International Journal of Engineering Research-Online
A Peer Reviewed International Journal
Articles available online http://www.ijoer.in

Vol.3., Issue.5., 2015
(Sept.-Oct.)

RESEARCH ARTICLE

ISSN: 2321-7758

ANALYSIS & SIMULATION OF DIFFERENT 32 BIT ADDERS

SHAHZAD KHAN, Prof. M. ZAHID ALAM, Dr. RITA JAIN
Department of Electronics and Communication Engineering, LNCT, Bhopal, (MP)

ABSTRACT
Adders are the used as digital components in digital circuit design. In this paper, various adder structures are used to execute addition such as serial and parallel structures and most of researches have done research on the design of high-speed, low-area, or low power adders. Adders like ripple carry adder, carry select adder, carry look ahead adder, carry skip adder exist numerous adder implementations each with good attributes and some drawbacks. This paper focuses on the delay comparison of these adders. We have recorded the performance improvements in propagating the carry and generating the sum when compared with the traditional carry look ahead adder designed in the same technology.

INTRODUCTION

Adders are widely used in generic computer for adding data in the processor, it is also commonly used in various electronic applications e.g. digital signal processing to perform various algorithms. In past, the major challenge for VLSI designer is to reduce area of chip by using efficient optimization techniques and then the next phase is to increase the speed of the operation to achieve fast calculations. Arithmetic logic unit is the main component of central processing unit, where the addition, multiplication, comparison and other logical operations are performed. There are three performance parameter on which a VLSI designer has to optimize their design, which are Area, Speed, and Power. Moreover, there are various types of adders such as Ripple Carry Adder (RCA), Carry-Look ahead Adder (CLA), Carry Select Adder (CSA), Carry-Bypass Adder or Carry Skip Adder (CSK) discussed.

RIPPLE CARRY ADDER (RCA)

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. 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. 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. A design with alternating carry polarities and optimized AND-OR-Invert gates can be about twice as fast.
Multiple full adder circuits can be cascaded in parallel to add an N-bit number. For an N- bit parallel adder, there must be N number of full adder circuits. A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. It is called a ripple carry adder because each carry bit gets rippled into

the next stage. In a ripple carry adder the sum and carry out bits of any half adder stage is not valid until the carry in of that stage occurs. Propagation delays inside the logic circuitry are the reason behind this. Propagation delay is time elapsed between the application of an input and occurrence of the corresponding output.

Consider a NOT gate, When the input is "0" the output will be "1" and vice versa. The time taken for the NOT gate's output to become "0" after the application of logic "1" to the NOT gate's input is the propagation delay here. Similarly the carry propagation delay is the time elapsed between the application of the carry in signal and the occurrence of the carry out (Cout) signal.

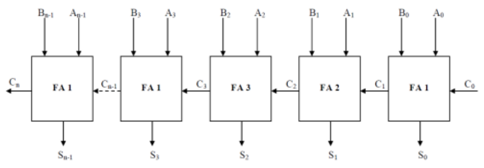


Fig. 1: Ripple Carry Adder

CARRY LOOK AHEAD ADDER

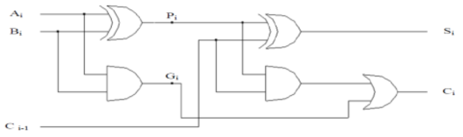


Fig. 2: Full Adder Block

From the above figure of full adder

$P_i = A_i \text{ EXOR } B_i$

$G_i = A_i . B_i$

$S_i = P_i \text{ EXOR } C_{i-1} = A_i \text{ EXOR } B_i \text{ EXOR } C_{i-1}$

$C_i = G_i + P_i.C_{i-1}$

The logic variable G_i is known as a carry generate and is independent of input carry. The logic variable P_i is known as a carry propagate because this is the term associated with the propagation of the carry from C_{i-1} to C_i.

