CHAPTER 3

ANALYSIS OF LOW POWER, AREA EFFICIENT AND HIGH SPEED ADDER TOPOLOGIES

3.1 INTRODUCTION

The design of high-speed and low-power VLSI architectures needs efficient arithmetic processing units, which are optimized for the performance parameters, namely, speed and power consumption. Adders are the key components in general purpose microprocessors and digital signal processors. They also find use in many other functions such as subtraction, multiplication and division. As a result, it is very pertinent that its performance augers well for their speed performance. Furthermore, for the applications such as the RISC processor design, where single cycle execution of instructions is the key measure of performance of the circuits, use of an efficient adder circuit becomes necessary, to realize efficient system performance. Additionally, the area is an essential factor which is to be taken into account in the design of fast adders. Towards this end, high-speed, low power and area efficient addition and multiplication has always been a fundamental requirement of high-performance processors and systems. The major speed limitation of adders arises from the huge carry propagation delay encountered in the conventional adder circuits, such as ripple carry adder and carry save adder.

The following adder topologies are simulated and analysis is made for proposed MAC unit.

- Ripple Carry Adder
- Block Carry Look-Ahead Adder
- Ripple block Carry Look-Ahead Adder
- Carry Increment adder
- Carry Skip Adder with fixed block size
- Carry Skip Adder with variable block size
- Carry Select Adder
- Conditional Sum Adder

The chapter is organized as follows: Section 3.1 contains an introduction of adders. In Section 3.2 the various adders are simulated and outputs are presented. In section 3.3 the analysis is done on different adders and graphs are presented based on area, delay and power dissipation. Finally, the conclusion is given in section 3.4.

3.2 ADDER TOPOLOGIES

3.2.1 Ripple Carry Adder

The Ripple Carry Adder (RCA) is one of the simplest adders to implement. This adder takes in two N-bit inputs (where N is a positive integer) and produces (N+1) output bits (an N-bit sum and a 1-bit carryout). The RCA is built from N full adders cascaded together, with the carryout bit of one FA tied to the $carry_{in}$ bit of the next FA. Figure 3.1 shows the schematic for an N-bit RCA. The input operands are labeled a and b, the carryout of each FA is labeled c_{out} (which is equivalent to the $carry_{in}$ (c_{in}) of the subsequent FA), and the sum bits are labeled sum. Each sum bit requires both input operands and c_{in} before it can be calculated. To estimate the

propagation delay of this adder, look at the worst case delay over every possible combination of inputs. This is also known as the critical path. The most significant sum bit can only be calculated when the carryout of the previous FA is known. In the worst case (when all carryout's are 1), this carry bit needs to ripple across the structure from the least significant position to the most significant position. Figure 3.2 has a darkened line indicating the critical path.

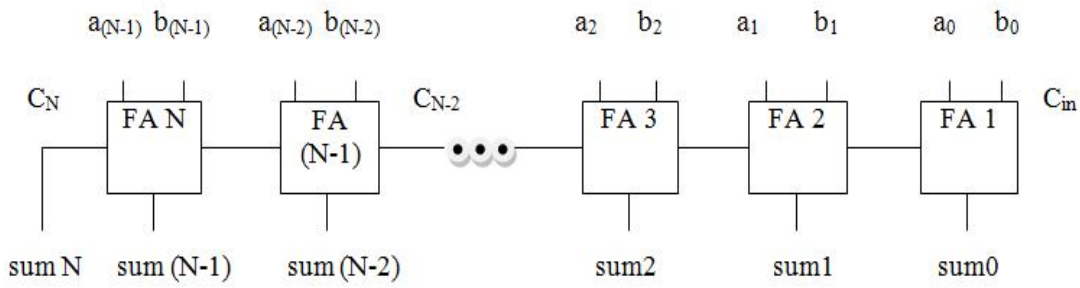

Figure 3.1 Schematic for an N-bit ripple carry adder

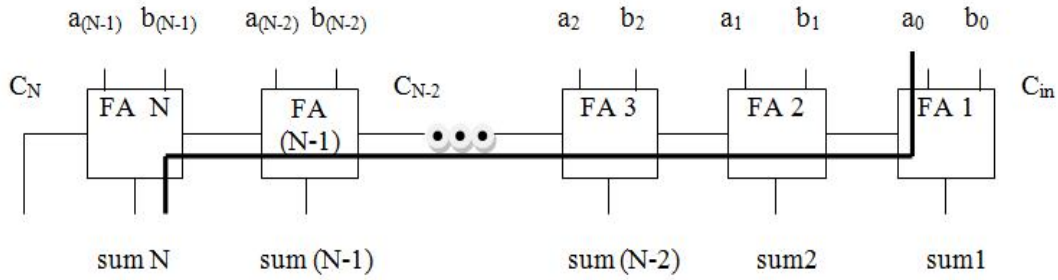

Figure 3.2 Critical path for an N-bit ripple carry adder

Hence the time for this implementation of the adder is expressed in Equation 3.1, where t_{RCA} carry is the delay for the carryout of a FA and t_{RCAsum} is the delay for the sum of a FA.

$$\text{Propagation Delay } (t_{RCAprop}) = (N-1).t_{RCAcarry} + t_{RCAsum} \qquad (3.1)$$

From Equation 3.1, the delay is proportional to the length of the adder. An example of a worst case propagation delay input pattern for an a 4 bit ripple carry adder is where the input operands change from 1111 and 0000 to 1111 and 0001, resulting in a sum changing from 01111 to 10000.

From a VLSI design perspective, this is the easiest adder to implement. One just needs to design and layout one FA cell, and then array N of these cells to create an N-bit RCA. The performance of the one FA cell will largely determine the speed of the whole RCA. From the critical path in equation 3.1, minimizing the carryout delay ($t_{RCAcarry}$) of the FA will minimize $t_{RCAprop}$. There are various implementations of the FA cell to minimize the carry out delay.

3.2.2 Carry Select Adder

Carry Select Adder (CSA) is known to be the fastest adder among the conventional adder structures. It is used in many data processing units for realizing faster arithmetic operations. Adding two numbers by using redundancy can speed addition even further. That is, for any number of sum bits, can perform two additions, one assuming the $carry_{in}$ is 1 and one assuming the $carry_{in}$ is 0, and then choose between the two results once the actual $carry_{in}$ is known. This scheme, proposed by Sklanski (1960), is called conditional-sum addition. An implementation of this scheme is called the Carry Select Adder (CSLA).The CSLA divides the adder into blocks that have the same input operands except for the $carry_{in}$.

