# Chapter 4

# Final Adder in Multiplication

The previous two chapters presented several methods for generating the partial products and accumulating (reducing) the generated partial products. Once the partial product reduction tree using compressors reduces all the partial products into two rows of sum and carry, a fast final adder is needed to get the result.

Several techniques have been developed to reduce the propagation delay of the final adder[Ling 1981, Stelling 1996, Chan 1992]. The use of a fast carry propagate adder is most widely used in implementation. So in this chapter the best carry-lookahead adder (CLA) structure [Hennesy 2000] is discussed which will be used in the final adder stage. A new addition technique using RB arithmetic is also discussed. This adds sum row and carry row to give the result in RB. It again converts it back to NB form using novel RB to NB converter. This can act as the fastest adder as well as RB to NB converter and hence reduce the delay.

## 4.1 Carry-lookahead adder

The most commonly used scheme for accelerating carry propagation is the carry-lookahead scheme. The main idea behind carry-lookahead addition is an attempt to generate all incoming carries in parallel and avoid the need to wait until the correct carry propagates from the stage of the adder where it has been generated. This can be accomplished in principle, since the carries generated and the ways they propagate depend only on the digits of the original numbers $a_{n-1}$, $a_{n-2}$,……, $a_0$ and $b_{n-1}$, $b_{n-2}$,……, $b_0$.

There are stages in the adder in which a carryout is generated regardless of the incoming carry, and as a result, no additional information on previous input digits is required. These are the stages where $a_i = b_i = 1$. There are other stages that are only capable of propagating the incoming carry i.e. where $a_i = 0$ and $b_i = 1$ or $a_i = 1$ and $b_i = 0$. Only a stage in which $a_i = b_i = 0$ cannot propagate a carry. To assimilate the information regarding the generation and propagation of carries, two functions can be defined as:

$$\text{Generate carry} = g_i = a_i \cdot b_i \tag{4.1}$$

$$\text{Propagate carry} = p_i = a_i \oplus b_i \tag{4.2}$$

As a result, the Boolean expression for the carry out can be written as

$$c_{i+1} = a_i \,.b_i + c_i\,(a_i \oplus b_i) = g_i + c_i \,.\, p_i \tag{4.3}$$

Substituting $c_i = g_{i-1} + c_{i-1}.\, p_{i-1}$ in the above expression yields

$$c_{i+1} = g_i + g_{i-1}.\, p_i + c_{i-1} \,.\, p_{i-1} \,.\, p_i \tag{4.4}$$

Further substitutions will result in

$$c_{i+1} = g_i + g_{i-1} \,.\, p_i + \text{-----} + c_0 \,.\, p_0 \,.\, p_1 \,\ldots\ldots p_i \tag{4.5}$$

This expression will allow us to calculate all the carries in parallel from the original bits. Using this principle a tree structure of a 8-bit carry-lookahead adder is shown in Fig. 4.1. The adder performs addition in two parts. In the first part, propagate and generate signals are derived from the operands (from sum and carry row in our case) applied to each A-cell at the first level of the tree structure, for the $i^{th}$ A-cell using equation 4.1 and 4.2.

These values are then used to find the group propagate and generate ($P_{01}$ and $G_{01}$) values for group of two successive A-cells at the second level of the tree (comprising of B-cells) using, for the first group, equations:

$$G_{01} = g_1 + p_1 \,.\, g_0 \tag{4.6}$$

$$P_{01} = p_0 \,.\, p_1 \tag{4.7}$$

$P_{01}$ and $G_{01}$ contain the information regarding the propagation through or generation of carry in the group comprising of the first and second A-cells.

The values of ($G_{01}$, $P_{01}$) and ($G_{23}$, $P_{23}$) are then used by another B-cell at the third level to find the values of group generate and propagate functions ($G_{03}$, $P_{03}$) for the group of four successive cells. Similarly, values of $G_{07}$ and $P_{07}$ are found by combining ($G_{03}$, $P_{03}$) and ($G_{47}$, $P_{47}$) at the fourth level of the tree structure.

In the second part, the input carry, '$c_0$' is fed to B-cell at the bottom of the tree structure. B-cell, at the fourth, third and second level generate $c_4$, $c_2$ and $c_1$ signals simultaneously with $c_0$ as one of the inputs using equations

$$\begin{aligned} c_1 &= g_0 + p_0.c_0 \\ c_2 &= G_{01} + P_{01}.c_0 \\ c_4 &= G_{03} + P_{03}.c_0 \end{aligned} \tag{4.8}$$

Similarly other carries $c_6$, $c_5$, $c_3$ are generated at the same time using a similar logic. Carry $c_7$ is obtained using $c_6$, $g_6$ and $p_6$ as the inputs.