Now,

$C_{i+1} = G_{i+1} + P_{i+1}.C_i$

$= G_{i+1} + P_{i+1} [G_i + P_i.C_{i-1}]$

$C_{i+1} = G_{i+1} + P_{i+1}.G_i + P_{i+1}.P_i.C_{i-1}$

Therefore, for 4-bit parallel adder

$P_0 = A_0 \text{ EXOR } B_0$

$G_0 = A_0.B_0$

$C_0 = G_0 + P_0.C_{-1}$

$C_1 = G_1 + P_1.G_0 + P_1.P_0.C_{-1}$

$C_2 = G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.C_{-1}$

$C_3 = G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 + P_3.P_2.P_1. P_0.C_{-1}$

The carry generate G_i variables can be generated directly from A_i and B_i inputs and the carry generate P_i variables are obtained by EXORing A_i and B_i inputs and C_{-1} is the carry input. Logic diagram of a look ahead carry generator shown in fig.2; if all the logic variables are available simultaneously, the carry outputs are implemented by using 2-level logic realization. Look ahead carry utilizes logic gates to look at the lower order bits of the augend and addend to see if a higher order carry is to be generated. The propagation delay time through the adder is considerably reduced and it becomes independent of the number of full-adders in circuit.

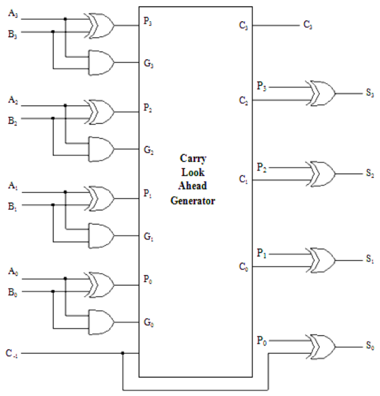


Fig. 3: 4-bit adder with look ahead carry generator

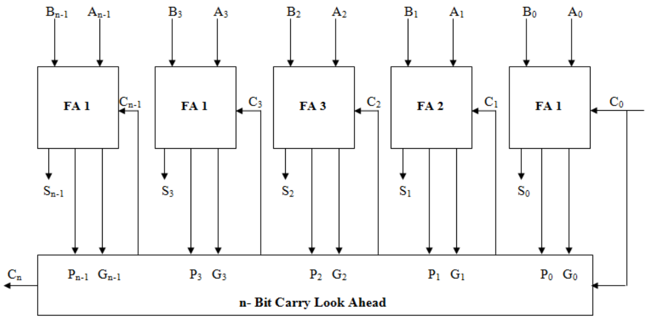


Fig. 4: n – bit Adder with Carry Look Ahead

CARRY-SKIP ADDER (RCA)

A carry-skip adder (also known as a carry-bypass adder) is an adder implementation that improves on the delay of a ripple-carry adder. The two addends are split in blocks of n bits. The output carry of each block is dependent on the input carry only if, for each of the n bits in the block, at least one addend has a 1 bit. The output carry Co_{i+n-1}, for the block corresponding to bits *i* to *i+n-1* is obtained from a multiplexer, wired as follows:

$SEL = (A_i + B_i) (A_{i+1} + B_{i+1}) ..... (A_{i+n-1} + B_{i+n-1})$

A = $C_{ripple,\ i+n-1}$ (the carry output for the ripple adder summing bits *i* to *i+n-1*)

B = $C_{out,\ i-1}$

This greatly reduces the latency of the adder through its critical path, since the carry bit for each block can now "skip" over blocks with a group propagate signal set to logic 1 (as opposed to a long ripple-carry chain, which would require the carry to ripple through each bit in the adder). With a RCA, if the input bits Xi and Yi are different for all position i, then the carry signal is propagated at all positions and the addition is completed when the carry signal has propagated through the whole adder. In this case, the RCA is as slow as it is large. Actually, RCA are fast only for some configurations of the input words, where carry signals are generated at some positions. Depending on the position at which a carry signal has been generated, the propagation time can be variable. Carry Skip Adders take advantage both of the generation or the propagation of the carry signal. They are divided into blocks, where a special circuit detects quickly if all the bits to be added are different (P_i = 1 in the entire block). The signal produced by this circuit will be called block propagation signal. If the carry is propagated at all positions in the block, then the carry signal entering into the block can directly bypass it and so be transmitted through a multiplexer to the next block. As soon as the carry signal is transmitted to a block, it starts to propagate through the block, as if it had been generated at the beginning of the block. Figure 5 shows the structure of a 16-bits skip carry adder, divided into 2, 4 and 8-blocks. It becomes now obvious that there exist a trade-off between the speed and the size of the blocks. In this part we analyze the division of the adder into blocks of equal size. We denote k1 the time needed by the carry signal to propagate through an adder cell (one FA propagates), and k_2 the time it needs to skip over one block. Suppose the N-bit Carry Skip Adder is divided into M blocks, and each block contains P adder cells. The actual addition time of a RCA depends on the configuration of the input words. The completion time may be small but it also may reach the worst case, when all adder cells propagate the carry signal. In the same way, we must evaluate the worst carry propagation time for the Carry Skip Adder. The worst case of carry propagation is for X_i=1 and Y_i=0 and C_{in}=1 inputs.

