16-BIT - BRENT-KUNG-ADDER.

Brent kung adder is a logarithmic tree adder which uses the scheme of carry look ahead addition. But unlike carry look ahead adder, it does not look ahead all the way to the last bit in a single step because the complexity and delay will be more. Instead it ~~ues.~~ uses propagation and generation values in a tree like fashion where in each row we will have $2^n, 2^{n-1}, 2^{n-2}$ ... upto 2^0 blocks where $n = \log_2$ (no. of bits in adder).

First, we will know, what carry look adder means.

Suppose, we have A, B, cin bits to be added

If $A=1, B=1$ irrespective of cin, cout will be 1. So it is generation $G = A \cdot B$. If $B, A = 1, 0$ or $0, 1$ the input carry should be propagated to Cout. So it is propagation $P = A \oplus B$. if $A=0, B=0$ $Cout = 0$ irrespective of cin. So it is carry kill. $K = \bar{A}\bar{B}$ but it is not needed really.

So in carry look ahead we can write the next carry like $C_{i+1} = G_i + P_i C_i$ → 1st order

$$C_{i+1} = G_i + P_i(G_{i-1} + P_{i-1} C_{i-1}) = (G_i + P_i G_{i-1}) + (P_i \cdot P_{i-1}) C_{i-1}$$

C_{i+1} can be selected by C_{i-1}. So P, G here are 2nd order.

These terms will be cleared by the following diagram.

