CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, carry save adder materials has been studied. The understanding of this design has been explained in details. The characteristics of achieving high-speed design circuit being explained.

2.2 Multioperand Carry-Save Adder (CSA) [1] [5]

Carry save adder (CSA) is the design of a high-speed multioperand adder. A carry save adder consists of a ladder of stand-alone full adders as shown in the figure 2.1. The n-bit CSA consists of *n* disjoint full adders (FAs) where each of which computes a single sum and carry bit based on the corresponding bits of the three input numbers. It consumes three n-bit input integers to be added and produces two outputs, n-bit partial sum and n-bit carry. Unlike the normal adders such as ripple carry adder, a CSA consists of multiple one-bit full adders without any carry chaining.

Carry save adder also known as (3, 2) counter where the addends are three. The carry save adder block diagram can be seen in the figure 2.2. It sums three 4-bits inputs, and returns the result as two 4-bits output. 3:2 counter can be used to speed up the summation of three or more addends. 3:2 counter can be used to speed up the summation of three or more addends. If the addends are four or more, more than one

layer of counter is necessary and there are various possible design for the circuit which the most common are Dadda and Wallace trees [1] [2].

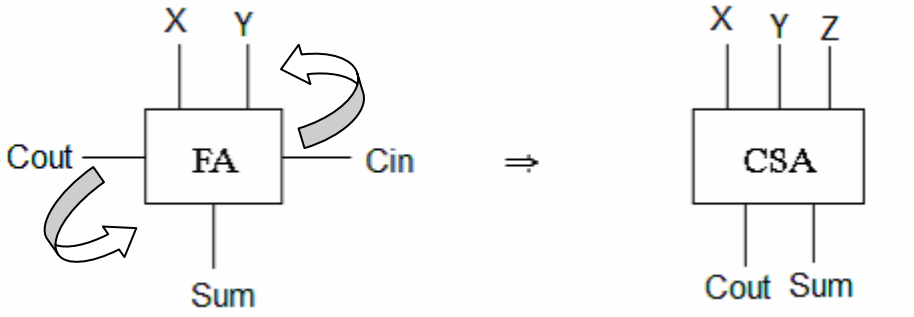


Figure 2.1: The 1-bit carry save adder block is the same circuit as a full adder.

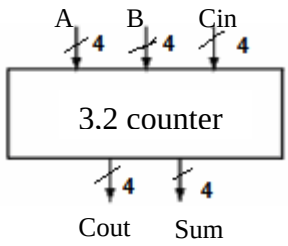


Figure 2.2: The 4-bits carry save adder block diagram

A carry save adder is a different thing all together. A CSA instead of trying to solve the addition problem, it solves a different problem. All a CSA does is converts the problem of adding three numbers together into a problem of adding two numbers together.

A carry-save adder may be implemented in several different ways. In the simplest implementation, the basic element of the carry-save adder is a full adder with three A, B and C inputs can describe Equation 1.0 whose arithmetic output operation. The truth table of CSA can be seen in Table 2.1. Mod displays or remainder, of x/y which using for binary operator. For example, to find 5 divided to 3, click 5 mod 3= which equals 2. The equation 2.0 used if the calculation is in hexadecimal operator.

$$Sum = (A + B + C) \bmod 2 \qquad and \qquad Cout = \frac{(A + B + C) - Sum}{2} \qquad (1.0)$$

$$A + B + C = 2Cout + Sum \quad (2.0)$$

Table 2.1: Truth table of 1-bit CSA

INPUT			OUTPUT		
A	**B**	**C**	**Cout**	**Sum**	**Comments**
0	0	0	0	0	0+0+0=00
0	0	1	0	1	0+0+1=01
0	1	0	0	1	0+1+0=01
0	1	1	1	0	0+1+1=10
1	0	0	0	1	1+0+0=01
1	0	1	1	0	1+0+1=10
1	1	0	1	0	1+1+0=10
1	1	1	1	1	1+1+1=11

2.3 Number Systems

A numeral is a single symbol that represents a quantity or number. A number system is a way of assigning combinations of numerals to different quantities. The act of assigning combinations of numerals to different through the use of a number system is called counting. Human being happens to have 10 fingers, five in each hand, which has resulted in the use of the base 10 or decimal number system by most cultures. In fact the word digit comes from the Latin digitus, meaning finger.

By regular way, the inputs add each column and bring the carries over to the next column (Cn + An + Bn = {Cn+1, SUMn}). This calculation can also be done if we separately produce the sum and carry bits and add them at the end. Independently, for each column produce a sum and carry bit with a normal full adder.

