CHAPTER 2

LITERATURE REVIEW

One of the ways on how to speed up multiplication is to use more adders to speed the accumulation of partial products. The best-known method for speeding up the accumulation is the Wallace Tree multiplier, which is an adder tree built from carry-save adders, which is simply an array of full adders whose carry signal are not connected, as in the early stages of the array multiplier [8].

2.1 Wallace Tree Multiplier

The Wallace tree multiplier is considerably faster than a simple array multiplier because its height is logarithmic in the word size, not linear [8]. However, in addition to the large number of adder required, the Wallace tree's wiring is much less regular and more complicated. As a result, Wallace trees are often avoided by designers, whom design complexity, is a consideration. Callaway *et. al* [8] also evaluated the power consumption of multipliers. They compared an array multiplier and a Wallace Tree multiplier, and found that the Wallace tree multiplier used significantly less power for bit widths between 8 and 32, with the advantage of the Wallace tree growing as word length increased [8]. Wallace Tree styles use a log-depth tree network for reduction. Faster, but irregular, they trade ease of layout for speed. Although the speed-size tradeoffs for these two styles are fairly well characterized, the power tradeoffs are not well understood. Wallace tree styles are best

avoided for low power applications, since the excess wiring is likely to consume extra power. While substantially faster than the carry-save structure for large multiplier word lengths, the Wallace Tree multiplier has the disadvantage of being very irregular, which complicates the task of coming up with an efficient layout. The irregularity is visible even in the 4-bit implementation [5].

The Wallace multiplier is a high speed multiplier. Figure 2.1 shows 8-bits x 8-bits high speed Wallace Tree multiplier design. The summing of the partial product bits in parallel using a tree of carry-save adders became generally known as the “Wallace Tree”. Three step processes are used to multiply two numbers.

- Formation of the bit products.
- Reduction of the bit product matrix into a two row matrix by means of a carry-save adder.
- Summation of the remaining two rows using a fast carry-propagate adder (ripple-carry adder) to produce the product.