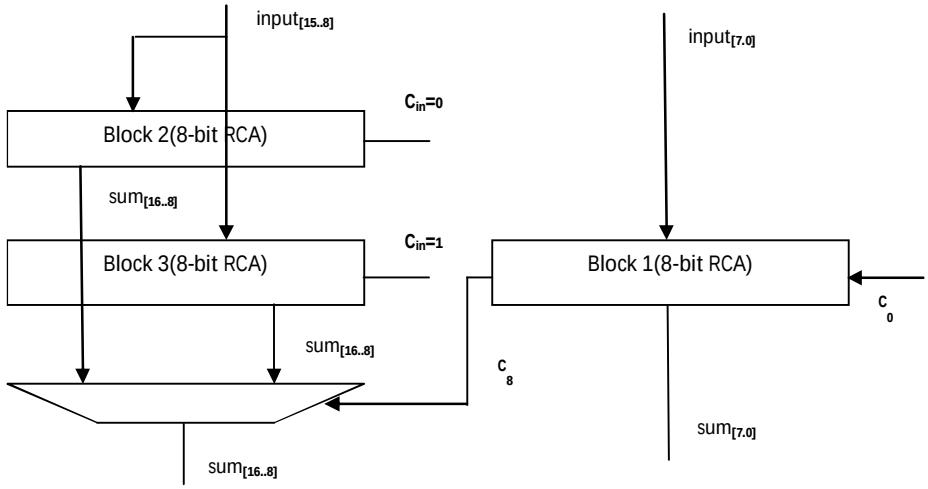

Figure 3.3 16-bit CSLA with 8-bit RCA blocks

Figure 3.3 shows a possible implementation for a 16-bit CSLA using ripple carry adder blocks. The carryout of the first block is used as the select line for the 9-bit 2-to-1 mux. The second and third blocks calculate the signals sum 16 - sum 8 in parallel, with one block having its $carry_{in}$ hardwired to 0 and another hardwired to 1. After one 8-bit ripple adder delay there is only the delay of the mux to choose between the results of block 2 or 3. Equation 3.2 shows the delay for this adder. The 16-bit CSLA can also be built by dividing it into more blocks. Figure 3.4 shows the block diagram for the adder if it were divided into 4-bit RCA blocks. Equation 3.3 expresses the delay for this structure.

$$t_{CSLA16a} = t_{8bitRCA} + t_{(9bitmux)} \tag{3.2}$$

$$t_{CSLA16b} = t_{4bitRCA} + 3.t_{(5bitmux)} \tag{3.3}$$

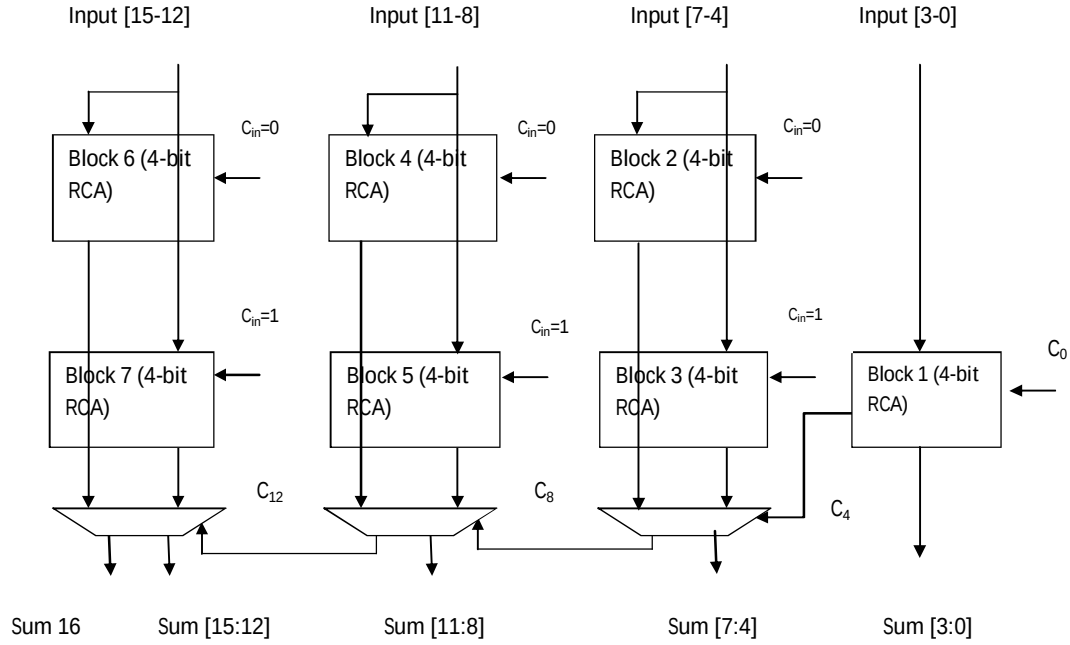

Figure 3.4 Schematic diagram of 16-Bit carry select adder

The CSLA can use any of the adder structures discussed in the previous sections as sub cells. The delay ultimately comes down to the speed of the adder sub cell used and the speed of the muxes used to select the sum bits. A general equation for this adder is expressed in Equation 3.4, where N is the adder size, and k is the group size of each adder sub cell.

$$t_{CSLAN} = t_{k\text{-bitadder}} + \frac{N}{K} t_{((k+1)\text{bitmux})} \quad (3.4)$$

The CSLA described so far is called the Linear Carry Select Adder, because its delay is linearly dependent on the length of the adder. In the worst case, the carry signal must ripple through each mux in the adder. Also, notice that the sub cells are done with their addition at the same time, yet the more significant bits are waiting at the input of the mux to be selected. An optimization to this structure is to vary the length of each of the adder sub cells, observing the fact that the later groups have more time to add because the select signal for their muxes take longer to arrive. The result is a structure called the Square Root Carry Select Adder, and Equation 3.5 expresses the

delay equation as discussed by Chandrakasan et al (2001), where t_{adder} is the delay of the first block which generates the select line for a mux, and $\sqrt{2}N$ is the number of groups the CSLA is divided into, the derivation for this square root CSLA is done by as discussed by Ramkumar & Kittur (2012),

$$t_{sqCSLAN} = t_{adder} + (\sqrt{2N}).t_{mux} \quad (3.5)$$

From a VLSI design perspective, the CSLA uses a large amount of area compared to the other adders. There is hardware in this architecture which computes results that are thrown away on every addition, but the fact that the delay for add can be replaced by the delay of a mux makes this architecture very fast. Also, the Linear CSLA has regularity that makes it easier to layout. The Square Root CSLA, on the other hand, has higher performance but is more time consuming to implement. The varying length of the adders makes sub cell reuse difficult. Rabaey (2003) demonstrates a Square Root CSLA with subsequent adder blocks increasing by one bit. In practice, using a high performance sub cell such as the CLA adder in the Square Root CSLA will result in subsequent blocks which differ by more than one bit. For example, in a 12-bit Square Root CSLA, the first block will consist of a 4-bit CLA adder, and the second and third blocks will consist of two 8-bit CLA adders, followed by a mux. This may not provide as much speed up as an optimized Square Root CSLA, but it requires less time to implement.