A	01100	12
+ *B*	11001	19
+ *C*	00110	6
Sum bits	10011	25
Carrybits	01100	6 * 2
Final result	110111	37

Figure 2.3: The example calculation of CSA

In the binary system, there are only two digits, 0 and 1. Binary digits are called bits (for binary digit). The value of each digit position in a binary numbers is exactly double that of the position to its right and exactly one-half that of the position to its left. A 0 and 1 can mean the numerals 0 and 1, or 0 can means false and 1 can mean true. 0 can mean off and 1 can mean on. The interpretations we may assign to the symbols 0 and 1 are endless [35].

In any number the digit in the leftmost position is called the most significant digit because it carries the most value in the number. The rightmost digit is called the least significant digit because it carries the least value in the number.

The hexadecimal number system has a base of sixteen; that is, it is composed of 16 numeric and alphabetic characters [37]. Most digital systems process binary data in groups that are multiples of four bits, making the hexadecimal very convenient because each hexadecimal digit represents a 4-bit binary number as listed in Table 2.2.

Table 2.2: Number System

Decimal	Binary	Hexadecimal
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Hexadecimal number system is used primarily as a compact way of displaying or writing binary numbers because it is very easy to convert between binary and hexadecimal. As we probably aware, long binary numbers are difficult to read and write because it is easy to drop or transpose a bit. Since computers and microprocessors understand only 1s and 0s, it is necessary to use these digits when we program in machine “language”. Hexadecimal is widely used in computer and microprocessor applications. Figure 2.3 shows the calculations for binary (refer to equation 1.0) and also hexadecimal which refer to equation 2.0.

2.4 Basic Function of Boolean Algebra in Full Adder (FA)

Carry Save adder is standing by multiple bit of full adder. Figure 2.4 shows a basic full adder (FA) gates logic [12]. Figure 2.5 [13] and figure 2.6 [11] [15] shows a modified carry save adder (CSA) gate logic [14]. The basic full adder operation of figure 2.4 can be stated as follows as in Equation 3 and Equation 4.

