$C_1 = A_0B_0 + C_0(A_0 + B_0) = G_0 + C_0P_0 \ .$$

Similarly, C_2 is expanded in terms G_1, P_1, and C_1 as

$$C_1 = G_1 + C_1P_1 \ .$$

When we substitute C_1 in the above equation with the value of C_1 in the preceding equation, we obtain C_1 in terms G_0, G_1, P_0, P_1, and C_0 as

$$C_1 = G_1 + C_1P_1 = G_1 + (G_0 + C_0P_0)P_1 = G_1 + G_0P_1 + C_0P_0P_1 \ .$$

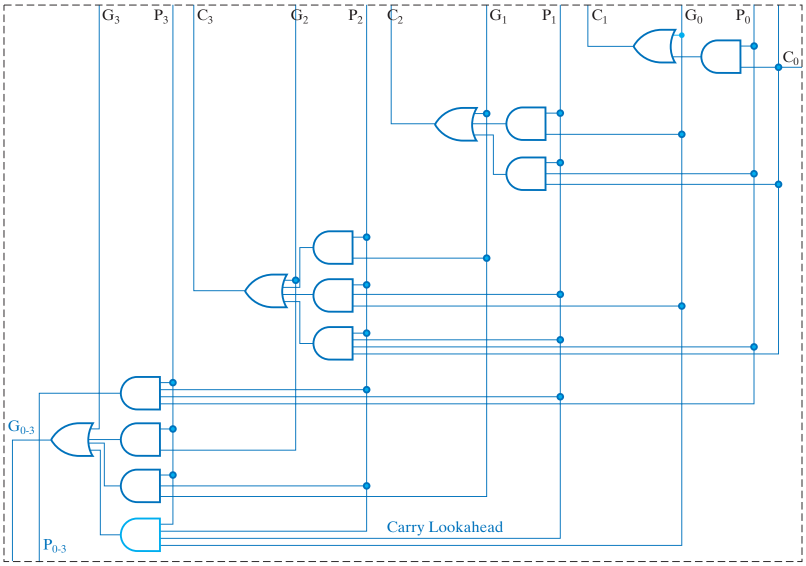
Proceeding in this fashion, we can obtain C_i as function of C_0 and $G_0, G_1, \ldots, G_i$ and $P_0, P_1, \ldots, P_i$. The carry functions up to C_4 are given below:

$$\begin{aligned} C_1 &= G_0 + C_0P_0 \ , \\ C_2 &= G_1 + G_0P_1 + C_0P_0P_1 \ , \\ C_3 &= G_2 + G_1P_2 + G_0P_1P_2 + C_0P_0P_1P_2 \ , \\ C_4 &= G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3 + C_0P_0P_1P_2P_3 \ . \end{aligned}$$

The carry look-ahead logic uses these functions in order to compute all C_is in advance, and then feeds these values to an array of EXOR gates to compute the sum vector S. The $i$the element of the sum vector is computed using

$$S_i = A_i \oplus B_i \oplus C_i \ .$$

The total delay in carry look ahead adders is much less than the ripple carry adder.
But this comes at the cost of large number of gates (which means it occupies more space and like real estate on ground, the real estate on a chip is also expensive)



(b)

Reference:
To get more intuitive feel of the Carry Look ahead adder and Ripple Carry Adders and their corresponding delay, you should spend some time with the following simulators. They model the various parameters required in designing the Adders.

http://www.ecs.umass.edu/ece/koren/arith/simulator/Add/ripple/ripple.html
http://www.ecs.umass.edu/ece/koren/arith/simulator/Add/lookahead/