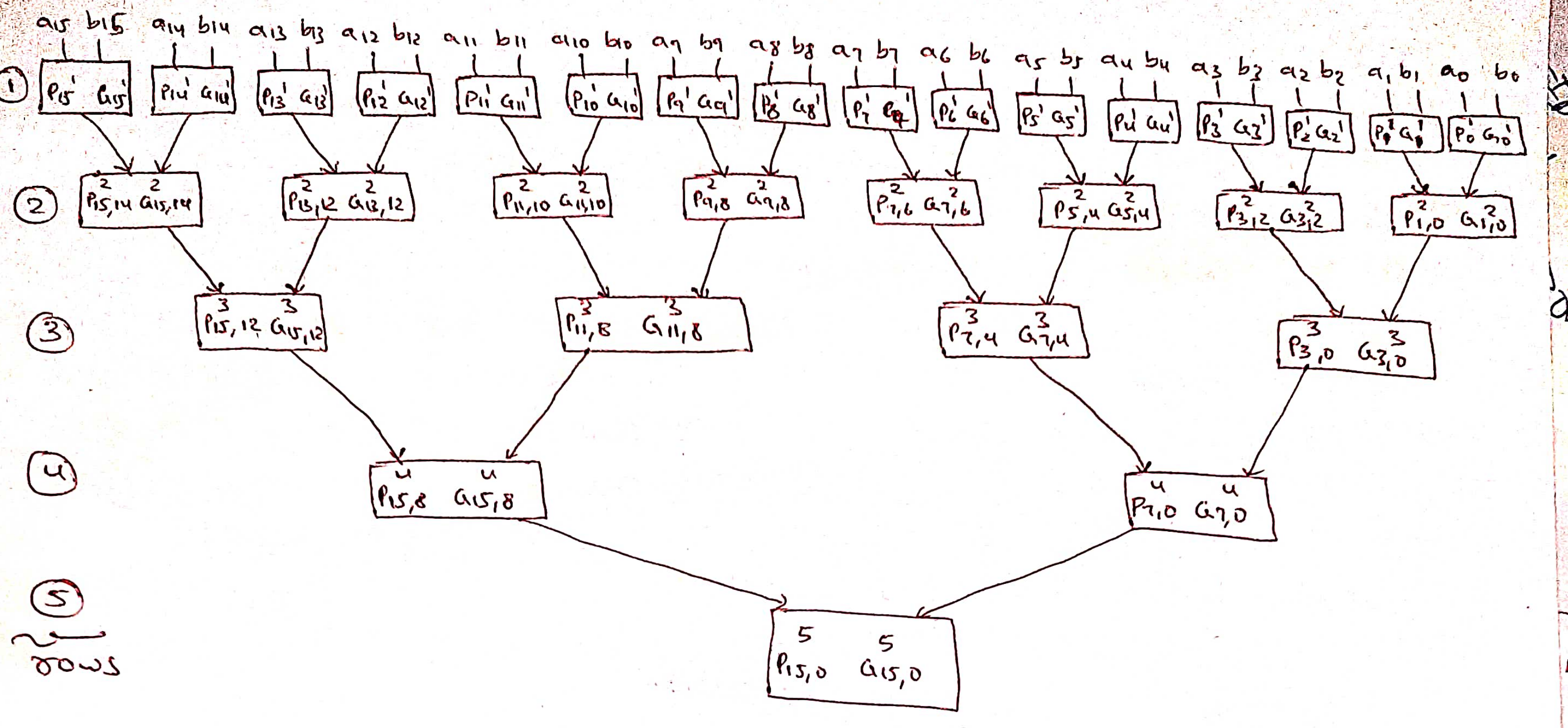


Fig1: Logarithmic propagation and generation computation

In the above figure, we can see for 16 bit brent kung adder propagation and generation signals are computed in logarithmic blocks in a tree fashion i.e; $2^n, 2^{n-1}, 2^{n-2}, 2^{n-3}, 2^{n-4} \dots 2^0$ where $n = \log_2(\text{no. of bits})$ ~~log no of bits~~. so basically we get from downwards tree like $2^0, 2^1, 2^2, 2^3$ and so on upto the no. of bits to be added. There are in total 5 tree stages in the above figure.

1st row → 1st order Propagation (P) and generation (G) values

2nd row → 2nd order P, G, values.

3rd row → 3rd order P, G values

4th row → 4th order P, G values

5th row → 5th order P, G values.

Basically, we know carry look ahead generation uses the expression

$$c_{i+1} = P_i c_i + G_i = G_i + P_i c_i$$

but c_i itself is carry from previous generated carry so we can write c_{i+1} as

$$c_{i+1} = G_i + P_i \underbrace{(G_{i-1} + P_{i-1} c_{i-1})}_{c_i}$$

$$c_{i+1} = \underbrace{(G_i + P_i G_{i-1})}_{\substack{2^{nd}\text{ order} \\ \text{generation}}} + \underbrace{(P_i P_{i-1})}_{\substack{2^{nd}\text{ order} \\ \text{propagation}}} c_{i-1}$$

so by using 2^{nd} order P, G values we can compute c_{i+1} from c_{i-1} directly without using c_i. That means we can skip internal carries if we have the highest order P, G values. And here the beauty is all P, G values of every order is independent of input carries and can be computed even before the carry arrives. The carry is always in the critical path as it has to ripple through all the blocks in ripple carry adder. So that is why we are more concerned about carry rather than sum in Brent kung adder. if all carries are generated, we can compute sum by $\boxed{sum = A_i \oplus B_i \oplus c_i}$.

In this project, delay of Brent_kung_adder is compared with ripple carry adder critical delay.

once, all P, G terms of various order are known, we can compute the carry outputs which depend on C_0 available at $t=0$.

$$C_1 = G_0^1 + P_0^1 C_0$$ → here 1st order because C_{i+1} is computing from C_i

$$C_2 = G_{1,0}^2 + P_{1,0}^2 C_0$$ → 2nd order because we are skipping C_1

$$C_4 = G_{3,0}^3 + P_{3,0}^3 C_0$$ → 3rd order because C_1, C_2, C_3 are skipped

$$C_8 = G_{7,0}^4 + P_{7,0}^4 C_0$$

$$C_{16} = G_{15,0}^5 + P_{15,0}^5 C_0$$

So basically if we want to compute C_n we get logarithmic carries like $C_1, C_2, C_4, C_8, C_{16}$ from C_0 by using 1st, 2nd, 3rd, 4th, 5th order P, G respectively.