$$Sum = A \oplus B \oplus Cin \tag{3.0}$$

$$Cout = XiYi + XiCin + YiCin$$
$$Cout = XiYi + (Xi + Yi)Cin \tag{4.0}$$

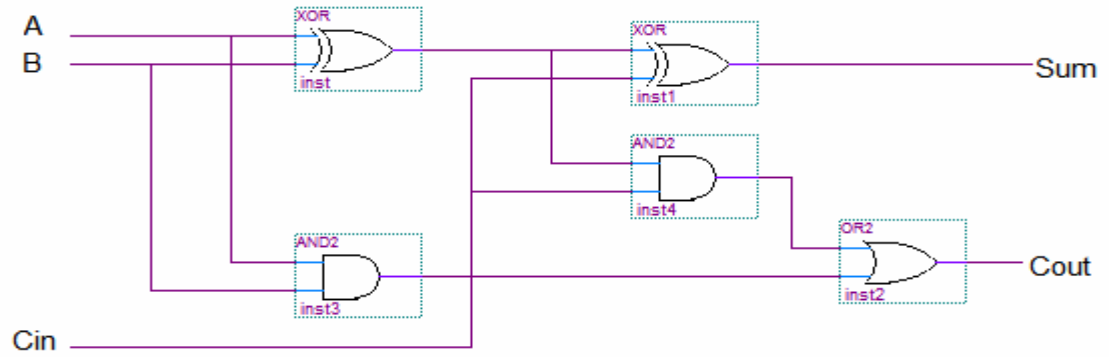


Figure 2.4: Full adder using XOR, OR and AND gate

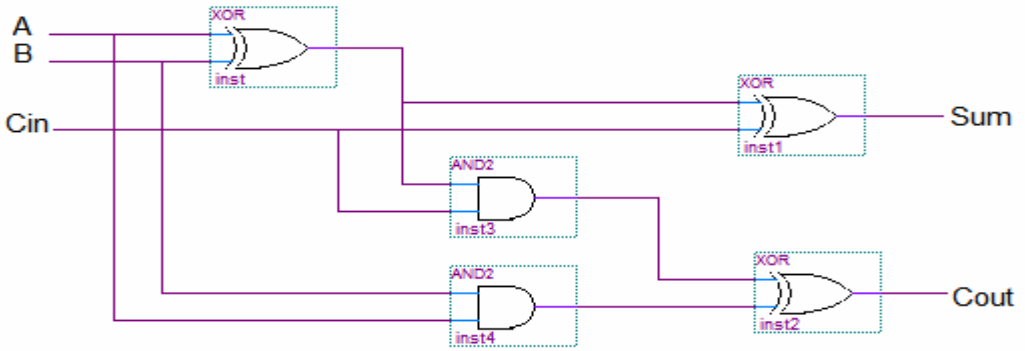


Figure 2.5: Full adder using AND and XOR gate

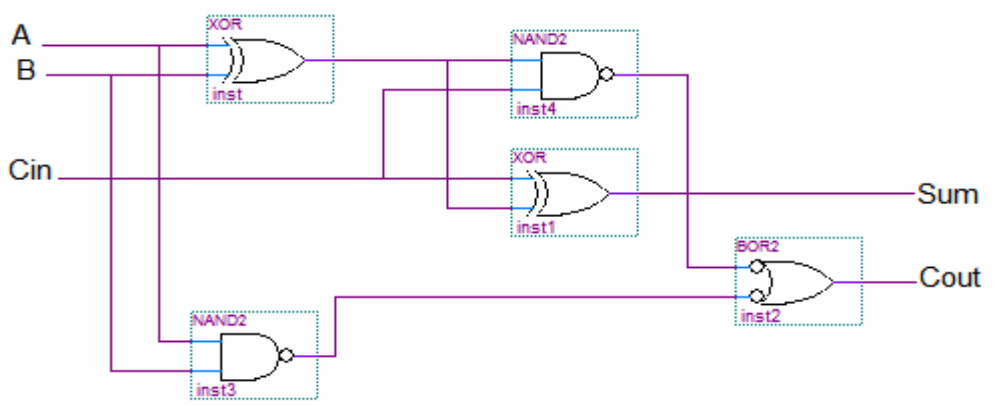


Figure 2.6: Full adder using XOR, NAND and negative-OR gate logic

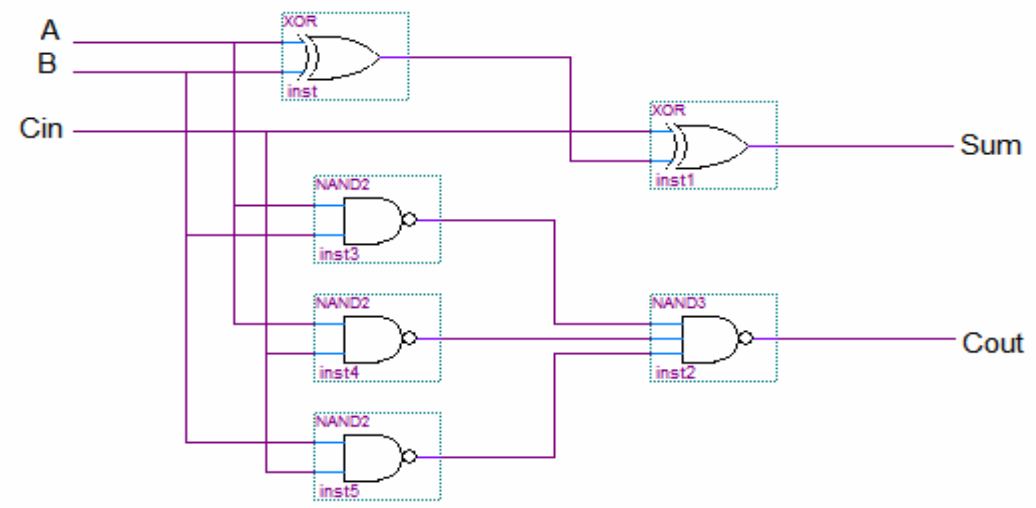


Figure 2.7: Full adder using XOR and NAND gate

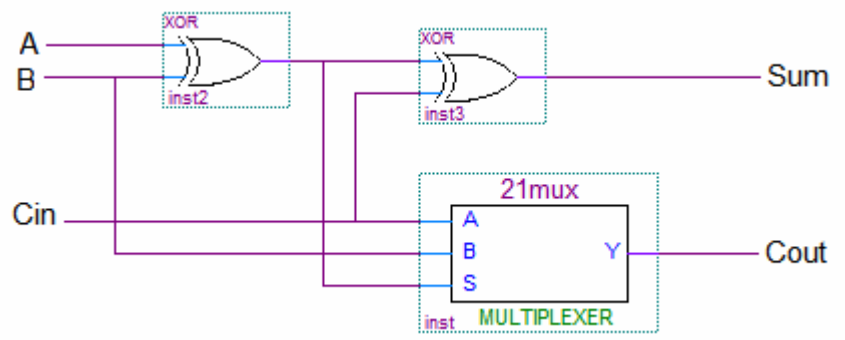


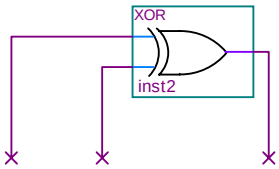
Figure 2.8: Full adder using XOR gate and 2 to 1 Multiplexer