3.2.3 Carry Look ahead Adder

From the critical path equations in Section 3.2.1, the delay is linearly dependent on N, the length of the adder. It is also shown in equations 3.1 and 3.2 that the $t_{carryout}$ signal contributes largely to the delay. An algorithm that reduces the time to calculate $t_{carryout}$ and the linear dependency on N can greatly speed up the addition operation. Equation carryout = $g_i + p_i * carry_{in}$ shows that the carryout can be calculated with g, p, and $carry_{in}$. The

signals g and p are not dependent on $carry_{in}$, and can be calculated as soon as the two input operands arrive. Weinberger and Smith invented the Carry Look Ahead (CLA) Adder and also discussed by Wallace (1964). Using Equation carryout = $g_i + p_i * carry_{in}$, and write the carryout equations for a 4 -bit adder. The above equations are shown in Equations 3.6-3.9, where c_i represents the carryout of the i^{th} position ($0 \leq i \leq (N-1)$), and gi, pi represent generate and propagate signal from each PFA. The equations for c2, c3 and c4 are obtained by substitution of c1, c2 and c3, respectively. These equations show that every carryout in the adder can be determined with just the input operands and initial $carry_{in}$ (c3). This process of calculating c_i by using only the pi, gi and c0 signals can be done indefinitely, however, each subsequent carryout generated in this manner becomes increasingly difficult because of the large number of high fan-in gates as discussed by Chen (2003).

$$c_1 = g_0 + p_0.c_0 \tag{3.6}$$

$$c_2 = g_1 + p_1.c_1 = g_1 + p_1.\, g_0 + p_1.\, p_0.c_0 \tag{3.7}$$

$$c_3 = g_2 + p_2.c_2 = p_2.\, g_1 + p_2.\, p_1.\, g_0 + p_2 p_1.\, p_0.c_0 \tag{3.8}$$

$$c_4 = g_3 + p_3.c_3 = g_3 + p_3.\, g_2 + p_3.\, p_2.\, g_1 + p_3.\, p_2.\, p_1.\, g_0 + p_3.\, p_2.\, p_1.\, p_0.c_0 \tag{3.9}$$

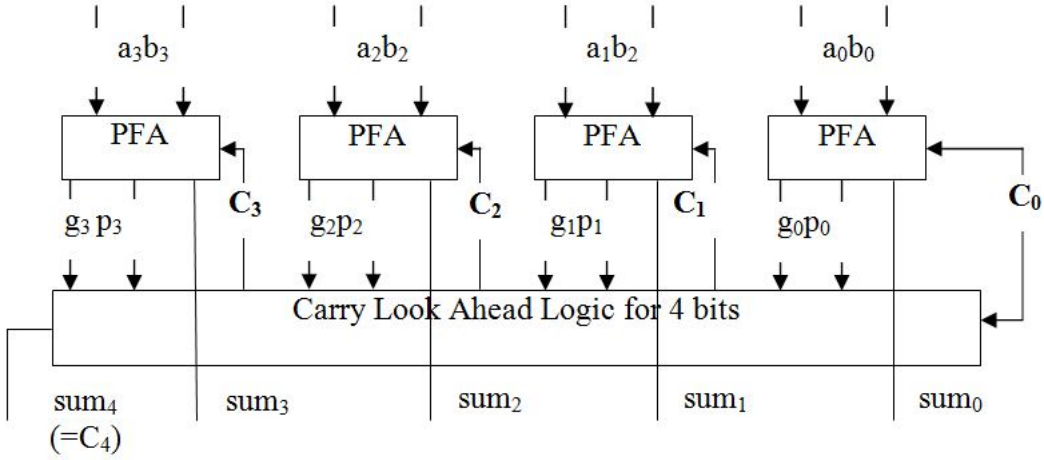

Figure 3.5 Schematic of 4-bit carry look ahead adder

The CLA adder uses partial full adders to calculate, generate and propagate signals needed for the carryout equations. Figure 3.5 shows the schematic for a 4-bit CLA adder. The CLA logic block implements the logic in Equations 3.6-3.9. For a 4-bit CLA adder the 4^{th} carryout signal can also be considered as the 5^{th} sum bit. Although it is impractical to have a single level of carry look ahead logic for long adders, this can be solved by adding another level of carry look ahead logic. To achieve this, each adder block requires two additional signals: groups generate and groups propagate. The equations for these two signals, assuming adder block sizes of 4 bits, are shown in Equations 3.10 and 3.11. A group generate occurs if a carry is generated in one of adder blocks, and a group propagate occurs if the carry in to the adder block will be propagated to the carryout.

$$\text{Group generate} = g_3 + p_3.\, g_2 + p_3.\, p_2.\, g_1 + p_3.\, p_2.\, p_1.c_3 \qquad (3.10)$$

$$\text{Group propagate} = p_0\, .p_1\, .p_2\, .\, p_3 \qquad (3.11)$$

With multiple levels of CLA logic, carry look ahead adders of any length can be built. The size of an adder block in a CLA adder is usually 4 bits because it is a common factor of most word sizes and there is a practical limit on the gate size that can be implemented. To illustrate the use of another level of CLA logic, Figure 3.6 shows the schematic for a 16-bit CLA adder. There is a second level of CLA logic which takes the group generate and group propagate signals from each 4-bit adder sub cell and calculates the carryout signals for each adder block. If an adder has multiple levels of CLA logic, only the final level needs to generate the c4 signal. All other levels replace this c4 signal with the group generate and group propagate. The CLA logic for this 16-bit adder is identical to the CLA logic for the 4-bit adder. Therefore the equations for the carryout signals are in Equations 3.6-3.9.

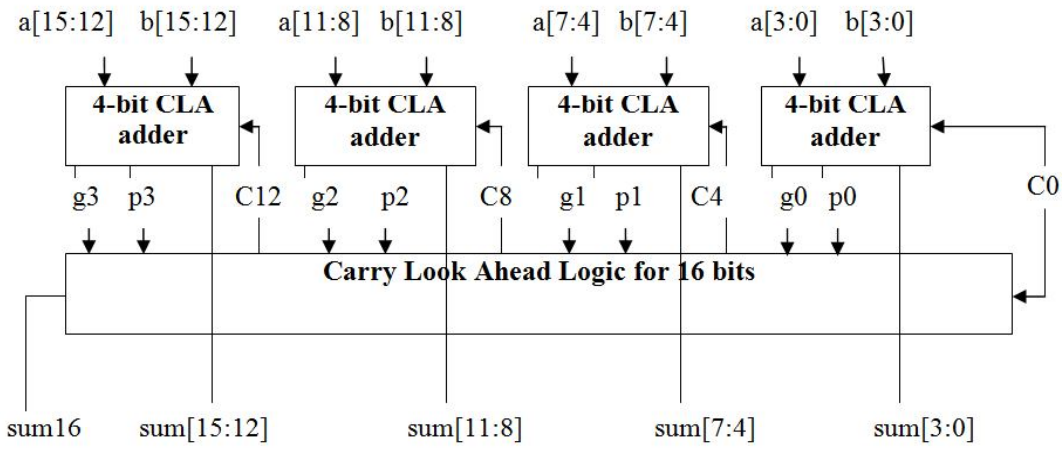

Figure 3.6 Schematic of 16-bit CLA adder

A third level of CLA logic and four 16-bit adder blocks can be used to build a 64-bit adder. The CLA logic would create the c16, c32, and c48 signals to be used as carry in to the 16-bit adder blocks and the c64 as the sum 64 signal. If a design calls for an adder of length 32, a designer can simply use two 16-bit adder blocks and the first two carryout signals (c16, c32) from the third level of CLA logic. The identical hardware in the CLA logic, coupled with the fact that the adder blocks can be instantiated as sub cells, makes building long adders with this architecture makes simple. Determining the critical path for a CLA adder is difficult because the gates in the carry path have different fan-in. To get a general idea, first assume that all gate delays are the same. The delay for a 4-bit CLA adder then requires one gate delay to calculate the propagate and generate signals, two gate delays to calculate carry signals, and one gate delay to calculate the sum signals; this equates to four gate delays. For a 16-bit CLA adder there is one gate delay to calculate the propagate and generate signal, two gate delays to calculate the group propagate and generate in the first level of carry logic, two gate delays for the carryout signals in the second level of carry logic, and one gate delay for the sum signals. The second level of carry logic for the 16-bit CLA adder contributes an additional two gate delays over the 4-bit CLA adder, thus