After these $C_3, C_5, C_9, C_6, C_{10}, C_{12}$ can be generated from $C_2, C_4, C_8, C_4, C_8, C_8$ respectively by using 1st order, 1st order, 1st order, 2nd order, 2nd order, 3rd order respectively.

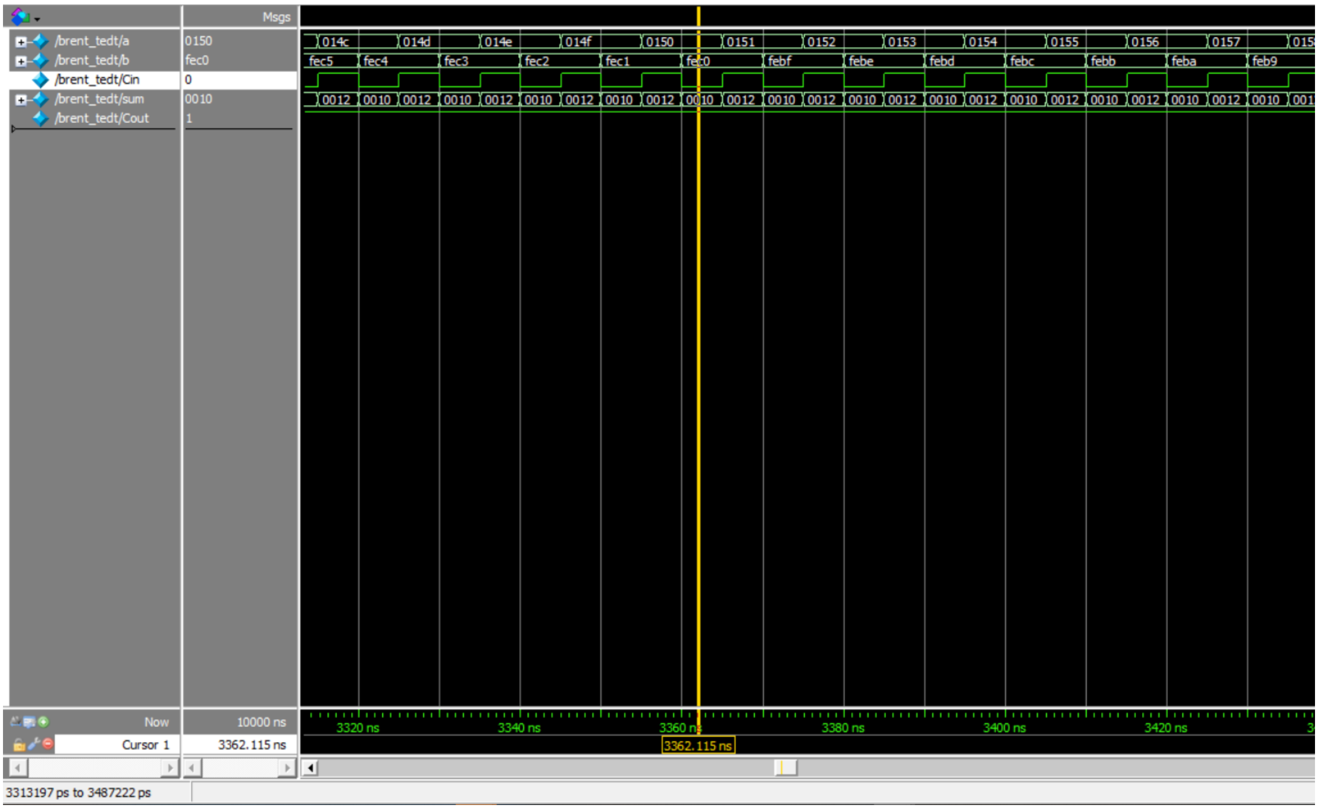
After this, we ~~get~~ compute $C_7, C_{11}, C_{13}, C_{14}$ from $C_6, C_{10}, C_{12}, C_{12}$ by using 1st order, 1st order first order, 2nd order respectively.