An extra logic can be used if required to compute the carryout $c_8$ of the 8 bit adder using equation $c_8 = g_7 + p_7 . c_7$

The sum signals are generated by individual A-cells from operands $a_i$, $b_i$ and $c_i$ using equations

$$p_i = a_i \oplus b_i$$

$$s_i = p_i \oplus c_i$$

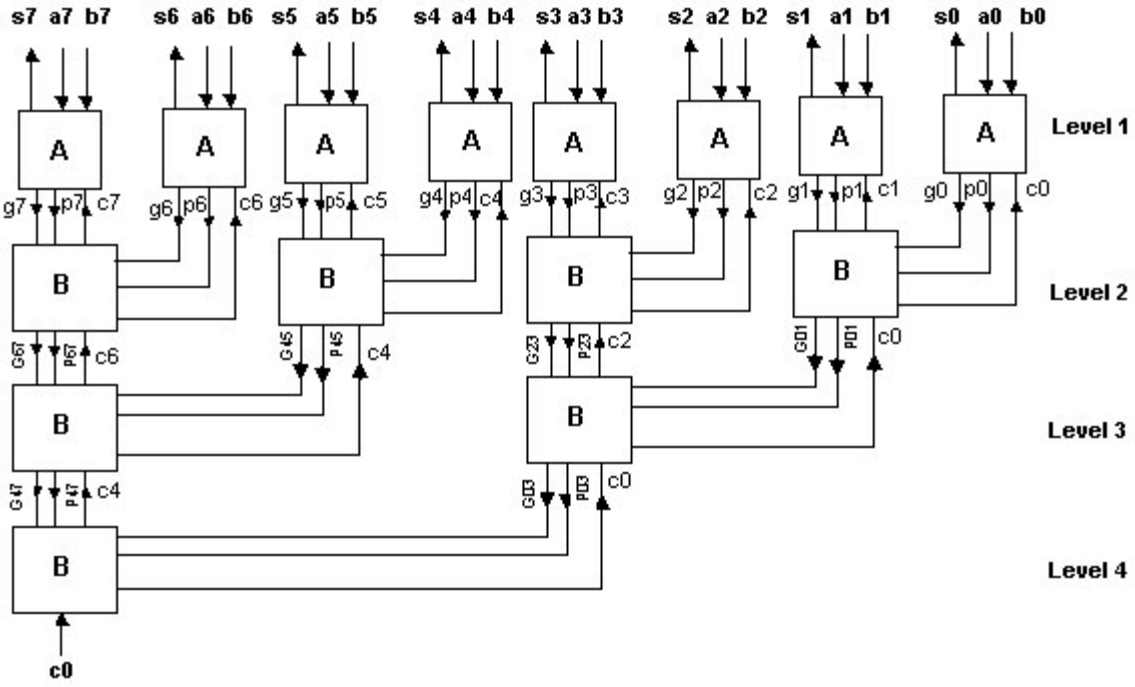



Fig. 4.1 8-bit carry-lookahead adder

For delay analysis we will find the delay of A-cell and B-cell in normalized form and use these delays to calculate the worst path delay of a complete CLA. In Fig. 4.2(a), the A-cell shows that the worst-case delay is $T_{XOR2}$=1, required to obtain p. Again in the reverse direction, the worst case delay is in getting the sum from c and $p_i$ and is $T_{XOR2}$=1. Similarly in Fig. 4.2 (b) for B-cell the worst case delay for getting group generate and group propagate is $T_{AND2} + T_{OR2} = 1.225$. In the reverse way to get the higher level of carry from input carry using p and g is $T_{AND2} + T_{OR2} = 1.225$.

For addition of two n-bit numbers they have to flow from top to bottom to obtain the group propagate and group generate values and again from bottom to top to get all the carries. So the worst delay path for this 8-bit CLA is given below:

(a0,b0),(a1, b1) →(g0,p0),(g1,p1) → (G01,P01),(G23,P23) → (G03,P03) → c4 → c6 → c7 →s7

So using the delay analysis of cell A and cell B the worst-case delay will be 8.125.