increasing the total to six gate delays. Continuing in this manner (a 64-bit add takes eight gate delays, a 256-bit add takes ten gate delays), the delay for a CLA adder is dependent on the number of levels of carry logic, and not on the length of the adder. If a group size of four is chosen, then the number of levels in an N -bit CLA is expressed in Equation 3.12 and in general the number of levels in a CLA for a group size of k is expressed in Equation 3.13. For a N-bit CLA adder, each level of carry logic introduces two gate delays in addition to a gate delay for generate and propagate signals and a gate delay for the sum. The total gate delay is expressed in Equation 3.14, which shows that the delay of a CLA adder is logarithmically dependent on the size of the adder. This theoretically results in one of the fastest adder architectures.

$$\text{CLA levels (with group size of 4)} = [\log_4 N] \tag{3.12}$$

$$\text{CLA levels (with group size of K)} = [\log_k N] \tag{3.13}$$

$$\text{CLA gate delay} = 2+2.[\log_k N] \tag{3.14}$$

Equation 3.14, however, lacks the detail necessary to make a good delay estimate. Each gate in the adder varies in both the number of inputs it has and the function it implements. These 5-input NAND gates are in the logic for c4, which is the most significant bit for the result of any add. If there are multiple levels of carry logic, the c4 logic is replaced with the group propagate and generate signals, and is used only in the final level of carry logic. Also, this signal is not in the critical path because once it is calculated, its result can be immediately used, as opposed to the other carryout signals. Signals c1, c2, and c3 feed into the PFAs, where the sum signal still needs to be calculated (another XOR gate delay). The second largest gates are the 4-input NAND gates, with four NMOS transistors in series. These gates are contained in the group generate and c3 logic. The critical path therefore goes

through the group generate signal (in the first and intermediate levels of carry logic), and the c3 signal in the last level of carry logic.

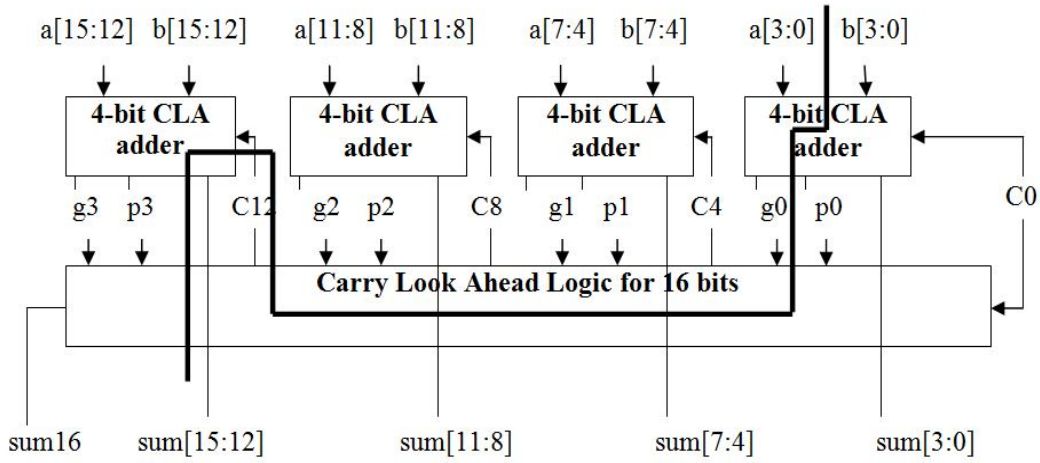

Figure 3.7 Critical path for 16-bit CLA adder

Figure 3.7 shows a darkened line indicating the critical path of the signals in the 16-bit CLA adder, and Equation 3.15 expresses the critical delay of a 16-bit CLA adder. In this equation, t_{prop} is the propagate delay for a PF A, $t_{GroupGen}$ is the delay for the group generate signal in the first level of carry logic, tc3 is the delay for c3 in the second level of carry logic, and t_{xor} is the second XOR delay of the PF A to calculate the sum. For a N -bit CLA adder with 4-bit groups, the delay is expressed in Equation 3.16. The second term in this equation is the number of carry levels (minus 1) multiplied by the delay of the group generates signal, and shows that the delay is logarithmically dependent on the length of the adder.

$$t_{CLA16} = t_{prop} + t_{Groupgen} + t_{couts} + t_{xor} \quad (3.14)$$

$$t_{CLAN} = t_{prop} + ([\log_k N]-1).\ t_{Groupgen} + t_{couts} + t_{xor} \quad (3.15)$$

From a VLSI design perspective, this adder may take more time to implement, but there still exists regularity with the architecture that allows

building long adders easily. The reuse of the CLA logic definitely contributes to the feasibility of building a long adder without additional design time. Also, after an adder is built, it can be used as a sub cell, as is done with the 4-bit adders as blocks in the 16-bit CLA adder. A drawback to CLA adders are their larger areas. There is a large amount of hardware dedicated to calculate the carry bits from cell to cell. However, if the application calls for high performance, then the benefits of decreased delay can outweigh the larger area.

3.2.4 Carry Increment Adder

A 16-bit increment adder includes four RCA (Ripple carry adder) of four bit each. The first ripple carry adder adds a desired number of first 4-bit inputs generating a plurality of partitioned sum and partitioned carry. Now the carry out of the first block RCA is given to C_{IN} of the conditional increment block. Thus the first four bit sum is directly taken from the ripple carry output. The second RCA block regardless of the first RCA output will carry out the addition operation and will give out results which are fed to the conditional increment block. The input C_{IN} to the first RCA block is given always low value. The conditional increment block consists of half adders. Based on the value of c_{out} of the 1st RCA block, the increment operation will take place. Here the half adder in carry increment block performs the increment operation. Hence the output sum of the second RCA is taken through the carry increment block. The design schematic of Carry Increment is shown in Figure 3.8.