The full adder circuit in the figure 2.5, 2.6, 2.7 and 2.8 that using different kind of gate logics has been calculating manually and the results are same as the circuit design in the figure 2.4 that can be seen in the Table 2.3. All of the gate logics in the block diagram of full adder can be calculate manually one by one based on its truth table as shown in Table 4a, 4b, 4c and 4d.

Table 2.3: FA truth table

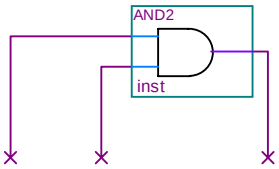
INPUT			OUTPUT	
X	**Y**	**Ci**	**Ci+1**	**Si**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 2.4a : AND gate truth table



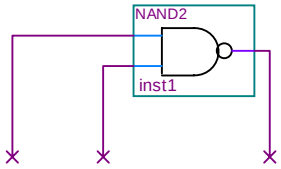
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.4b : negative-OR gate truth table



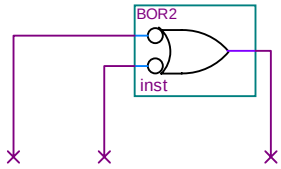
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.4c : NAND gate truth table



0	0	1
0	1	1
1	0	1
1	1	0

Table 2.4d : XOR gate truth table



0	0	1
0	1	1
1	0	1
1	1	0

2.5 Multilevel Carry-Save Adder

Multilevel CSA use a number of CSAs interconnected as a multilevel adder tree to add more than one number per cycle. The number of level of CSA tree determines the basic cycle time of the addition process [3] [4] [5] [7].

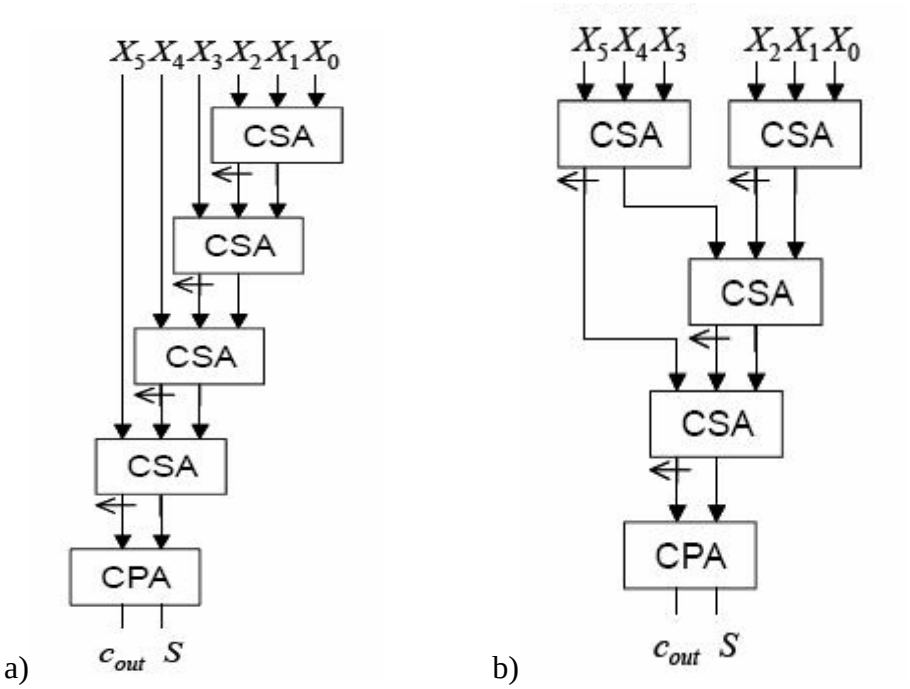


Figure 2.9: CSA tree

We define a *CSA* tree to be a tree of CSA operators and one adder at the root of the tree. A CSA tree can be used to transform an arbitrary number of additions to produce two adding operands and the adder is used at the root of CSA tree to produce a final sum [8] [9] [10]. Figure 2.9 shows the 6 inputs CSA in two versions. Figure 2.9a is the basic CSA tree while Figure 2.9b is the CSA tree in Wallace tree.