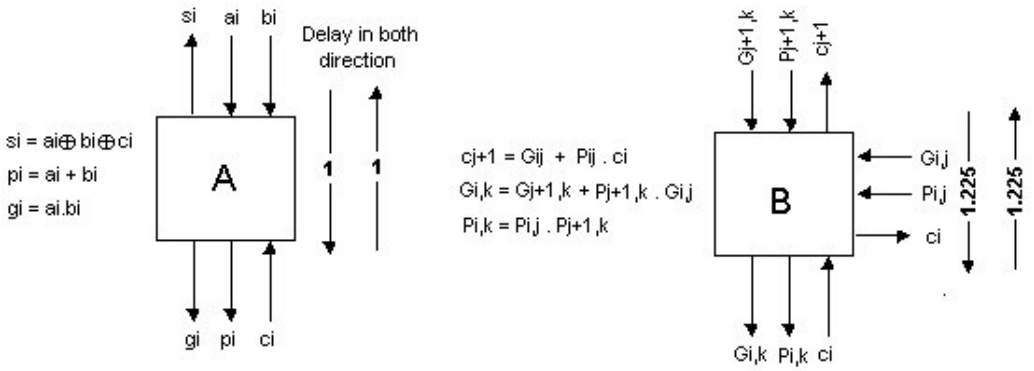



Fig. 4.2 Delay analysis of basic cells used in CLA (a) A- cell (b) B- cell

The structure of an 8-bit CLA can be extended to implement larger CLA like 16-bit, 32-bit, 64-bit and 128-bit. In extending an 8-bit CLA to 16-bit the number of levels increases by one B-cell resulting in an additional delay of 2.550. In the same way the delay analysis for different sizes of CLA are done and summarized in Table 4.1.

Table 4.1 Worst-case delay of different sizes of CLA adder in terms of normalized gate delay

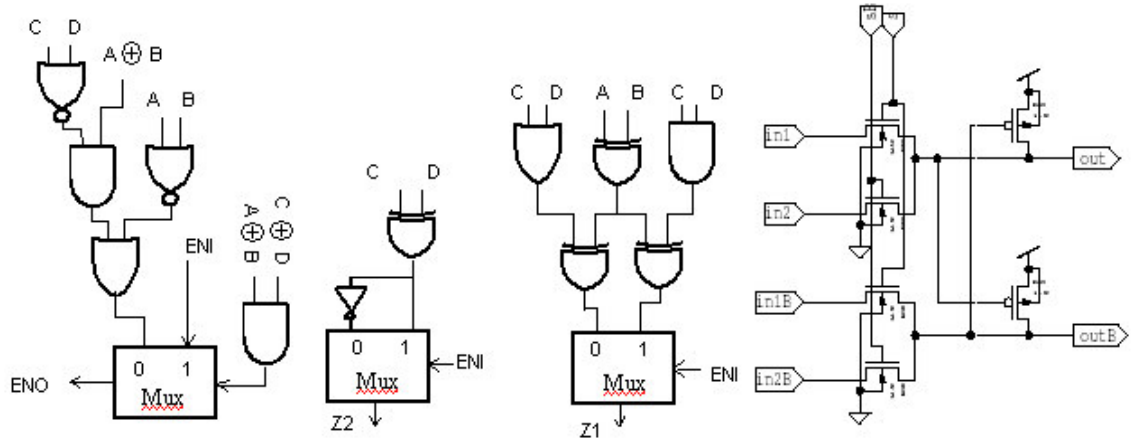
| CLA adder operand size | Worst case delay in terms of $T_{XOR}$ |
|---|---|
| 8-bit | 8.125 |
| 16-bit | 10.575 |
| 32-bit | 13.025 |
| 64-bit | 15.475 |
| 128-bit | 17.925 |

## 4.2 Equivalent binary converter

In chapter 3 (section 3.1, 3.3, 3.4), the accumulation methods like Array method and Wallace tree method using 3:2 or 4:2 compressors give the accumulation results in the form of a sum row and a carry row. Use of RB adder (in section 3.5) in the compressor tree is giving the result in RB form, which needs to be converted to NB form. For this a number of researchers have proposed different techniques [Ercegovac 1987, Kim 2003]. In this section

we will discuss a novel architecture based on equivalent binary conversion algorithm (EBCA) [Kim 2001, Kim 2003]. This can act as the fastest RB to NB converter as well as the fastest adder for word lengths upto 16 bits. This converter will be used for a novel carry-lookahead equivalent bit converter (CLEBC) in the next section, which can act as the fastest adder and RB to NB converter for any operand size.