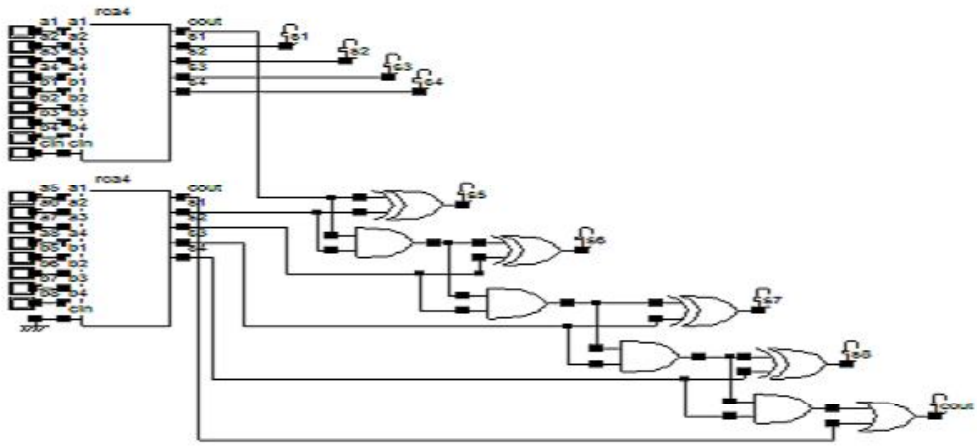

Figure 3.8 Carry increment adder

3.2.5 Carry Skip Adder

From examination of the RCA, the limiting factor for speed in that adder is the propagation of the c_{out} bit. The Carry Skip Adder (CSKA, also known as the Carry Bypass Adder) addresses this issue by looking at groups of bits and determines whether this group has a carryout or not. This is accomplished by creating a group propagate signal (p_{CSKA}group) to determine whether the group $carry_{in}$ ($carry_{inCSKAgroup}$) will propagate across the group to the carryout ($carryout_{CSKAgroup}$). To explore the operation of the whole CSKA, take an N-bit adder and divide it into N/M groups, where M is the number of bits per group. Each group contains a 2-to-1 multiplexer, logic to calculate M sum bits, and logic to calculate $p_{CSKAgroup}$. The select line for the mux is simply the $p_{CSKAgroup}$ signal, and it chooses between $carry_{inCSKAgroup}$ or $cout_4$. To aid the explanation, referring figure 3.9, which shows the hardware for a group of 4 bits (M=4) in the CSKA. There are four full adders cascaded together and each FA creates a carryout (c_{out}), propagate (p) signal, and a sum (sum not shown). The propagate signal from each FA comes at no extra hardware cost since it is calculated in the sum logic (the hardware is identical to the sum hardware for the PFA shown in Figure 2.1). For the

$carryout_{CSKAgroup}$ to equal $carry_{inCSKAgroup}$, all of the individual propagates must be asserted (Equations 3.17 and 3.18). If this is true then $carryin_{CSKAgroup}$ skips the group of full adders and equals the $carryout_{CSKAgroup}$. For the case where $p_{CSKAgroup}$is 0, at least one of the propagate signals is 0. This implies that either a delete and/or generate occurred in the group. A delete signal simply means that the carryout for the group is 0 regardless of the $carry_{in}$, and a generate signal means that the carryout is 1 regardless of the $carry_{in}$. This is advantageous because it implies that the carryout for the group is not dependent on the $carry_{in}$. No hardware is needed to implement these two signals because the group carryout signal will reflect one of the three cases (a d, g or group p occurred). The additional hardware is to realize the group carryout in Figure 3.9 and is accomplished with a 4-input AND gate and a 2-to-1 multiplexer (mux). In general, an M-input AND gate and a 2-to-1 mux are required for a group of bits, including the logic to calculate the sum bits.

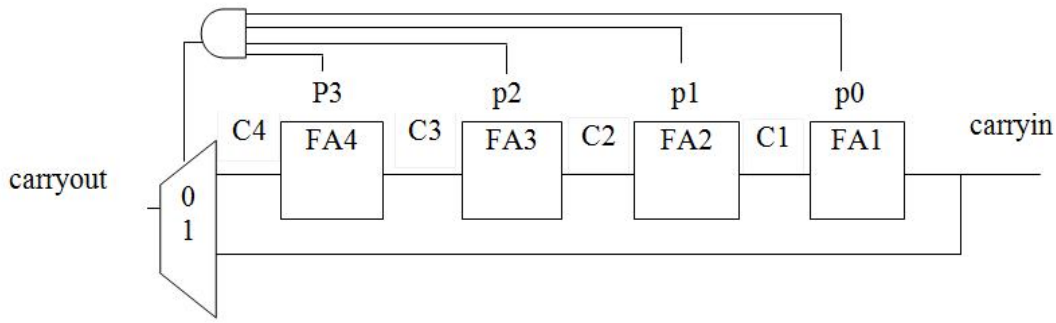

Figure 3.9 Carry skip adder

$$p_{CSKAgroup} = p_0 \cdot p_1 \cdot p_2 \cdot p_3 \quad (3.17)$$

$$carryout_{CSKAgroup} = carryin_{CSKAgroup} \cdot p_{CSKAgroup} \quad (3.18)$$

By examining the critical path for the CSKA, it is important to concern whether the $carry_{in}$ can be propagated (“skipped") across a group or not. Assuming all input bits come into the adder at the same time, each group

can calculate the group propagate signal (mux select line) simultaneously. Every mux knows signal to pass as the carryout of the group. There are two cases to consider after the mux select line has been determined. In the first case, $carryin_{CSKAgroup}$ will propagate to the carryout. This means $p_{CSKAgroup}=1$ and the carryout is dependent on the $carry_{in}$. In the second case, the carryout signal of the most significant adder will become the group carryout. This means $p_{CSKAgroup}=0$ and the carryout is independent of the $carry_{in}$. By isolating a particular group (as in Figure 3.9), the second case (signal cout4) always takes longer because the carryout signal must be calculated through logic, whereas the first case requires only a wire to propagate the signal. Looking at the whole architecture, however, this second case is part of the critical path for only the first CSKA group. Since the second case is not dependent on the group $carry_{in}$, all the groups in the CSKA can compute the carryout in parallel. If a group needs its $carry_{in}$ ($p_{CSKAgroup}=1$), then it must wait until it arrives after being calculated from a previous group. In the worst case, a carryout must be calculated in the first group, and every group afterwards needs to propagate this carryout. When the final group receives this propagated signal, then it can calculate its sum bits. Figure 3.10 shows a 16-bit CSKA with 4-bit groups and Figure 3.11 shows a darkened line indicating the critical path of the signals in the 16-bit CSKA.