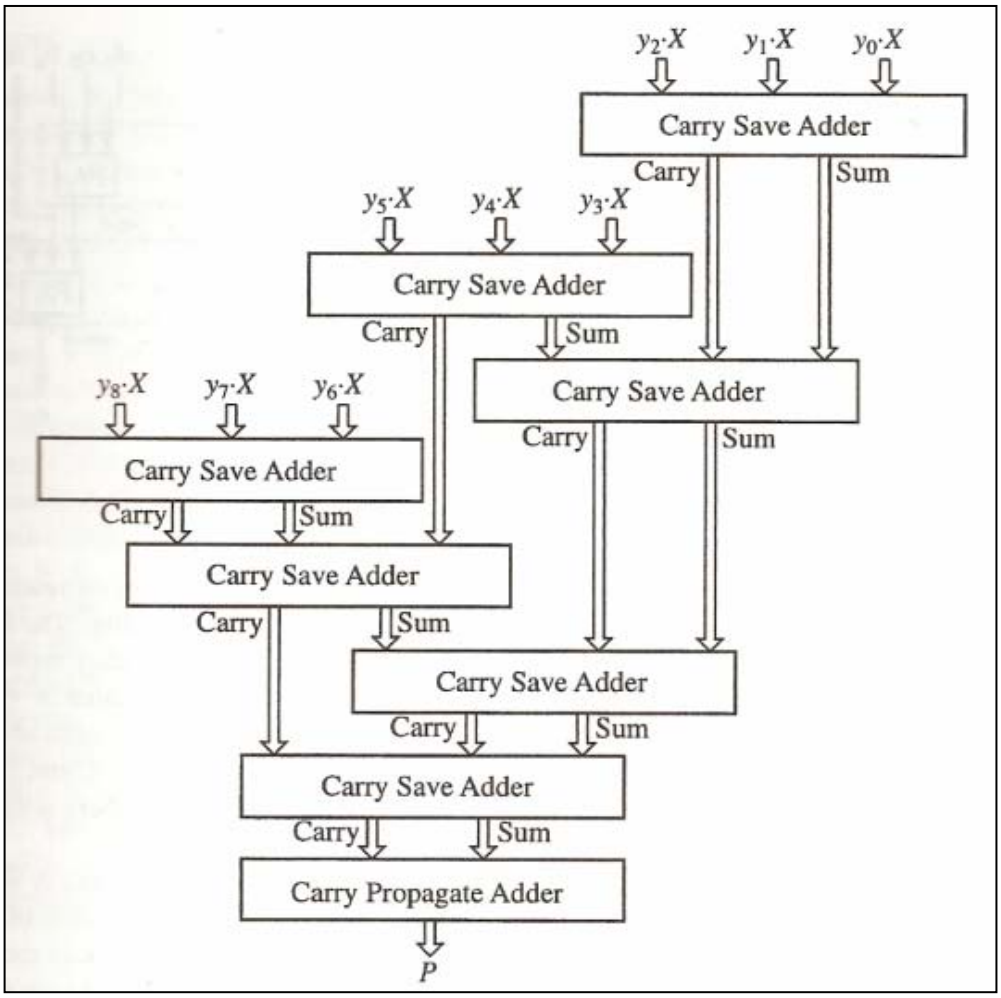

Figure 2.1: An 8-bits x 8-bits high speed Wallace Tree multiplier design [3]

2.2 D Flip-flops

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 [20].

Flip-flops can be either simple or clocked. Simple flip-flops consist of two cross-coupled inverting elements – transistors, or NAND, or NOR-gates – perhaps augmented by

some enable/disable (gating) mechanism. Clocked devices are specially designed for synchronous (time-discrete) systems and therefore ignore its inputs except at the transition of a dedicated clock signal (known as clocking or pulsing). 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.

Flip-flops can be further divided into types that have found common applicability in both asynchronous and clocked sequential systems: the SR ("set-reset"), D ("data"), T ("toggle"), and JK types are the common ones; all of which may be synthesized from (most) other types by a few logic gates [20]. The behavior of a particular type can be described by what is termed the characteristic equation, which derives the "next" (i.e., after the next clock pulse) output, Q_{next}, in terms of the input signal(s) and/or the current output, Q.

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 data flip-flop since the output takes the value in the Data (D) [20]. The D flip-flops are used as a clocking mechanism to get the speed when using the timing analyzer tools in the Altera Quartus II software. It is also used as the pipelined stages in this design to increase the speed of the 8-bits x 8-bits high speed Wallace Tree multiplier.

The characteristic equation and the truth table of the D flip-flop are as below:

$$\mathbf{Q_{next} = D}$$

Table 2.1: Truth table for D flip-flop

D	Q	Clock	Q_{next}
0	X	Rising edge	0
1	X	Rising edge	1

2.3 Carry-save Adder (CSA)

When three or more operands are to be added simultaneously (e.g. in multiplication) using two-operand adders, the time consuming carry-propagation must be repeated several times. If the number of operands is *k*, then carries have to propagate (*k*-1) times. Several techniques for multiple operand addition that attempt to lower the carry-propagation penalty have been proposed and implemented. The technique that is most commonly used is carry-save addition. In carry-save addition, we let the carry propagate only in the last step, while in all the other steps we generate a partial sum and a sequence of carries separately. A carry-save adder (CSA) is therefore, capable of reducing the number of operands to be added from 3 to 2, without any carry propagation.

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 combination of two half adder or a single full adder. Figure 2.2 and Table 2.2 shows the logic circuit and the truth table for a 1-bit half adder.

Logic expression for a 1-bit half adder:

- **sum = a ⊕ b**
- **c_out = ab**

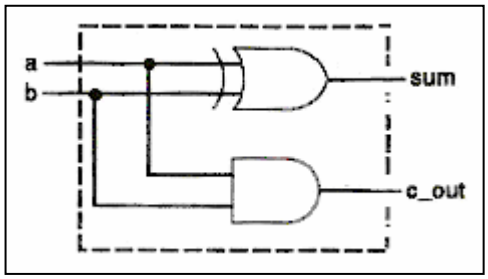

Figure 2.2: Logic circuit for a 1-bit half adder

Table 2.2: Truth table for a 1-bit half adder

a	b	sum	c_out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Figure 2.3 and Table 2.3 shows the logic circuit and the truth table for a 1-bit full adder.