A better way to organize the CSAs, and reduce the operation time, is in the form of a tree commonly called Wallace tree [3] [11]. In this tree, the number of operands is reduced by a factor of 2/3 at each level. Let $\lambda(l)$ be the maximal number of operands that can be added by an l-level CSA tree. $\lambda(1)=3$. Each CSA has 3 inputs and two outputs, hence the number of output times (3/2) will be the number of inputs, that is, the number of outputs in the upper level. If the number of outputs is not a multiple of 2, then it mod 2 indicates the number of extra outputs in the upper level. Hence $\lambda(l)$ can be defined recursively as in Equation 5.0. Or else, the easier way, the equation 6.0 can be use.

$$\lambda(l) = \left\lfloor \frac{\lambda(l-1)}{2} \right\rfloor \times 3 + \lambda(l-1) \bmod 2 \tag{5.0}$$

$$k.(\frac{2}{3})^l \le 2$$
$$number\ of\ level \approx \frac{\log(k/2)}{\log(3/2)} \tag{6.0}$$

For different values of l can, the maximal numbers of input operands that a CSA tree can add are listed in Table 2.5.

One of the major speed enhancement techniques used in modern digital circuits is the ability to add numbers with minimal carry propagation. Based on the Table 5.0, we could know how many level of CSA we need to design based on the input operand. For example, if we want to add 9 numbers together, we can use 3 CSAs to reduce it to 6 numbers; and then reduce 6 numbers to 4 numbers and then reduce it to 3 numbers. The basic idea is that three numbers can be reduced to 2, in a 3:2 compressor or also known

as (3,2) counters, by doing the addition while keeping the carries and the sum separate. This means that all of the columns can be added in parallel without relying on the result of the previous column, creating a two-output "adder" with a time delay that is independent of the size of its inputs.

Table 2.5: The number of level in a CSA tree for k operands

l	$\lambda(l)$
1	3
2	4
3	6
4	9
5	13
6	19
7	28
8	42
9	63
10	94

2.6 Ripple Carry Adder

When multiple full adders are used with the carry ins and carry outs chained together then this is called a ripple carry adder because the correct value of the carry bit ripples from one bit to the next (refer to figure 2.10) [16]. It is possible to create a logical circuit using several full adders to add multiple-bit numbers. Each full adder inputs a C_{in}, which is the C_{out} of the previous adder. This kind of adder is a ripple carry adder, since each carry bit "ripples" to the next full adder. Note that the first (and only the first) full adder may be replaced by a half adder.

The layout of a ripple carry adder is simple, which allows for fast design time; however, the ripple carry adder is relatively slow, since each full adder must wait for the carry bit to be calculated from the previous full adder. The gate delay can easily be

calculated by inspection of the full adder circuit. Following the path from C_{in} to C_{out} shows 2 gates that must be passed through.

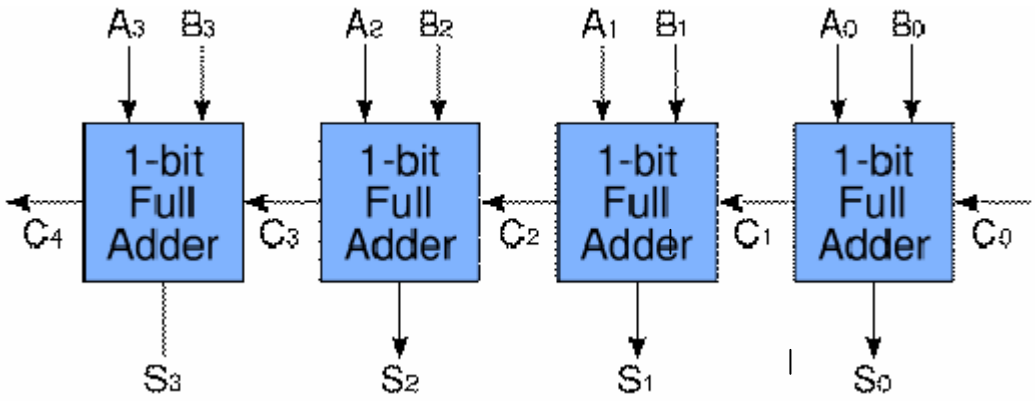


Figure 2.10: 4-bit ripple carry adder circuit diagram

2.7 Design Circuit

Figure 2.11 shows the basic design of CSA that has 4 operands in 2 levels of CSA. Operand means a quantity upon which a mathematical operation is performed. Given in