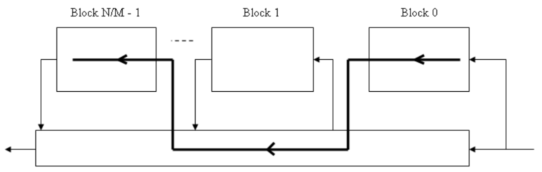


Fig. 5: Carry Skip Logic

Let an Example: M = 4 bits; N = 16 bits.

If P(j) = 1 (Propagation); then Group(j) will be skipped

X(j) : m-bits of group (j)

Y(j) : m-bits of group (j)

Cin(j) : Carry in to group(j)

Cout(j) = Cin(j+1) : Carry out of group(j) = Carry in to next group(j+1). (j) : Group(j) consisting of m-bits numbers to add

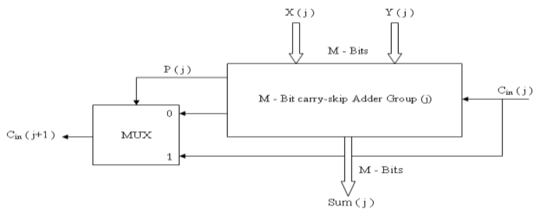


Fig. 6: Carry Skip Adder Block diagram

consider addition of the following numbers

..... $a_{k+1}\ a_{k+2}\ a_k$ 010101 $a_{l+2}\ a_{l+1}\ a_l$ ....

..... $b_{k+1}\ b_{k+2}\ b_k$ 101010 $b_{l+2}\ b_{l+1}\ b_l$ ....

If C_{l+3} = 1 then carry will propagate to position k; to speed-up operation, propagation is skipped to position i without waiting for rippling operation time varies according to operands as in carry-complete addition to implement carry-skip adder, stages are divided into blocks and carry-skip logic is added to each block to detect when carry-in the block can be passed directly to the next block define carry transfer

$T_i = a_i + b_i$

carry skipping can be detected for a block size of m as follows (carry propagates through all stages):

$T_j \cdot T_{j+1} \ldots\ldots T_{j+m-1} = 1 \quad (= (a_j + b_j) \cdot (a_{j+1} + b_{j+1}) \ldots)$

note: this takes into account both propagated and generated carries!

Carry out from the block (m-bits in a block) is

$T_j \cdot T_{j+1} \ldots\ldots\ldots\ldots T_{j+m-1} \cdot C_j + C_{j+m}$

Skipped Generated

Block size in carry-skip adder is very important; worst case operation time takes place when carry is generated in the first block; carry skips intermediate stages and carry is killed in the last block.

Worst case addition time is T; where (n = adder width, m = block size)

In practice, non-uniform block sizes give the best performance but in general, outer blocks should be smaller than middle blocks.

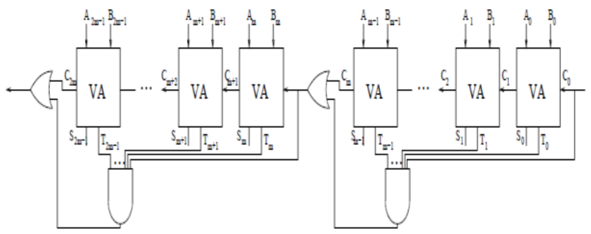


Fig. 7: Carry Skip Adder Circuit

CARRY-SELECT ADDER (CSA)

The carry-select adder generally consists of two ripple carry adders and a multiplexer. Adding two n-bit numbers with a carry select adder is done with two adders (therefore two ripple carry adders) in order to perform the calculation twice, one time with the assumption of the carry being zero and the other assuming one. After the two results are calculated, the correct sum, as well as the correct carry, is then selected with the multiplexer once the correct carry is known. The number of bits in each carry select block can be uniform, or variable. In the uniform case, the optimal delay occurs for a block size of √n. When variable, the block size should have a delay, from addition inputs A and B to the carry out, equal to that of the multiplexer chain leading into it, so that the carry out is calculated just in time. The o(√n) delay is derived from uniform sizing, where the ideal number of full-adder elements per block is equal to the square root of the number of bits being added, since that will yield an equal number of MUX delays.

