IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 5, Issue 6, Ver. I (Nov -Dec. 2015), PP 01-06
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Comparison among Different Adders

Prof. Rashmi Rahul Kulkarni[1]
[1](*Electronics and Telecommunication, Finolex Academy of Management and Technology, India)*

Abstract: *Addition is most commonly performed arithmetic operation.Adder is basic building block of most of digital systems. Improvement in speed of adder indirectly improves speed of system. Hence careful design optimization is required. VHDL coding of different adders is done and comparative analysis is made. Each adder has its own positives and negatives in terms of speed and area. Various adders are designed using VHDL. Then, they are simulated and synthesized using Xilinx ISE 9.2i for Spartan 3E family device with speed grade -5.*
Keywords – *Adder, Carry Look Ahead, Carry Save Adder, Ripple Carry Adder, FPGA*

I. Introduction

In Processors adder is an important element. As such, extensive research continues to be focused on improving the power-delay performance of the adder. In VLSI implementations, parallel-prefix adders are known to have the best performance. Reconfigurable Field Programmable Gate Arrays (FPGAs) has been gaining in popularity in recent years because it offers improved performance in terms of speed and power over DSP-based and microprocessor-based solutions for many practical designs and a significant reduction in development time and cost over Application Specific Integrated Circuit (ASIC) designs.

Fast and accurate operation of digital system depends on the performance of adders [5]. Hence improving the performance of adder is the main area of research in system design. Arithmetic (such as addition, subtraction, multiplication and division) performed in a program, additions are required to increment the program counter and to calculate the effective address . show that in a prototypical RISC machine (DLX) 72% of the instructions perform additions (or subtractions) in the data path. Over the last decade many different adder architectures were studied and proposed to speed up the binary additions.

Performance of different adders is discussed. Several adder structures are implemented and characterized on a FPGA including Ripple Carry Adder (RCA) and the Carry Look Ahead Adder (CLA).

II. Ripple Carry Adder

Arithmetic operations like addition, subtraction, multiplication, division are basic operations to be implemented in digital computers using basic gates like AND, OR, NOR, NAND etc. Among all the arithmetic operations if we can implement addition then it is easy to perform multiplication (by repeated addition), subtraction (by negating one operand) or division (repeated subtraction). Half Adders can be used to add two one bit binary numbers and Full adders to add two three bit numbers.The block diagram of 4-bit Ripple Carry Adder is shown here below in Figure.1.-

It is possible to create a logical circuit using multiple full adders to add *N*-bit numbers. Each full adder inputs a C_{in}, which is the C_{out} of the previous adder. This kind of adder is called 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 (under the assumption that $C_{in} = 0$). The block diagram of 4-bit Ripple Carry Adder is shown here below –

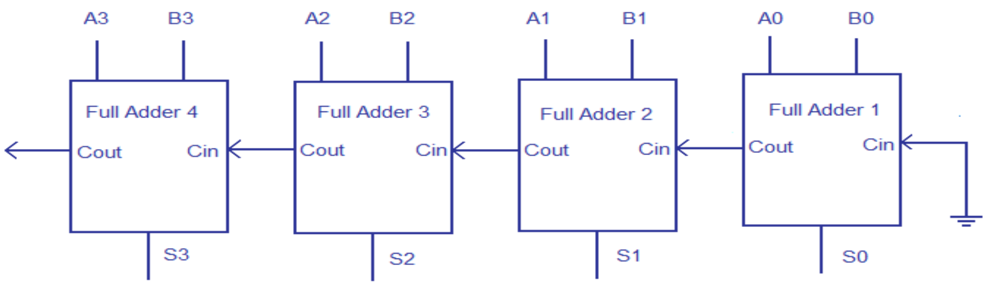

Fig.1. 4 bit Ripple Carry Adder

The layout of a ripple-carry adder is simple, which allows for fast design time; however, the ripple-carry adder is relatively slow when number of stages get increased[2][3], 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. Each full adder requires three levels of logic. In a 32-bit ripple-carry adder, there are 32 full adders, so the critical path (worst case) delay is 3 (from input to carry in first adder) + 31 * 2 (for carry propagation in later adders) = 65 gate delays.

