16 - bit Brent kung adder

Project report

Submitted by

Dhanraj Bhukya

203070087

EE Department (EE6)

Introduction :

The Brent–Kung adder is a parallel prefix adder form of carry-lookahead adder (CLA) but it is faster than CLA. Ripple-carry adders were the initial multi-bit adders which were developed in the early days and got their name from the ripple effect which the carry made while being propagated from right to left. The time taken for addition was directly proportional to the length of the bit being added. This is reverse in Brent–Kung adders where the carry is calculated in parallel thus reducing the addition time drastically.

A Brent–Kung adder is made with an aim of minimizing the chip area and ease of manufacturing. The addition of n-bit number can be performed in time $O(\log_2 N)$ *with a chip size of area* $O(N\log_2 N)$ *thus making it a good-choice adder with constraints on area and maximizing the performance.*

Brent kung adders use the idea of carry look ahead addition. However, these do not try to implement the complex logic expressions which result from looking ahead all the way. Instead, these build up the logic in a tree like structure, where each node performs simple logic operations on the results of the previous node. Because of the tree structure used in this, the delay is of the order of log N for an N bit adder.

Implementation :

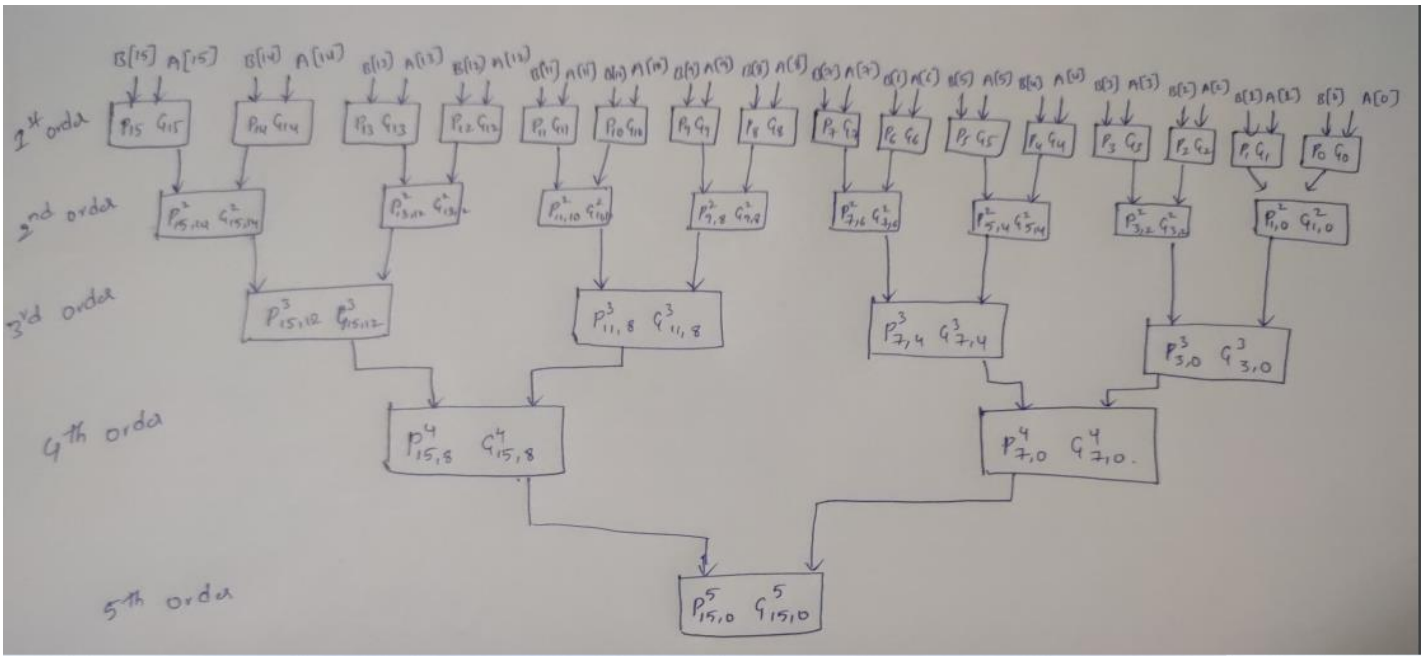
Carry equation :

$$C_{i+1} = G_i + P_i C_i$$

where : G_i (generate) $= A_i.B_i$

P_i (Propagate) $= A_i \oplus B_i$.

Values of P and G are computed in a tree fashion. The figure below shows the generation of P and G values for an 16 bit adder.



First order P and G values are computed from the below equation :

$$C_{i+1} = G_i + P_i C_i \quad \text{here} \quad i = 0 \text{ to } 15$$

Second order P and G values are computed from the below equation :

$$C_{i+1} = G^2_{i,i-1} + P^2_{i,i-1} \cdot Ci-1$$

$$G^2_{i,i-1} = G_i + P_i \cdot G_{i-1} \quad \text{and} \quad P^2_{i,i-1} = P_i \cdot P_{i-1}$$

which evaluates C_{i+1} directly from C_{i-1}

Third order P and G values are computed from the below equation :

$$C_{i+1} = G^3_{i,i-3} + P^3_{i,i-3} \cdot C_{i-3}$$

Where $G^3_{i,i-3} \equiv G^2_{i,i-1} + P^2_{i,i-1} \cdot G^2_{i-2,i-3}$ and $P^3_{i,i-3} \equiv P^2_{i,i-1} \cdot P^2_{i-2,i-3}$

we combine two second order G and P values to compute third order G and P, which will permit computation of C_{i+1} directly from C_{i-3} i.e, Carry can now be passed over groups of 4 bits

Fourth order P and G values are computed from the below equation :

using $G^3_{4i+3,4i}$ and $P^3_{4i+3,4i}$ (with i = 0 to 3) we can compute G^4 and P^4 and get carry which can be passed over 8 bits.

Using G^4 and P^4 we can compute G^5 and P^5 and obtain final carry ..

All these computed carries are used to find the sum by just performing xor operation with P_i i.e,

$$Sum = P_i \oplus C_i$$

The intermediate carries are found as follows :

Note : i)Here g,p represents first order

ii) g2,p2 represents seond order and simillarly remaining orders.

```
c[1] = g[0] | (p[0] & c[0]);
c[2] = g2[0] | (p2[0] & c[0]);
c[4] = g3[0] | (p3[0] & c[0]);
c[8] = g4[0] | (p4[0] & c[0]);
C[16] = g5 | (p5 & c[0]);
c[3] = g[2] | (p[2] & c[2]);
C[5] = g[4] | (p[4] & c[4]);
c[6] = g2[2] | (p2[2] & c[4]);
c[7] = g[6] | (p[6] & c[6]);
c[9] = g[8] | (p[8] & c[8]);
c[10] = g2[4] | (p2[4] & c[8]);
c [11] = g[10] | (p[10] & c[10]);
c [12] = g4[1] | (p4[1] & c[4]);
c[13] = g[12] | (p[12] & c[12]);
c[14] = g[13] | (p[13] & c[13]);
c[15] = g[14] | (p[14] & c[14]);
```

The carry comutation is faster than CLA in brent kung adder.

Simulated Results :