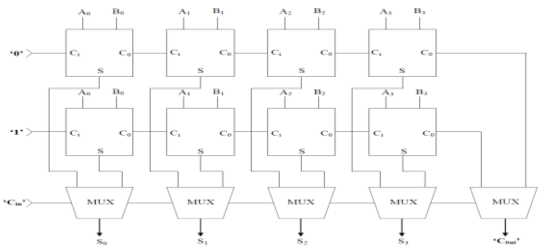
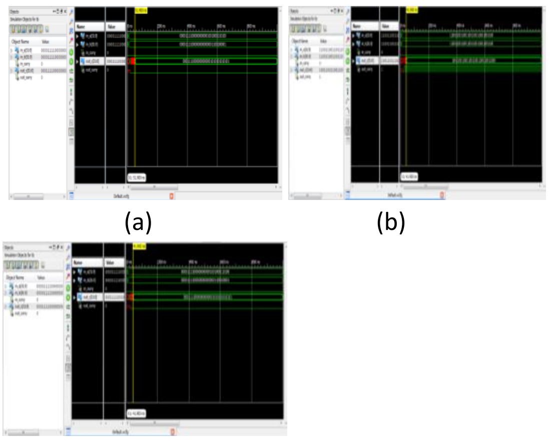


Fig. 8: Carry Select Adder

Above is the basic building block of a carry-select adder, where the block size is 4. Two 4-bit ripple carry adders are multiplexed together, where the resulting carry and sum bits are selected by the carry-in. Since one ripple carry adder assumes a carry-in of 0, and the other assumes a carry-in of 1, selecting which adder had the correct assumption via the actual carry-in yields the desired result. A 16-bit carry-select adder with a uniform block size of 4 can be created with three of these blocks and a 4-bit ripple carry adder. Since carry-in is known at the beginning of computation, a carry select block is not needed for the first four bits. The delay of this adder will be four full adder delays, plus three MUX delays.

SIMULATION RESULTS



(a) (b)

(c)

Fig. 9: Simulation Results
(a) CLA (b) Carry Skip (c) CSA

CONCLUSION

This work has presented improved 32-bit conditional sum adders for high-speed low-power applications. The capacitance of the multiplexer network tree is reduced, yielding benefits in power consumption and operation speed. It was found that the proposed adders always outperformed the old ones in these circuit implementations. These improved 32-bit adders can reduce the power delay product by 10% to 25% and reduce the layout area by about 20%.

Table: Delay Comparison between various adder circuits (in nanoseconds, ns)

ADDER	CLA	Carry Skip	CSA
DELAY	41.900	31.900	31.900

REFERENCES

[1]. Pierre, L. "VHDL description and formal verification of systolic multipliers. In *CHDL*", N. Holland, 1993.

[2]. A. Kaldewaij, "*Programming: The Derivation of Algorithms*", Prentice-Hall, 1990.

Vol.3., Issue.5., 2015
(Sept.-Oct.)

[3]. John G. Proakis and Dimitris G. Manolakis (1996), “Digital Signal Processing: Principles,. Algorithms and Applications”, Third Edition.

[4]. A. D. Booth, “A signed binary multiplication technique”. *Quart. J. of Mech. Appl. Math*, 4(2), 1951.

[5]. Volnei A. Pedroni: “*Circuit Design and Simulation with VHDL”, Second edition,* PHI.

[6]. [6] BROWN, Stephen D., “Fundamentals of Digital Logic with VHDL design”, Boston: McGraw-Hill, 2000.

[7]. Peter J. Ashenden, “*The Designer’s Guide to VHDL”*, Morgan Kaufmann Publishers, 95 Inc., 1996.

[8]. V. Kantabulra, “Designing optimum one-level carry-skip adders,” IEEE Transactions on Computers, vol. 42, no. 6, pp. 759-764, June 1993.

[9]. Beebe, H.F. Nelson, “Floating Point Arithmetic, Computation in Modern Science & Technology”, December, 2007.

[10]. N. Weste, D. Harris, “CMOS VLSI Design”, Third Edition, Addison Wesley.