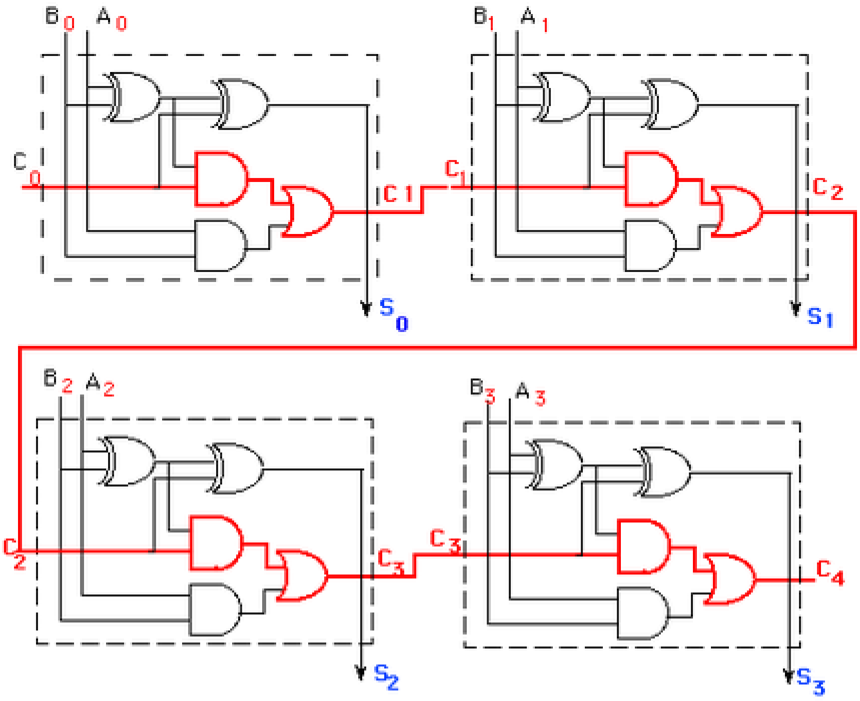

Fig.2. Ripple-carry adder, illustrating the delay of the carry bit.

The disadvantage of the ripple-carry adder is that it can get very slow when one needs to add many bits.

III. Carry Look Ahead Adder

To reduce the computation time, there are faster ways to add two binary numbers by using carry look ahead adders. They work by creating two signals P and G known to be Carry Propagator and Carry Generator. The carry propagator is propagated to the next level whereas the carry generator is used to generate the output carry , regardless of input carry. The Figure shows the full adder circuit used to add the operand bits in the i th column; namely Ai & Bi and the carry bit coming from the previous column (Ci).

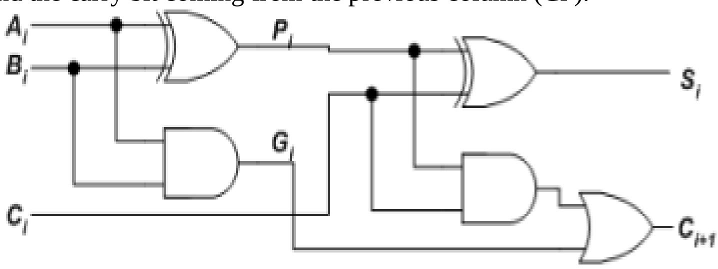

Fig.3. Full Adder using two Half Adders.

In this circuit, the 2 internal signals Pi and Gi are given by:

Propagate Term = Pi = Ai ⊕Bi ……………………..(1)

Generate Term = Gi = AiBi ……………….………...(2)

The output sum and carry can be defined as :

Si = Pi ⊕ Ci ………………………………………...(3)

$C_{i+1} = G_i + P_iC_i$ ……………………………………...(4)

where i =0,1,…….., n 1. Equation (4) can be further expanded into

Ci+1 = gi+pigi-1+………..+pipi-1……p1g0+pipi-1……p0C0………(5)

the carry-look ahead scheme can be built in the form of a tree-like circuit, which has a simple, regular structure ,Gi is known as the carry Generate signal since a carry (Ci+1) is generated whenever Gi =1, regardless of the input carry (Ci). Pi is known as the carry propagate signal since whenever Pi =1, the input carry is propagated to the output carry, i.e., Ci+1. = Ci. Computing the values of Pi and Gi only depend on the input operand bits (Ai & Bi) as clear from the Figure and equations. Thus, these signals settle to their steady-state value after the propagation through their respective gates. Computed values of all the Pi's are valid one XOR-gate delay after the operands A and B are made valid. Computed values of all the Gi's are valid one AND-gate delay after the operands A and B are made valid. The Boolean expression of the carry outputs of various stages can be written as follows:

C1 = G0 + P0C0 ………………………………………………………………..(6)
C2 = G1 + P1C1 = G1 + P1 (G0 + P0C0) ……………………………………...(7)
= G1 + P1G0 + P1P0C0
C3 = G2 + P2C2 = G2 + P2G1 + P2P1G0 + P2P1P0C0 ……………………….(8)
C4 = G3 + P3C3 = G3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1P0C0……....(9)

In general, the i th. carry output is expressed in the form Ci = Fi(P's, G's , C0). In other words, each carry signal is expressed as a direct SOP function of C0 rather than its preceding carry signal. Since the Boolean expression for each output carry is expressed in SOP form, it can be implemented in two-level circuits. The 2-level implementation of the carry signals has a propagation delay of 2 gates, i.e., 2τ.

IV. Carry Save Adder

There are many cases where it is desired to add more than two numbers together. The straightforward way of adding together m numbers (all n bits wide) is to add the first two, then add that sum to the next, and so on. This requires a total of m − 1 additions, for a total gate delay of O(m lg n) (assuming lookahead carry adders). Instead, a tree of adders can be formed, taking only O(lg m · lg n) gate delays.

Using carry save addition, the delay can be reduced further still. The idea is to take 3 numbers that to add together, x + y + z, and convert it into 2 numbers c + s such that x + y + z = c + s, and do this in O(1) time. The reason why addition can not be performed in O(1) time is because the carry information must be propagated. In carry save addition, we refrain from directly passing on the carry information until the very last step.

To add three numbers by hand, typically align the three operands, and then proceed column by column in the same fashion that perform addition with two numbers. The three digits in a row are added, and any overflow goes into the next column. when there is some non-zero carry, it is like adding four digits (the digits of x,y and z, plus the carry).

carry:		1	1	2	1	
x:		1	2	3	4	5
y:		3	8	1	7	2
z:	+	2	0	5	8	7
sum:		7	1	1	0	4

The carry save approach breaks this process down into two steps. The first is to compute the sum ignoring any carries:

x:		1	2	3	4	5
y:		3	8	1	7	2
z:	+	2	0	5	8	7
s:		6	0	9	9	4

Each si is equal to the sum of xi + yi + zi modulo 10. Now, separately, compute the carry on a column by column basis:

x:		1	2	3	4	5
y:		3	8	1	7	2
z:	+	2	0	5	8	7
c:		1	0	1	1	

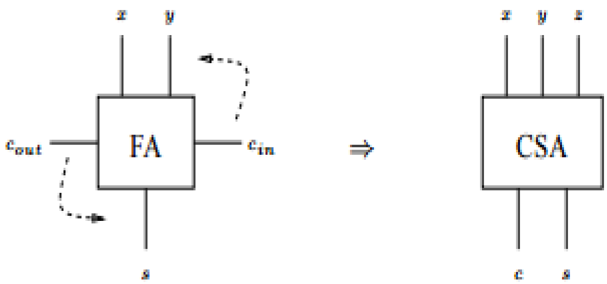

Fig.4. Carry Save Adder block same as Full adder

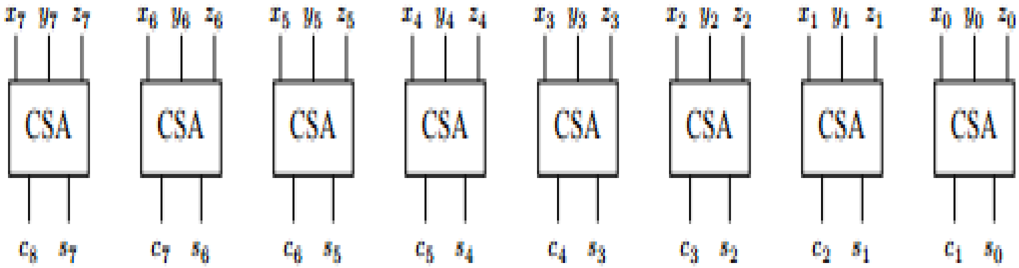

Fig.5. One CSA block is used for each bit.