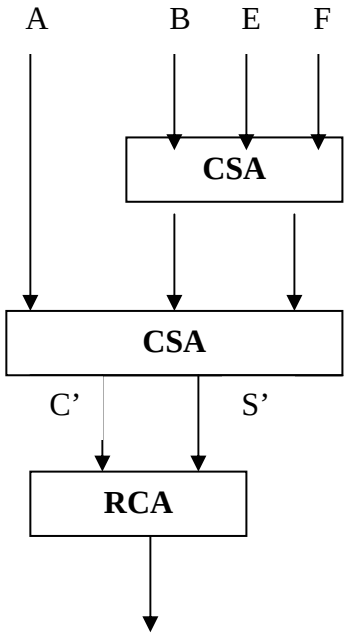


Figure 2.11: Basic Design of n-bit CSA

Figure 2.12, is a 6 input 4-bit CSA tree. Notice that each input or output line represents n single-bit lines. Each CSA block includes n full adders with no flip-flops. The left pointed arrows on the carry output lines indicate that the carries are shifted left for one bit position before being fed to the next stage and a 0 is entered into the LSB position and so on. At the bottom of the tree, a CPA (which using Ripple carry adder (RCA) here) is required to add the sum vector and carry vector together. When the word length n is very long, the ripple carry propagation in the final stage will significantly degrade the performance of CSA. We can connect multiple levels of CSAs in a tree fashion to add k numbers or operands simultaneously where k=4, 5, 6, 7, 8 and 9. Figure 2.12 shows the clear addition method based on the CSA tree block diagram. The motive is to generate the design circuit result easily by manual.

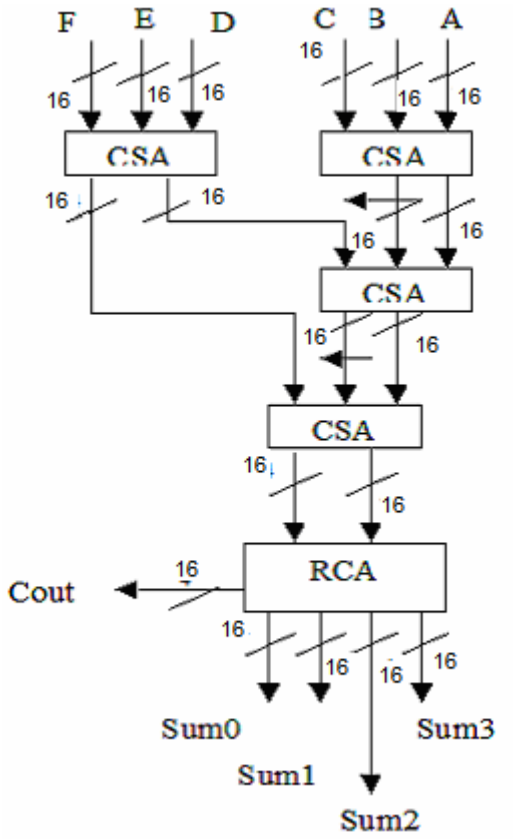


Figure 2.12: 6 operands of 16-bit CSA using (3,2) counter

2.8 Delay

2.8.1 CSA

To add k numbers, (k-2) CSAs are required with each additional input operand increasing the number of CSAs by one. Hence the area complexity of the CSA tree is in equation 5.0. The total delay time of a CSA tree is dependent on the number of levels in the tree. Each level of CSAs contributes $2\Delta_g$ to the propagation delay, which is the delay time of a full adder. Hence, in equation 6.0, *l* is the number of levels [3] [5].

$$A_{CSA} = (k-2)A_{CSA} + A_{CPA}) \tag{5.0}$$

$$\Delta_{CSA-Tree} = 2l\Delta_g + \Delta_{CPA} \tag{6.0}$$

When adding together three or more numbers, using a carry-save adder followed by a ripple carry adder is faster than using two ripple carry adders. This is because a ripple carry adder cannot compute a sum bit without waiting for the previous carry bit to be produced, and thus has a delay equal to that of *n* full adders. A carry-save adder, however, produces all of its output values in parallel, and thus has the same delay as a single full-adder. Thus the total computation time (in units of full-adder delay time) for a carry-save adder plus a ripple carry adder is $n + 1$, whereas for two ripple carry adders it would be $2n$ [32].