At last we can compute C_{15} from C_{14} from 1st order. These are one after another as some are dependant on others.

By generating all carries, corresponding sums can be generated by $sum_i = P_i^1 \oplus C_i$.

Note: we can't generate all carries at same time because some internal carries are dependant on other. see the fig 1 for understanding.

Waveforms for Brent kung adder



Maximum combinational Delay for Brent kung adder

```
Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 633 / 17
-------------------------------------------------------------------------
Delay:               18.582ns (Levels of Logic = 11)
  Source:            a<4> (PAD)
  Destination:       Cout (PAD)

  Data Path: a<4> to Cout
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             7   0.715   1.201  a_4_IBUF (a_4_IBUF)
     LUT2:I0->O            1   0.479   0.740  Carry_8_or000054 (Carry_8_or000054)
     LUT4:I2->O            1   0.479   0.976  Carry_8_or000063 (Carry_8_or000063)
     LUT4:I0->O            3   0.479   0.794  Carry_8_or000076 (Carry_8_or000076)
     LUT4:I3->O            1   0.479   0.851  Carry_12_or000011_SW0 (N32)
     LUT4:I1->O            1   0.479   0.740  Carry_12_or000031_SW1 (N42)
     LUT3:I2->O            4   0.479   1.074  Carry_12_or000031 (Carry<12>)
     LUT4:I0->O            1   0.479   0.851  Carry_16_or000011_SW0 (N30)
     LUT4:I1->O            1   0.479   0.740  Carry_16_or000031_SW1 (N40)
     LUT3:I2->O            1   0.479   0.681  Carry_16_or000031 (Cout_OBUF)
     OBUF:I->O                 4.909          Cout_OBUF (Cout)
    ----------------------------------------
    Total                     18.582ns (9.935ns logic, 8.647ns route)
                                       (53.5% logic, 46.5% route)
```

The delay for 16 bit ripple carry adder is also provided to compare how fast Brent kung adder is.

Maximum combinational Delay for ripple carry adder

```
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 321 / 17
-------------------------------------------------------------------------
Delay:               28.741ns (Levels of Logic = 18)
  Source:            b<0> (PAD)
  Destination:       cout (PAD)

  Data Path: b<0> to cout
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             2   0.715   1.040  b_0_IBUF (b_0_IBUF)
     LUT3:I0->O            2   0.479   0.915  ripple_gen[0].f1/cout1 (carry<1>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[1].f1/cout1 (carry<2>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[2].f1/cout1 (carry<3>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[3].f1/cout1 (carry<4>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[4].f1/cout1 (carry<5>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[5].f1/cout1 (carry<6>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[6].f1/cout1 (carry<7>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[7].f1/cout1 (carry<8>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[8].f1/cout1 (carry<9>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[9].f1/cout1 (carry<10>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[10].f1/cout1 (carry<11>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[11].f1/cout1 (carry<12>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[12].f1/cout1 (carry<13>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[13].f1/cout1 (carry<14>)
     LUT3:I1->O            2   0.479   0.915  ripple_gen[14].f1/cout1 (carry<15>)
     LUT3:I1->O            1   0.479   0.681  ripple_gen[15].f1/cout1 (cout_OBUF)
     OBUF:I->O                 4.909          cout_OBUF (cout)
    ----------------------------------------
    Total                     28.741ns (13.288ns logic, 15.453ns route)
                                       (46.2% logic, 53.8% route)
```

From the dealys,we can see Brent kung adder is fast compared to ripple carry adder.

The device specifications used for both synthesis of ripple carry adder and brent kung adder are given below