The EBCA for RB to NB conversion is discussed in section 1.2 and in Table 1.4. Thorough analysis of the table shows that ENO is independent of ENI except for X=0 and Y=0. If X=0 and Y=0, in such condition ENO=ENI. Taking this into consideration we have designed the circuit for generating ENO as shown in Fig. 4.3(a). Here ENI is given as one of the input to the 2:1 multiplexer whose selection line is activated from X and Y. So as X and Y are available at time t=0, selection of which input is to be passed is predetermined. If X=0 and Y=0 then ENI input is just passed to give the ENO output. To pass ENI we have used n-channel pass transistor multiplexer as shown in Fig. 4.3(b). Use of this novel equivalent bit converter (EBC) circuit for 8-bit RB to NB conversion is shown in Fig. 4.4. This is giving much better performance than 8-bit CLA implementation. The performance evaluation of this EBC with respect to fastest CLA structure (discussed in previous section) is shown in Table 4.2. The plot for delay, power and EDP with decreasing channel length is shown in Fig. 4.5. This indicates that the new circuit of block size 8 is 4 times faster than the CLA of block size 8. The power consumption is also 30% less than that of the CLA along with a smaller transistor count.




(a) (b)

Fig. 4.3 (a) Circuit of RB to NB converter. Here A = $X^+$, B = $\overline{X^-}$, C = $Y^+$, D = $\overline{Y^-}$ (b) Multiplexer

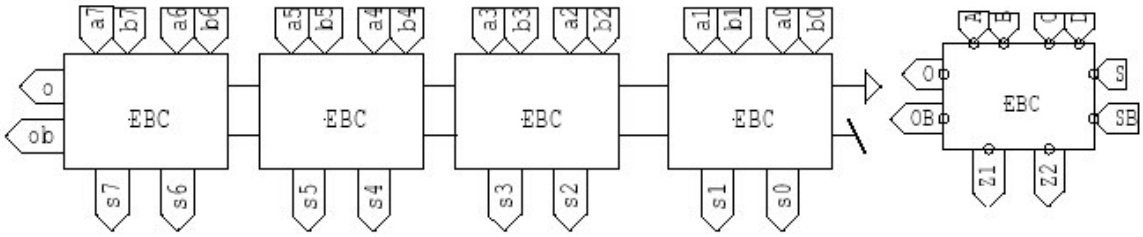



Fig. 4.4 8-bit equivalent bit converter

Table 4.2 Comparisons of CLA and EBC

| Adder type | No,Of Tr used | Delay (Normalized) | | | | | | Power (Normalized) | | | | | | EDP(Normalized) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1. 2 | 1. 3 | 1. 4 | 1. 5 | 1. 6 | 1. 7 | 1. 2 | 1. 3 | 1. 4 | 1. 5 | 1. 6 | 1. 7 | 1. 2 | 1. 3 | 1. 4 | 1. 5 | 1. 6 | 1. 7 |
| 8-bit CLA | 510 | 4.13 | 4.26 | 4.38 | 4.51 | 4.65 | 4.80 | 1.45 | 1.46 | 1.48 | 1.49 | 1.52 | 1.54 | 6.02 | 6.24 | 6.49 | 6.77 | 7.07 | 7.41 |
| 8-bit EBCA | 424 | 1.00 | 1.01 | 1.03 | 1.05 | 1.07 | 1.09 | 1.00 | 1.01 | 1.03 | 1.06 | 1.08 | 1.11 | 1.00 | 1.03 | 1.07 | 1.11 | 1.16 | 1.22 |
| 16-bit CLA | 1038 | 1.57 | 1.62 | 1.67 | 1.72 | 1.77 | 1.83 | 0.85 | 0.87 | 0.88 | 0.90 | 0.93 | 0.95 | 1.35 | 1.41 | 1.48 | 1.56 | 1.65 | 1.74 |
| 16-bit EBCA | 848 | 1.00 | 1.03 | 1.08 | 1.15 | 1.24 | 1.34 | 1.00 | 1.01 | 1.03 | 1.06 | 1.10 | 1.13 | 1.00 | 1.04 | 1.12 | 1.23 | 1.36 | 1.52 |