2.8.2 Ripple Carry Adder

The latency of a k-bit ripple carry adder can be derived by considering the worst-case signal propagation path [3] [5]. As shown in figure 2.6.a, the critical path usually begins at the A_o or Y_o input, proceeds through the carry-propagation chain to the leftmost FA, and terminates at the S_{k-1} output. Of course, it is possible that for some FA implementations, the critical path might begin at C_k and/or terminate at C_k. However, given that the delay from the carry-

in to carry-out is more important than from X to carry-out or from carry-in to S, full adder designs often minimize the delay from carry-in to carry-out, making the critical path. We can thus write the following equation 7.0 for the latency of a k-bit ripple carry adder, where T_{FA} (input->output) represents the latency of a full adder on the path between its specified input and output [32]. As an approximation to the foregoing, we can say that the latency of a ripple carry is $kT_{FA.}$

$$T_{ripple-add} = T_{FA}(x, y \rightarrow Cout) + (k-2) \times T_{FA}(Cin \rightarrow Cout) + T_{FA}(Cin \rightarrow S) \quad (7.0)$$

2.9 Applications of CSA

Carry-save arithmetic, well known from multiplier architectures, can be used for the efficient CMOS implementation of a much wider variety of algorithms for high-speed digital signal processing than multiplication [17]. Carry save adder has efficient concepts for implementation of high speed. [18] [19] [20].

Carry save adders applied in the partial product lines of an array multiplier circuit used to speed-up the summation of the partial products in order to speed-up the carry propagation along the array [21] [22]. One of main time saving techniques used in the fastest designs is the use of carry save adders to combine the partial products into final answer.

Carry-save represented in joint module selection and retiming optimization as well as optimizes technique. The use of carry-save signal representation is a powerful technique in the high-speed implementation of arithmetic circuits that solve the joint module selection and retiming problem [23].

2.10 Advantages

A carry save adder (CSA) is very fast where there is no carry propagation within each CSA cell. It is only the final recombination of the final carry and sum requires a carry propagating addition [24] that simply outputs the carry bits instead of propagating them to the side. Since it save all the carries from all the adds to the last stage and do one Carry Look-ahead Adder (CLA) or Ripple Carry Adder (RCA) at the end. Thus, using carry-save adders avoids carry propagation and will result in a higher throughput. [Minimum adder integer multipliers using Carry Save Adder [21]. The CSA design automatically avoids the delay in the carryout bits [26]. It is well known that carry-save arithmetic is a useful technique in the implementation of high-speed arithmetic functions. Carry save adder used widely in design because carry save addition saves logic and time.

2.11 Characteristics of high-speed

2.11.1 Transistor Sizing

Increasing the transistor size improves the speed of the circuit, also power dissipation increases since the load capacitance increases [26].

2.11.2 Propagation Delay

When gate inputs change, outputs don't change instantaneously. This delay is known as "gate" or "propagation" delay. The delay of the logic gates depends on the width of the transistors in gate. The CSA implementation becomes much faster and also relatively smaller in size than the implementation of other normal basic adders [26][27].

2.11.3 CMOS Technology

In 2002, Mohammed Sayed and Wael Badawy [28] and in 2004, Amir Ali Khatibzadeh and Kaamran Raahemifar [29] present that the XOR and transmission gate full adder using 0.18μm CMOS Technology is the fastest. XOR and transmission gate is in the family of Static CMOS. It has 14 transistors, good noise margins, fast, and superior in power consumption, insensitive to device variation. In 2006, T. Vigneswaran, B. Mukundhan, and P. Subbarami Reddy present high-speed new design of the fourteen Transistor (14T) adder based on Static energy recovery full (SERF) adder. It produced better result in threshold loss, speed and power which using channel length of 1.5μ and a channel width of 1.9μ with 1.2volt logic. It has 45% higher speed and reduces 50% threshold loss problem [30].

2.11.4 Pipelining

In 2004, Vinesh Sukumar, Dong Pan, Kevin Buck, Herbert Hess, Harry Li, Dave Cox, M.M.Mojarradi [31] present that to increase the frequency of operation, pipelining is considered. As the frequency of operation is increased, the cycle time measured in gate delays continues to shrink. Pipelining has emerged as the design technique of choice that helps to achieve high throughput digital systems. This technique breaks down a single complex computational block into discrete blocks separated by clock storage elements CSE -like flip-flops, latches. Pipelining improves throughput at the expense of latency, however once the pipe is filled we can expect one data item per unit of time. The gain in speed is achieved by clocking sub-circuits faster and also achieves path delay equalization by inserting registers. As result, it achieve performance gains also the propagation delay and delay variation decreasing. The project used the applications of pipeline to achieve the objective.