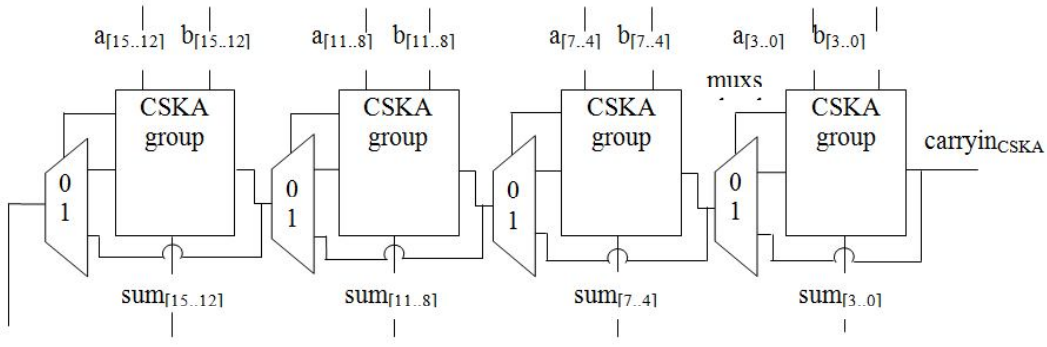

Figure 3.10 16-bit carry skip adder

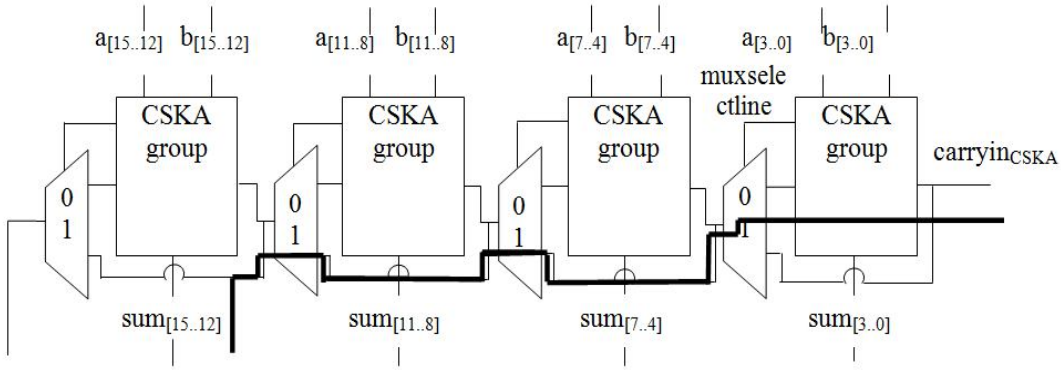

Figure 3.11 Critical Path through 16-bit carry skip adder

A 16-bit CSKA with 4-bit groups, with each group containing a 4-bit RCA for the sum logic, and then the worst case propagation delay through this adder is expressed in equation 3.19. In this equation, $t_{RCAcarry}$ and t_{RCAsum} are the delays to calculate the carryout and sum signals of an RCA, respectively. Each group has 4 bits, so the delay through the first group has 4 RCA carryout delays. This carryout of the first group potentially propagates through 3 muxes, where one mux delay is expressed as $t_{muxdelay}$. Finally, when the carryout signal reaches the final group, the sum for this group can be calculated.

$$t_{CSKA16} = 4*t_{RCAcarry} + 3*t_{muxdelay} + 3* t_{RCAcarry} + t_{RCAsum} \quad (3.19)$$

For Equation 3.19, there are some assumptions about the delay through the circuit. Consider the first CSKA group that the group propagates signal is calculated before the carryout of the most significant adder. Thus, the mux for this first group is waiting for the carryout. For the final CSKA group, assume that it takes longer for sum15 to be calculated than for sum 16 to be calculated. Once the $carry_{in}$ for this last group is known, the delay for sum 16 is the delay of the mux; for sum 15 it is a delay of $3*t_{RCAcarry} + t_{RCAsum}$ (3 ripples through the adder before the last sum bit can be calculated). For an N-bit CSKA, the critical path equation is expressed in Equation 3.20. M

represents the number of bits in each group. There are N/M groups in the adder, and every mux in this group except for the last one is in the critical path. As in Equation 3.19, Equation 3.20 assumes that each group contains a ripple carry adder.

$$t_{CSKAN} = M * t_{RCAcarry} + \left(\frac{N}{M} - 1\right) t_{muxdelay} + (M-1) * t_{RCAcarry} + t_{RCAsum} \quad (3.20)$$

From a VLSI design perspective, this adder shows improved speedup over a RCA without much area increase. The additional hardware comes from the 2-to-1 mux and group propagates logic in each group, which is about 15% more area. One drawback to this structure is that its delay is still linearly dependent on the width of the adder, therefore for large adders where speed is important, the delay may be unacceptable. Also, there is a long wire in between the groups that carryout CSKA group needs to travel on. This path begins at the carryout of the first CSKA group and ends at the $carry_{in}$ to the final CSKA group. This signal also needs to travel through ($\frac{N}{m}$ -1) muxes, and these will introduce long delays and signal degradation if pass gate muxes are used. If buffers are required in-between these groups to reproduce the signal, then the critical path is lengthened. An example of a worst case delay input pattern for a 16-bit CSKA with 4-bit groups is where the input operands are 1111111111111000 and 0000000000001000. This forces a carryout in the first group that skips through the middle two groups and enters the final group. This $carry_{in}$ to the final group ripples through to the final sum bit (sum15). To determine the optimal speed for this adder, one needs to find the delay through a mux and the carryout delay of a FA. It is one of these two delays that will dominate the delay of the whole CSKA. For short adders ($\leq$ 16 bits), the $t_{carryout}$ of a FA will probably dominate delay, and for long adders the long wire that skips through stages and muxes will probably dominate the delay.

3.2.5.1 Fixed block size carry skip adder

Figure 3.12 shows 16-bit carry-skip adder consisting of four fixed-size blocks, each of size 2. The fixed block size should be selected so that the time for the longest carry-propagation chain can be minimized. The optimal block size k opt follows: $K_{opt} = \sqrt{\frac{n}{2}}$

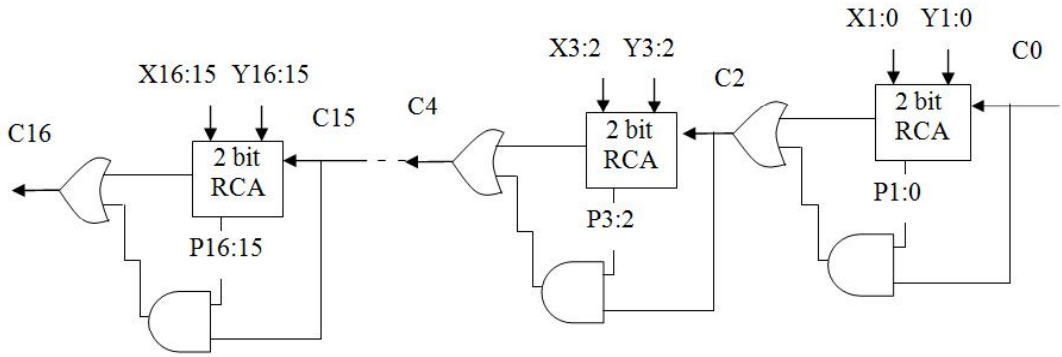

Figure 3.12 16-bit fixed-block-size carry-skip adder

3.2.5.2 Variable block size carry skip adder

Figure 3.13 shows a 16-bit carry-skip adder consisting of seven variable-size blocks. This optimal organization of block size includes L blocks with sizes k1, k2..., kL = 1, 2, 3..., 3, 2, 1. This reduces the ripple-carry delay through these blocks.