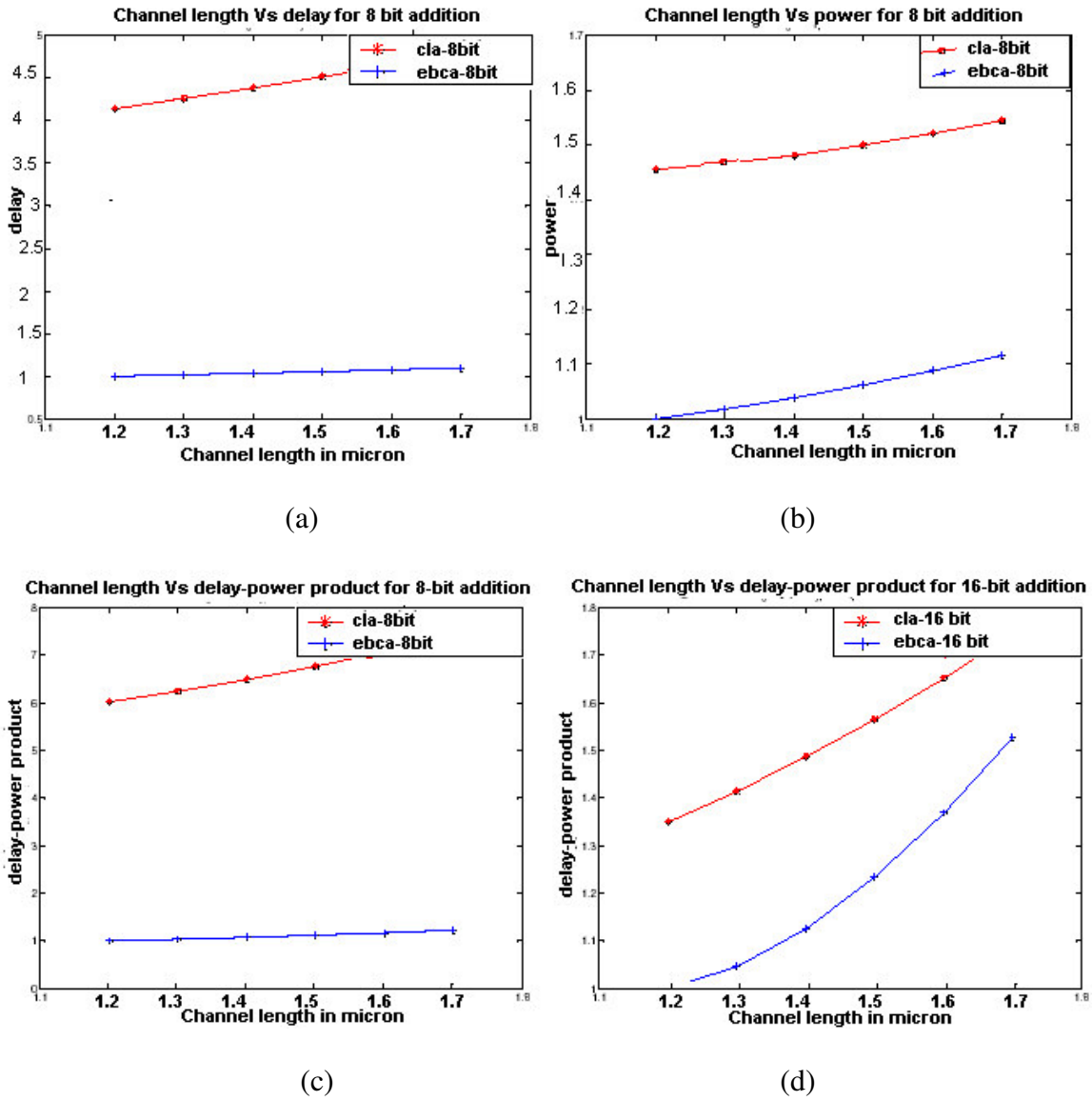



Fig. 4.5 Comparison between CLA and EBC (a) Channel length Vs normalized delay for 8-bit addition (b) Channel length Vs power for 8-bit addition (c) Channel length Vs delay power product for 8-bit addition (d) Channel length Vs delay power product for 16-bit addition.

As seen in chapter 1, two NB numbers can be added to give a RB number using the grouping term in equation 1.1, but the result will be in excess by 1. So after forming the grouping term we have to do RB to NB conversion as well as subtract 1 to get the result in NB form. Now the analysis of the same EBCA in Table 1.4 shows that for ENI = 1, the converter will convert the RB number to NB number as well as subtract one. Such an example

is given in Fig. 4.6. Assume that two NB numbers to be added are A = +2 (0 0 1 0) and B = -7 (1 0 0 1). The addition of these two numbers can be given as A + B +1= (A, $\overline{B}$ ) = - 4. To get the actual result, we have to do RB to NB conversion as well as subtract 1. Now using the EBCA algorithm of Table 1.4 and assuming the first input ENI0 at LSB side as 1, will give the result as –5. So this indicates that the same RB to NB converter can be used as adder by giving ENI0 =1. We also have shown that this RB to NB converter is faster than the CLA for 8-bit and 16-bit operand sizes. So use of this adder circuit will make the final stage addition faster for 8-bit and 16-bit additions.