Logic expression for a 1-bit full adder:

- **sum = a ⊕ b ⊕ c_in**
- **c_out = ab + (a ⊕ b)c_in**

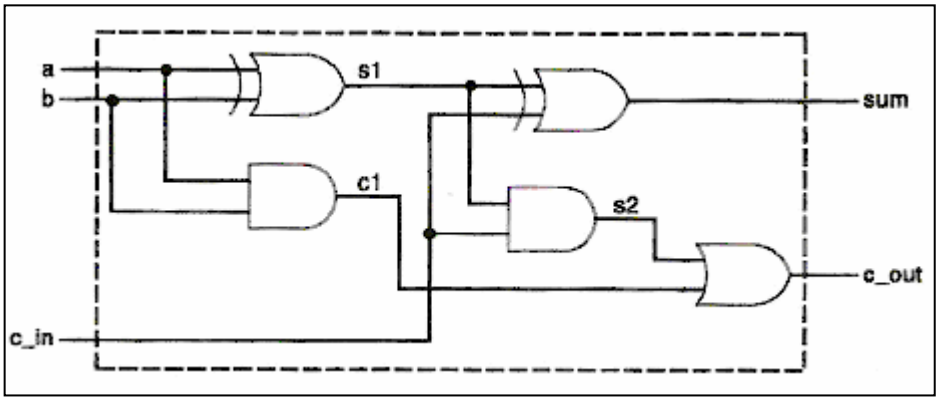

Figure 2.3: Logic circuit for a 1-bit full adder

Table 2.3: Truth table for a 1-bit full adder

a	b	c_in	sum	c_out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The basic CSA accepts three *n*-bit operands and generates two *n*-bit results; an *n*-bit partial sum and an *n*-bit carry as in Figure 2.4.

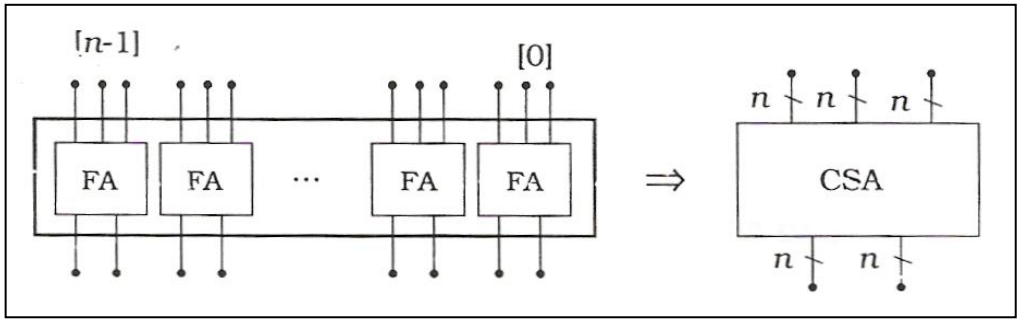

Figure 2.4: Creation of an *n*-bit carry-save adder

A carry-save adder is effectively a “1’s counter” that adds the number of 1’s on the inputs and encodes them on the sum and carry outputs as summarized in the table below.

Table 2.4: A carry-save adder as a 1’s counter

A	**B**	**C**	**Carry**	**Sum**	**Number of 1’s**
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	2
1	0	0	0	1	1
1	0	1	1	0	2
1	1	0	1	0	2
1	1	1	1	1	3

A carry -save adder is therefore also known as a (3, 2) counter, Figure 2.5, because it converts three inputs into a count encoded in two outputs. The carry-out is passed to the next more significant column, while a corresponding carry-in is received from the previous column. Therefore, for simplicity, a carry is represented as being passed directly down the column. The Wallace Tree multiplier requires $[\log_{3/2}(N/2)]$ levels of (3, 2) counters to reduce N inputs down to 2 carry-save redundant from outputs. Unfortunately, the routing between levels becomes much more complicated. The longer wires have greater wire capacitance and the irregular tree is difficult to layout.

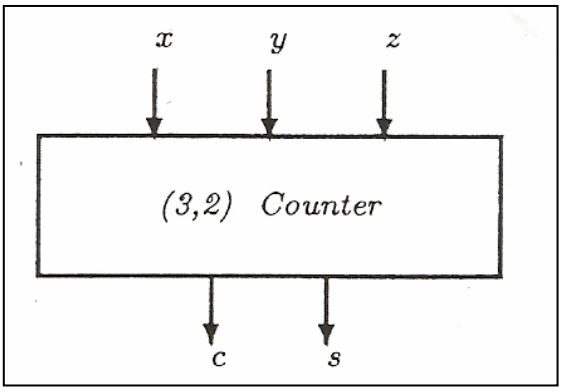

Figure 2.5: Carry-save adder (CSA) or also known as a (3, 2) counter

A [4:2] compressor as in Figure 2.6 can be used in a binary tree to produce a much more regular layout [12]. It takes four inputs of equal weight and produces two outputs. It can be constructed from two (3, 2) counters. Along the way, it generates an intermediate carry into the next column and accepts a carry from the previous column, so it may more aptly be called a (5, 3) counter [12]. The regular layout and routing also make the binary tree attractive.