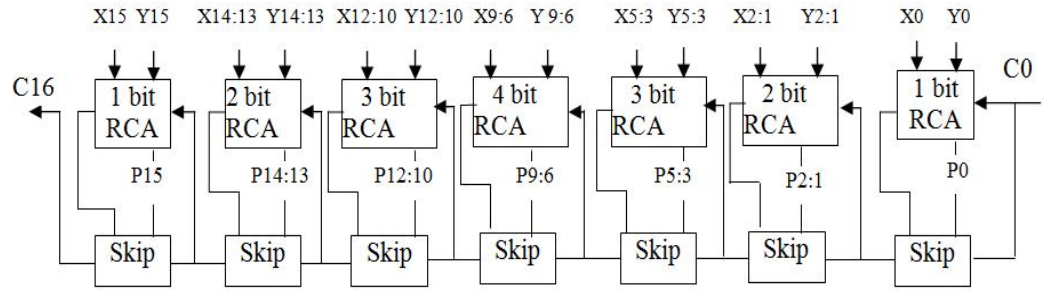

Figure 3.13 16 bit variable block size carry skip adder

3.2.6 Conditional Sum Adder

The basic idea in the conditional sum adder is to generate two sets of outputs for a given group of operand bits and it can be taken as k bits. Each set includes k sum bits and an outgoing carry. One set assumes that the eventual incoming carry will be zero, while the other assumes that it will be one. Once the incoming carry is identified, it should only to select the correct set of outputs (out of the two sets) without waiting for the carry to further propagate through the k positions.

In this generator,the given n-bit operands is divided into two groups of size n/2 bits each. Each of these can be further divided into two groups of n/4 bits each. In principle, this process can be continued until a group of size 1 is reached. The above idea is applied to each of groups separately. The Figure 3.14 shows the conditional sum adder for 4-bits which can be extended upto 16-bits.

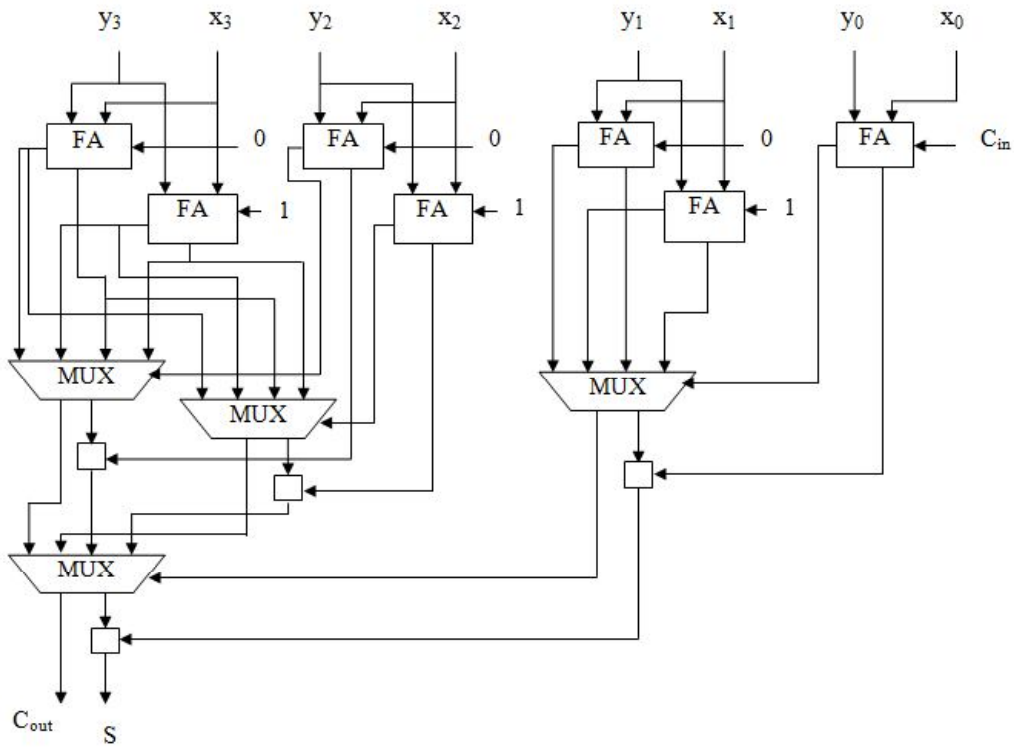

Figure 3.14 Conditional sum adder

3.3 SIMULATION RESULTS

The choices of adders mentioned above are synthesized and simulated using FPGA board. The FPGA board used here is Spartan- 3E XC3S500E, PQ208 configuration. Table 3.1 shows the simulated output of the adders such as Area, Delay and Power dissipation.

Table 3.1 Area, delay and power comparison of adder topologies

Adders	Area	Delay	Power Dissipation
Block Carry Look ahead adder (BCLA)	0.33%	20.20 ns	85.83mW
Carry Select Adder (CSA)	0.39%	17.69 ns	83.88mW
Carry Skip adder with fixed block size (CSFBA)	0.33%	20.083ns	85.83mW
Carry Skip adder with Variable block size (CSVBA)	0.36%	20.34 ns	83.85mW
Conditional Sum Adder (CoSA)	0.55%	13.67 ns	88.72mW
Ripple block carry look ahead adder (RBCLA)	0.33%	20.20 ns	85.83mW
Ripple Carry Adder (RCA)	0.33%	20.20 ns	84.20mW
Carry Increment Adder (CIA)	0.42%	27.89 ns	92.35mW

In the area assessment, it is observed that the maximum area is required for Conditional sum adder and next comes Carry increment adder. The slightest area required for Block Carry look ahead adder, Carry skip adder with fixed block size, Ripple block carry look ahead adder and Ripple carry adder as shown in figure 3.15.