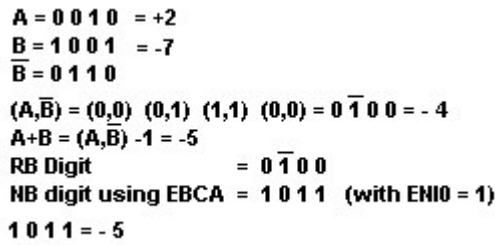



Fig. 4.6 An example of addition of two NB numbers using EBCA

## 4.3 Carry-lookahead EBC

In the previous section we have discussed that the EBC can convert RB to NB (when ENI=0) as well as add two NB numbers (when ENI=1). This is also faster than the CLA for smaller operand sizes. But this circuit will have a longer critical delay in comparison to CLA for larger operand sizes, because of carry propagation. To make the addition and RB to NB conversion faster, carry-lookahead technique along with EBCA is used to have the fastest addition for any operand size.

In a two bit EBC the input carry (ENI) propagates as output carry if the two RB digits are 0. So as the length of EBC adder increases, the worst-case delay path increases. To overcome this, we have used a carry-lookahead technique. In this we have taken a block size

of 8-bits, which can take 8 RB digits as input and gives 8 NB bits as output. A 16-bit carry-lookahead equivalent bit converter (CLEBC) is shown in Fig. 4.7. This consists of two 8-bit EBCs. Each 8-bit EBC generates a group generate G and group propagate P signal. These G and P signals will speed up the ENI generation for each 8-bit EBC.

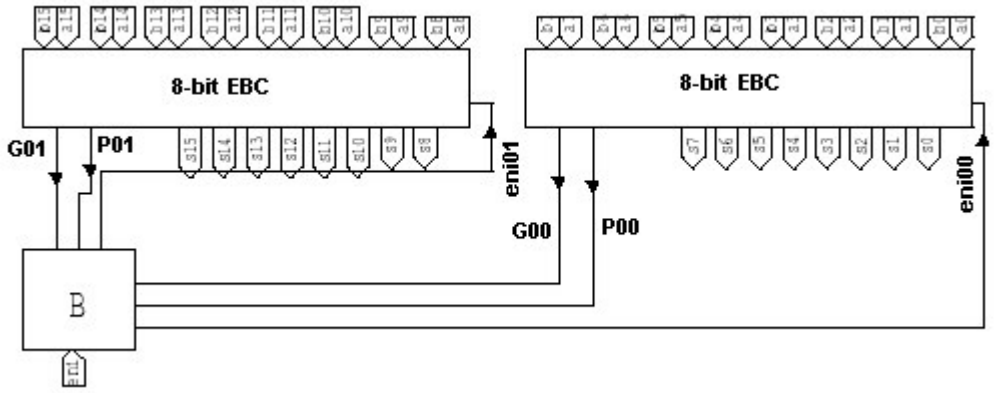



Fig. 4.7 A 16-bit carry-lookahead equivalent bit converter

An 8-bit EBC used in a 16-bit CLEBC is shown in Fig. 4.8. It consists of four 2-bit EBCs. An EBC will have output carry ENO as one if it generates a carry or propagates the input carry ENI. A carry in a 2-bit EBC will be generated if the two input digits are (0, $\bar{1}$ ), ($\bar{1}$ , 0), ($\bar{1}$ , $\bar{1}$ ), ($\bar{1}$ , 1). In EBC1 and EBC2 along with ENO, generate carry signals g1 and g2 are also to be generated. The generate signal is obtained in a similar manner as the ENO discussed in the previous section and is shown in Fig. 4.9(a). The only difference is that the input to the multiplexer for obtaining generate signal is the generate signal of the previous stage. In EBC0, no MUX is needed to get g0 as shown in Fig. 4.9(b). In EBC3 no ENO generation is needed. Each 2-bit EBC gives the propagate signal as one for both RB input digits as 0. So p0, p1, p2, p3 are used to get the final 8-bit EBC propagate signal P. The generate signal of EBC3 is the final generate signal G of 8-bit EBC block.