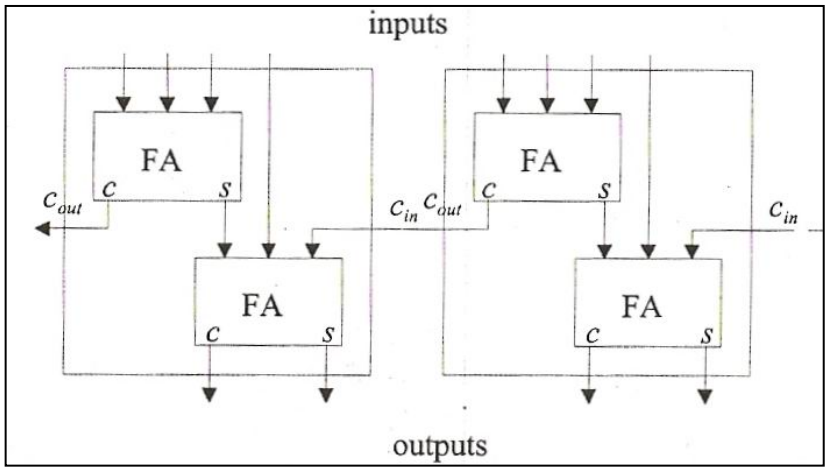

Figure 2.6: 4-2 compressor [12]

2.4 Ripple-carry Adder (RCA)

The addition of two operands is the most frequent operation in almost any arithmetic unit. A two-operand adder is used not only when performing additions and subtractions, but also often employed when executing more complex operations like multiplication and division. Consequently, a fast two operand adder is essential.

The most straightforward implementation of a parallel adder for two operands x_{n-1}, $x_{n-2}\ldots$, x_0 and y_{n-1}, $y_{n-2}\ldots$, y_0 is through the use of n basic units called full adders (FA). In a parallel arithmetic unit, all $2n$ input bits (x_i and y_i) are usually available to the adder at the same time. However, the carries have to propagate from the full adder in position 0 (the position of the full adder whose inputs are x_0 any y_0) to position i in order for the full adder in that position to produce the correct sum and carry out bits. In other words, we need to wait to wait until the carries ripple through all n FAs before we can claim that the sum outputs are correct and may be used in further calculations. Because of this, the parallel adder shown in Figure 2.7 is called a ripple-carry adder. Note that the full adder in position i, being a combinational circuit, will see an incoming carry $c_i = 0$ at the beginning of the operation, and will accordingly produce a sum bits s_i. The incoming carry c_i may change later on, resulting in a corresponding change in s_i. Thus, a ripple effect can be observed at the sum outputs of the adder as well, continuing until the carry propagation is complete.

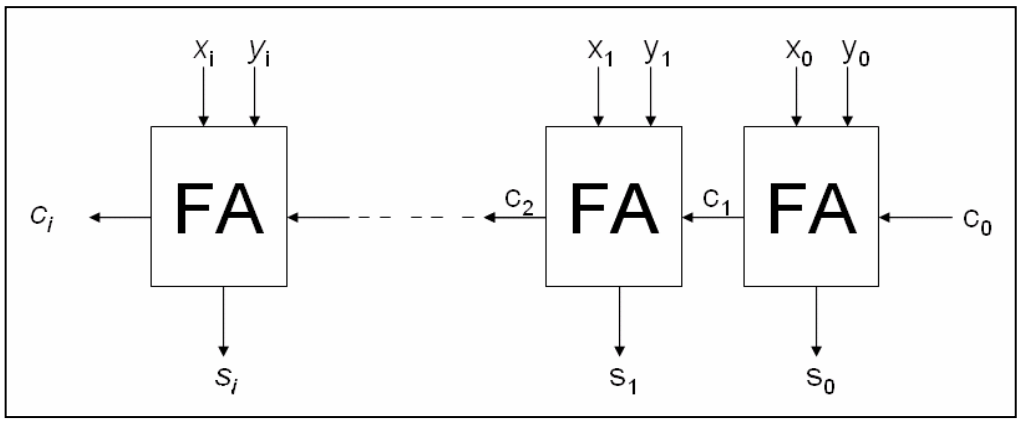

Figure 2.7: A ripple-carry adder (RCA)