This circuit adds three n = 8 bit numbers together into two new numbers.

s:		6	0	9	9	4
c:	+	1	0	1	1	
sum:		7	1	1	0	4

The important point is that c and s can be computed independently, and furthermore, each ci (and si) can be computed independently from all of the other c's (and s's). This achieves original goal of converting three numbers that to add into two numbers that add up to the same sum, and in O(1) time.

The same concept can be applied to binary numbers. As a quick example:

x:			1	0	0	1	1
y:			1	1	0	0	1
z:	+		0	1	0	1	1
s:			0	0	0	0	1
c:	+	1	1	0	1	1	
sum:		1	1	0	1	1	1

A carry save adder simply is a full adder with the cin input renamed to z, the z output (the original "answer" output) renamed to s, and the cout output renamed to c. Figure 2 shows how n carry save adders are arranged to add three n bit numbers x,y and z into two numbers c and s. The CSA block in bit position zero generates c1, not c0. Similar to the least significant column when adding numbers by hand (the "blank"), c0 is equal to zero. All of the CSA blocks are independent, thus the entire circuit takes only O(1) time. To get the final sum, it need a RCA, which will cost O(lg n) delay. The asymptotic gate delay to add three n-bit numbers is thus the same as adding only two n-bit numbers. So how long does it take to add m different n-bit numbers together? The simple approach is just to repeat this trick approximately m times over. This is illustrated in Figure 3. There are m−2 CSA blocks (each block in the figure actually represents many one-bit CSA blocks in parallel) and then the final RCA. Every time when pass through a CSA block, number increases in size by one bit. Therefore, the numbers that go to the RCA will be at most n + m − 2 bits long. So the final RCA will have a gate delay of O(lg (n + m)). Therefore the total gate delay is O(m + lg (n + m)) Instead of arranging the CSA blocks in a chain, a tree formation can actually be used.

V. Performance Analysis

Below in table 1.,table 2. table 3. Show delay and area utilized by different adders, for family Spartan 3E(90 nm process technology) and with speed grade -5.

TABLE 1.

Simulation Of Different Adders For Addition Of Two Numbers, Each One Is 8 Bit Long

Adder name	Area			Delay ns
	Slices	LUTs	IOBs	
Carry look ahead	9	15	25	12.025
Ripple carry	9	15	25	12.675

TABLE 2

Simulation Of Different Adders For Addition Of Three Numbers, Each One Is 8 Bit Long

Adder name	Area			Delay ns
	Slices	LUTs	IOBs	
Carry save with Ripple carry at last stage	17	30	34	13.245
Carry save with Carry look ahead at last stage	17	30	34	13.367
Carry look ahead	18	31	34	13.398
Ripple carry	18	31	34	13.476

TABLE 3

Simulation Of Different Adders For Addition Of Four Numbers, Each One Is 8 Bit Long

Adder name	Area			Delay ns
	Slices	LUTs	IOBs	
Carry save with Ripple carry at last stage	25	43	42	14.338
Carry save with Carry look ahead at last stage	26	46	42	14.377
Carry look ahead	26	46	42	14.349
Ripple carry	26	46	42	14.466

Verification is carried out by ISE simulator. Simulation results of proposed design is shown in following figures.