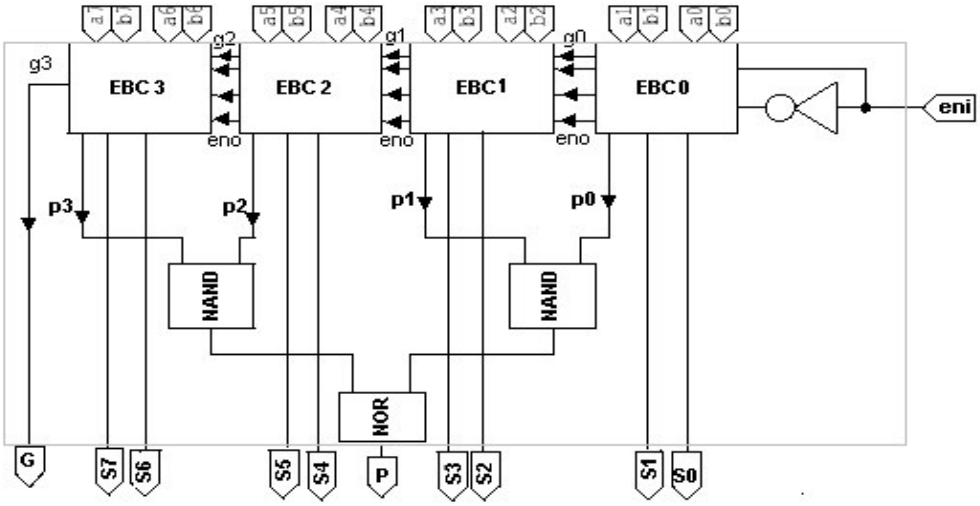



Fig. 4.8 A 8-bit EBC used in 16-bit CLEBC

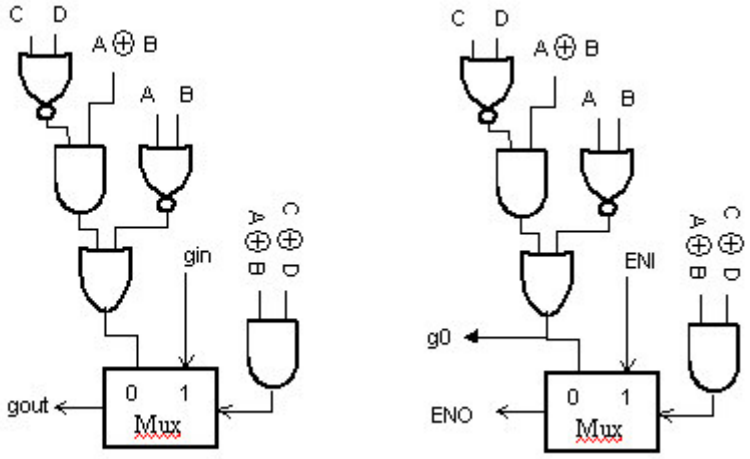



Fig. 4.9 Circuit for obtaining generate signal (a) gout is same as g1 in EBC1, g2 in EBC2 and g3 in EBC3 (b) g0 in EBC0

Based on this methodology CLEBCs are implemented using S-edit for operand sizes of 16, 32 and 64 bits. CLAs are also implemented using the same technology for 16, 32 and 64 bit sizes. The delay results obtained from simulation are normalized with respect to the delay of 16-bit CLEBC, which is assumed as 1 and plotted as shown in Fig. 4.10. This plot of normalized delay versus operand size shows that the delay of addition using CLEBC is

smaller than that of CLA for operand size equal to or greater than 16 bits. So this novel CLEBC is also put to use in the final adder stage for addition and RB to NB conversion.