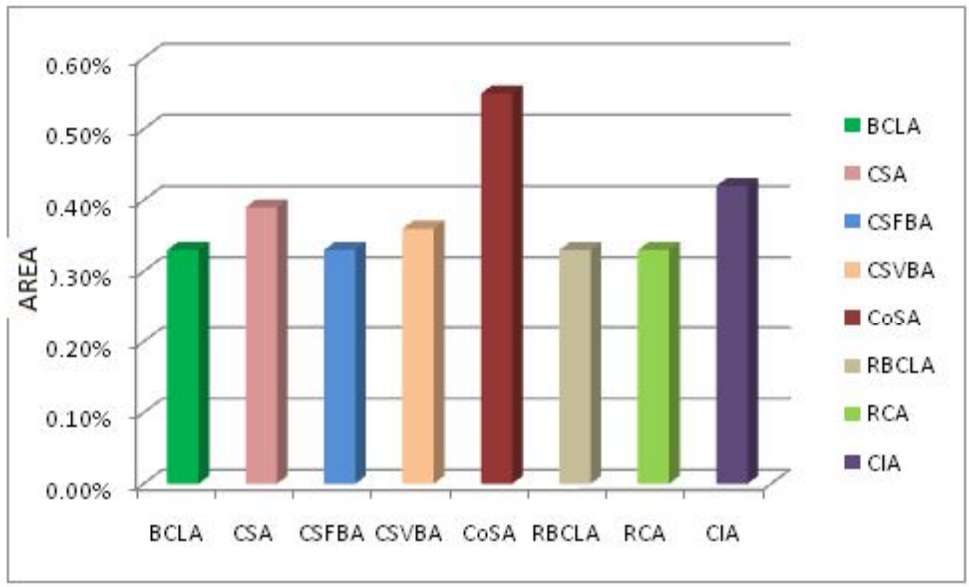

Figure 3.15 Area comparisons of adder topologies

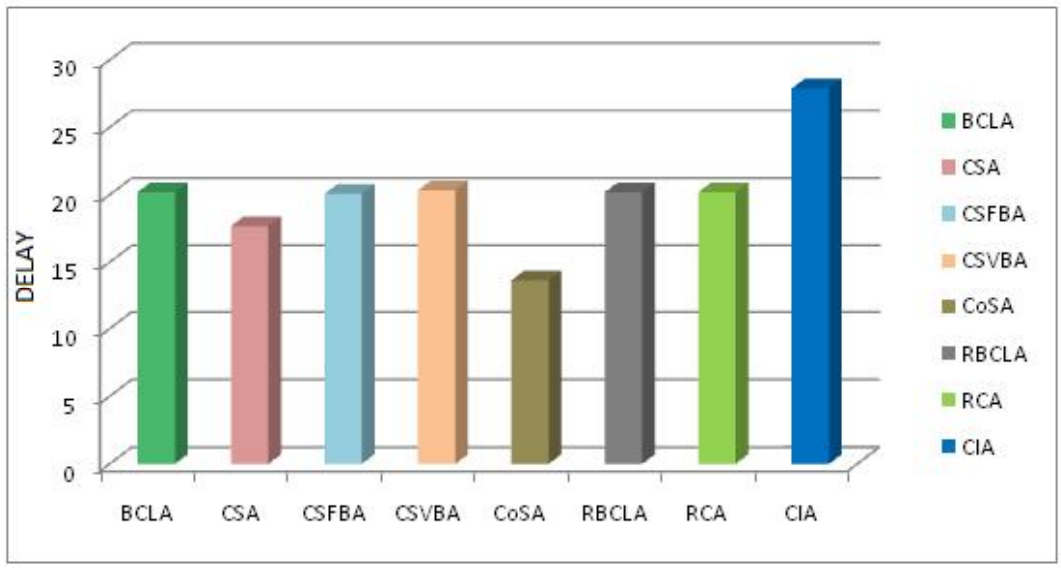

Figure 3.16 Delay comparison of adder topologies

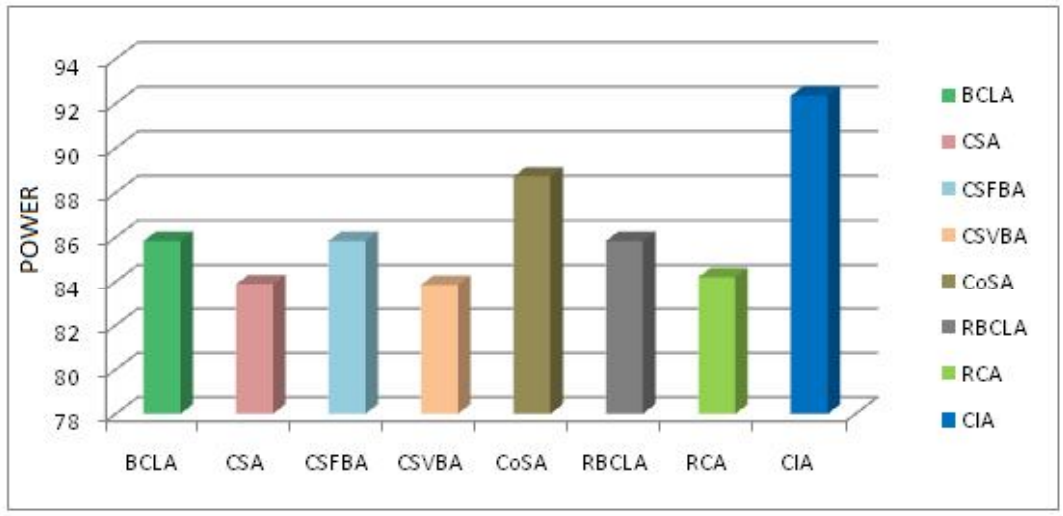

Figure 3.17 Power dissipation comparison of adder topologies

From the delay comparison shown in Table 3.1 and Figure 3.16 it is observed that the maximum delay take place for Carry increment adder. The least delay occurs for Conditional sum adder and Carry select adder and it is somewhat varied to Conditional sum adder.

Together with the adders analyzed, the Carry increment adder is having larger power dissipation as show in Figure 3.17. Among these adders, power dissipation is almost same and compared to these entire adders carry select adder is having reasonable power dissipation and area with high speed.

The overall contrast presents the tradeoff between area, delay and power consumption. By analyzing various adder topologies, Conditional sum adder has nominal delay but greatest area consumption. Correspondingly, the adders such as Block carry look ahead adder, Carry skip adder with fixed block size, Ripple block carry look ahead adder and Ripple carry adder are having minimal area requirement but having maximum delay, when related to Conditional sum adder. When match up to all the above four adders, Carry select adder is having a maximum area consumption but having minimum delay.

Next by analyzing the Carry select adder with Conditional sum adder, the delay of Conditional sum adder is lower than carry select adder. In the same way, the area consumption of Carry select adder is lower than Conditional sum adder. According to the available outcome, the adder topology has the best compromise between area, delay and power dissipation is Carry select adder which is appropriate for high performance and low-power circuits.

From Table 3.1, the power dissipation is analyzed by taking switching power (dynamic power) in account which mainly depends on the input test vectors that can be applied through the test bench. Static power is not considered because lack of ASIC tools available.

Project File:	Adders_complete.xise	**Parser Errors:**	No Errors
Module Name:	BCLA	**Implementation State:**	Placed and Routed
Target Device:	xc3s500e-4pq208	**•Errors:**	No Errors
Product Version:	ISE 13.2	**•Warnings:**	2 Warnings (0 new)
Design Goal:	Balanced	**•Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**•Timing Constraints:**	
Environment:	System Settings	**•Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	31	9,312	1%	
Number of occupied Slices	22	4,656	1%	
Number of Slices containing only related logic	22	22	100%	
Number of Slices containing unrelated logic	0	22	0%	
Total Number of 4 input LUTs	31	9,312	1%	
Number of bonded IOBs	49	158	31%	
Average Fanout of Non-Clock Nets	1.76			

Figure 3.18 Device Utilization Summaries for BCLA Adder

Figure 3.19 Simulated output waveform for BCLA Adder

The number of LUT's and Flip Flops used for Block Carry Look Ahead Adder is shown in figure 3.18 and output waveform is shown in figure 3.19.

3.4 CONCLUSION

In this chapter, an exhaustive analysis of adder topologies has been passed out. The comparison has been analyzed with area, delay and power dissipation. The result which has been presented for the adder topology which has the best compromise linking area, delay and power dissipation is Carry select adder which is suitable for high performance and low-power MAC unit.