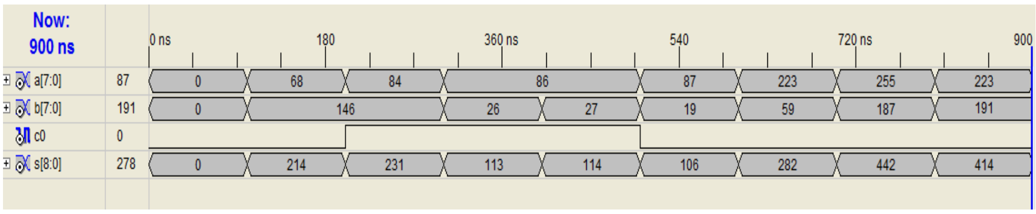

Fig.6. Simulation result for addition of two 8 bit numbers

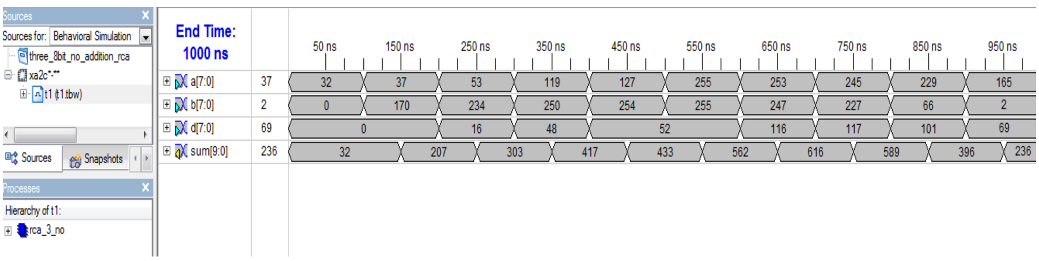

Fig.7. Simulation result for addition of three 8 bit numbers

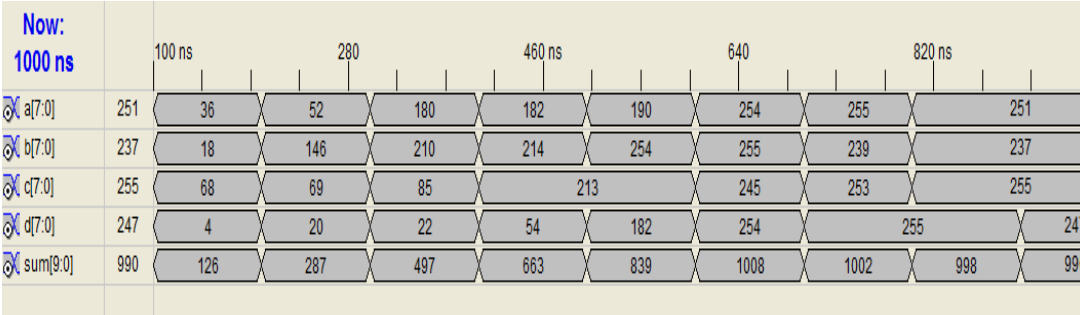

Fig.8. Simulation result for addition of two 8 bit numbers

VI. Conclusion

After observing results of comparisons, for two 8bit numbers addition, carry look ahead adder is better. For three and four 8 bit numbers addition carry save adder with last stage built by ripple carry adder is preferable. In future work, it is needed to design unique adder which provides low area as well as delay in order to meet the needs of current industry. Further, this work can be extended by designing and simulating the adders with increased number of bits such as 16bits, 32bits and 64 bits.

References

Journal Papers:

[1] R.UMA, Vidya Vijayan , M. Mohanapriya, Sharon Paul," Area, Delay and Power Comparison of Adder Topologies", International Journal of VLSI design & Communication Systems (VLSICS) Vol.3, No.1, February 2012.

[2] Animul islam, M.W. Akram, S.D. pable ,Mohd. Hasan, "Design and Analysis of Robust Dual Threshold CMOS Full Adder Circuit in 32 nm Technology", International Conference on Advances in Recent Technologies in Communication and Computing,2010.

[3] Deepa Sinha, Tripti Sharma, k.G.Sharma, Prof.B.P.Singh, "Design and Analysis of low Power 1-bit Full Adder Cell",IEEE, 2011.

[4] V.Krishna Kumari, Y.Sri Chakrapani," Designing and Characterization of koggestone, Sparse Kogge stone, Spanning tree and Brentkung Adders" International Journal of Modern Engineering Research (IJMER) Vol. 3, Issue. 4, July-august. 2013 pp-2266-2270

[5] Pakkiraiah Chakali, Madhu Kumar Patnala," Design of High Speed Kogge-Stone Based Carry Select Adder " International Journal of Emerging Science and Engineering (IJESE) ISSN: 2319–6378, Volume-1, Issue-4, February 2013.

[6] B. Ramkumar, Harish M Kittur, "Low –Power and Area-Efficient Carry Select Adder", IEEE transaction on very large scale integration (VLSI) systems, vol.20, no.2, pp.371-375, Feb 2012.

[7] J. Lim, D. G. Kim, and S. I. Chae, "A 16-bit carry-lookahead adder using reversible energy recovery logic for ultra-low-energy systems", IEEE Journal of Solid-State Circuits, 1999, vol. 34, pp. 898-903.

Books:

[8] Mano, M. M. and C. R. Kime, Logic and computer design fundamentals, Prentice-Hall, 2001.

[9] Weste, N. H. E. and K. Eshraghian, Principles of CMOS VLSI Design: A Systems Perspective 2/E, Addison-Wesley, 1998.