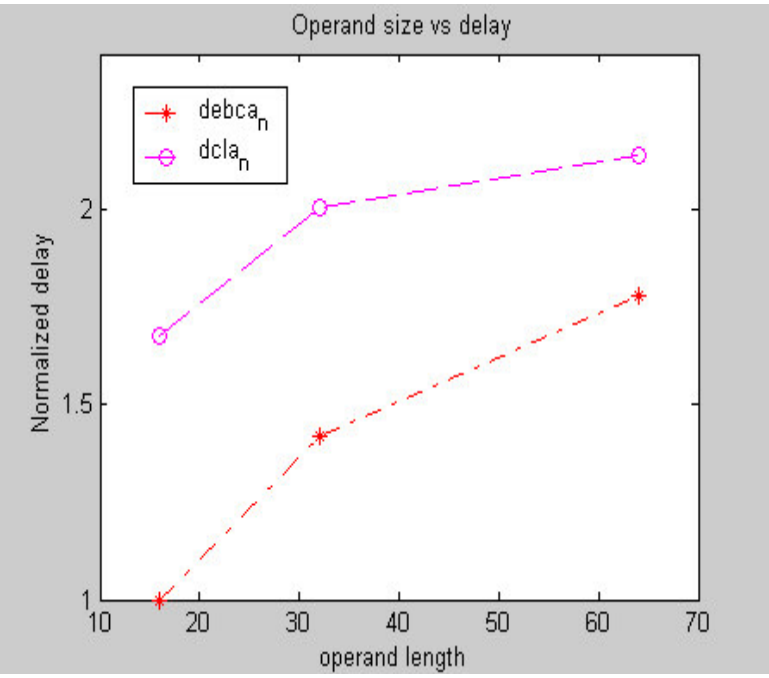



Fig. 4.10 Comparison of delay performance of CLA and CLEBC.

In this EBC a pass transistor based multiplexer as shown in Fig. 4.3(b) is used. But this cell is not available in the Toshiba library. So for finding the delay of this cell, we have implemented a 2-input XOR gate and this multiplexer with 1.2µm technology available to us. Worst-case delays for both the cells are found and the multiplexer delay is normalized with respect to the XOR delay and is found to be 0.16. This delay is used for the critical delay path calculation.

In the 16-bit CLEBC the EBC block size is chosen as 8 as shown in Fig. 4.7. In an 8-bit EBC block, the generate and propagate signals G00 and P00 are obtained from a0, b0, …., a3, b3. These signals are used by the B-cells to generate eni01. Finally s0, s1, …., s15 will be obtained. So the critical delay path for this 16-bit CLEBC is given below:

$$(a0,b0)(a1,b1) \xrightarrow{2.705} G00 \xrightarrow{1.225} eni\,0 \xrightarrow{0.94} s15$$

The total delay of this CLEBC is 4.87.

For a 32-bit CLEBC, one more level of B-cells will be used. This will result in an additional delay of 2.550. In the same way the delay for different sizes of CLEBC is computed which is summarized in Table 4.3. The delay of CLEBC with CLA for different word lengths in terms of $T_{XOR2}$ delay is plotted for comparison in Fig. 4.11. This shows that the CLEBC is much faster than CLA.

Table 4.3 Worst-case delays for different sizes of CLEBC in terms of normalized gate delay

| CLEBC operand size | Worst case delay in terms of $T_{XOR}$ |
|---|---|
| 8-bit | 2.865 |
| 16-bit | 4.87 |
| 32-bit | 7.42 |
| 64-bit | 9.97 |
| 128-bit | 12.52 |

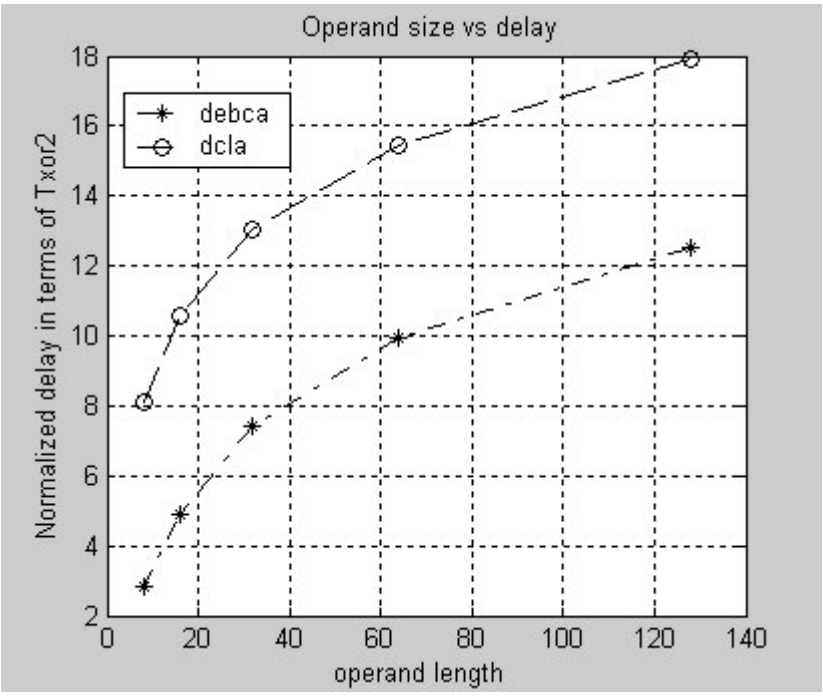



Fig. 4.11 Comparison of delay performance of CLA and CLEBC in terms of $T_{XOR2}$