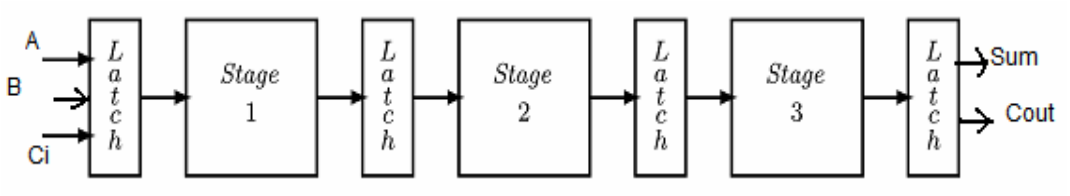


Figure 2.13: Pipeline applications in 16-bit CSA

Figure 2.13 shows how pipeline applications in CSA circuit based on CSA per stage. The design is in 3 stages of 6 operands 16-bit CSA. Latches between stages 1 and 2 store intermediate results of step 1 "Used by stage 2 to execute step 2 of algorithm". Stage 1 starts executing step 1 on next set of operands X,Y. Pipeline was just another transformation which is adding the delay and retiming it based on clock using D-flip-flop.

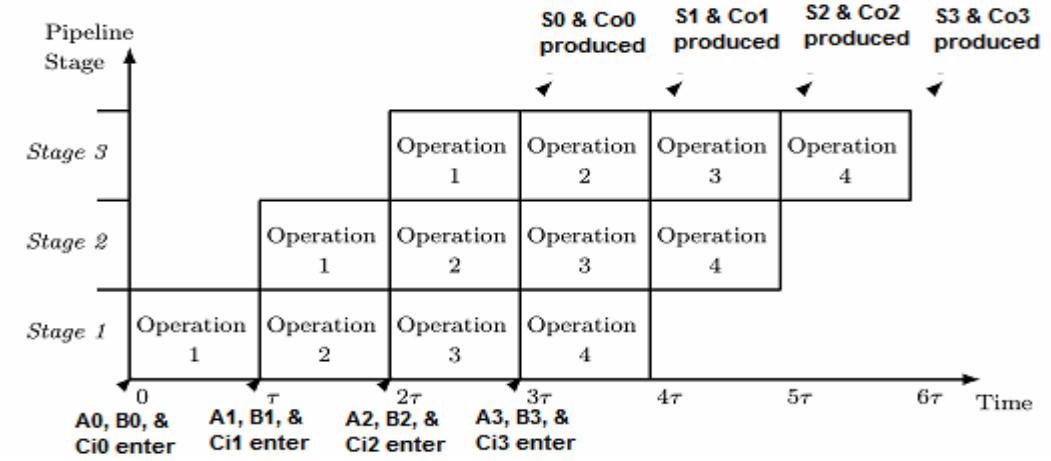


Figure 2.14: Pipelining Timing Diagram

Pipeline shows how it reduces delay by multiple are overlap in execution. Based on the figure 2.14, when the inputs were given, the operation 1 execute at the 0 time in ladder. The process transforms continuously at the end of 3τ times at the stage 3 of pipeline. Without pipeline, the operation 2 would execute at the 3τ times. But in this diagram, the operation 2 execute next to the operation 1 has begun. Thus,

the delay can be reduced. The process continuously executes per stage as explained.

In digital circuits, the *flip-flop* is an electronic circuit which has two stable states and thereby is capable of serving as one bit of memory. A flip-flop is controlled by one or two control signals and/or a gate or clock signal. The output often includes the complement as well as the normal output. As flip-flops are implemented electronically, they naturally also require power and ground connections.

Flip-flops can be either simple or clocked. Clocked devices are specially designed for synchronous (time-discrete) systems and therefore ignores its inputs except at the transition of a dedicated clock signal (known as clocking, pulsing, or strobing). This causes the flip-flop to either change or retain its output signal based upon the values of the input signals at the transition. Some flip-flops change output on the rising edge of the clock, others on the falling edge.

These flip flops are very useful, as they form the basis for shift registers, which are an essential part of many electronic devices. The advantage of this circuit over the D-type latch is that it "captures" the signal at the moment the clock goes high, and subsequent changes of the data line do not matter, even if the signal line has not yet gone low again. The D flip-flop can be interpreted as a primitive delay line or zero-order hold, since the data is posted at the output one clock cycle after it arrives at the input. It is called delay flip flop since the output takes the value in the Data-in.. The corresponding of truth table shown in Table 2.6

Table 2.6 : Truth Table of D Flip-Flop

D	**Q**	>	$\mathbf{Q_{next}}$
0	X	Rising	0
1	X	Rising	1