

A New Design for High Speed and High-Density Carry Select Adders

Reza Hashemian, Senior member IEEE
Northern Illinois University,
Department of Electrical Engineering
Reza@ceet.niu.edu

Abstract- An algorithm with logarithmic carry propagation delay is developed for high-density adders. The design procedure is introduced for the construction of a 64-bit adder with maximum path delay equivalent to 8 gate delays. The algorithm is based on the carry select technique with operands partitioned into very fine slices for both quick response and low gate counts. Another property observed in this algorithm is resource (hardware) sharing which is due to regularity of the carry channel structure. The design is coded in Verilog, simulated and implemented using XC4010E Xilinx FPGA technology.

I. INTRODUCTION

The binary addition has been a subject of extensive study for many years. Among them Carry Look-ahead, Carry-Select, and other parallel schemes have been particularly studied at length for high-speed addition [1-9]. Our objective here is to develop an algorithm for high speed and high-density adders with reduced hardware suitable for limited gate integrated circuits such as Field Programmable Gate Arrays (FPGA). Designing with FPGA technology, however, has its own features and advantages [10], and may very well be different from other conventional design strategies. For example, designs which use precharging techniques, those using pass transistors or transmission gates as switches, or the use of the domino logic [11] and similar design techniques used in ASIC/CMOS technology may not be as attractive for FPGA in its present stage of development. FPGAs are very reliable, cost effective and quick to the market. FPGAs are electronically re-configurable which makes them essential in algorithm development and design verification resulting in fast prototyping.

The proposed algorithm is originally implemented for the construction of a 64-bit adder, but it could be as efficient for other sizes of adder/subtractor devices. In general, for a 2^i - bit adder the proposed design operates with a maximum path delay not exceeding $i+2$ gate delays, where a gate delay is assumed to be equivalent to a two input XOR gate or a 2-to-1 multiplexer.

The algorithm is based on the well-known carry-select adder technique [3], where the operands are sliced for partial additions with both choices of 0 and 1 input carries. Here, for quick computation of partial sums and carries the slices are selected to be only a single bit long. This choice of slices reduces the time delay associated with each partial addition equivalent to a single gate (two input XOR gate) delay. This selection also reduces the overall hardware in

the adder, and optimally uses 2-to-1 multiplexers, definitely preferable in FPGA technology. Note also that, we are getting the entire partial sum and carry terms produced simultaneously and after one gate delay. However, one disadvantage in this small size slicing is that the actual carry (*carry flag*) selection process becomes a timely process, and the carry generation/propagation channel (similar to that experienced in a typical adder) becomes long for high density adders. This is specifically worse if we adopt the normal linear carry generation/propagation process, where the carry propagation delay is proportional to the operand size. Here, in an effort to reduce the carry generation/propagation delays, we propose a logarithmic approach that simultaneously produces carries in groups rather than individually. It is shown that, the reduction in the carry propagation delay associated with this logarithmic approach is more significant for higher density codes. In fact, as indicated, the delay increase for each doubling of the size of the operands is equal to one gate (2-to-1 multiplexer) delay. In general, to generate all carry flags, including the output carry, from m number of slices of partial adders, it takes a maximum delay equivalent to delay of $k = \log_2(m)$ number of 2-to-1 multiplexers connected in series. For instance, for a 64-bit adder with single bit slices, it takes 6 gate delays to generate all 64 carry flags.

II. ALGORITHM AND DESIGN ARCHITECTURE

Consider the addition operation

$$S = A + B \quad (1)$$

Where, both A and B are 64-bit operands, i.e., $A = a_{63} a_{62} \dots a_1 a_0$ and $B = b_{63} b_{62} \dots b_1 b_0$. For the i th bit the partial addition is presented by:

$$\begin{aligned} m_i &= a_i \oplus b_i \\ O_i &= a_i \cap b_i \\ n_i &= \overline{a_i \oplus b_i} = \overline{m_i} \\ I_i &= a_i \cup b_i \end{aligned} \quad (2)$$

Where, m_i and O_i are the primary sum and carry terms for 0 input carry, and n_i and I_i are the primary sum and carry terms for 1 input carry, all respectively. Note that we get the entire partial sum and carry terms within one gate delay. This is because all the sums and carries are generated simultaneously.

Now, to get the final value for the sum term, S_i , as well as for the carry flag, C_i , we need to use the preceding carry flag, C_{i-1} , and make the selection, as depicted in Eq.(3).

$$\begin{aligned} S_i &= m_i \bar{C}_{i-1} + n_i C_{i-1} \\ C_i &= O_i \bar{C}_{i-1} + I_i C_{i-1} \end{aligned} \quad (3)$$

Equation (3) shows a typical and sequentially generated carry flag scheme, where it starts from the LSB and carries are generated sequentially in ascending order. Our main objective in this proposed algorithm is to have an alternative scheme with higher speed, particularly for high density codes. In our presented technique the carry flags are grouped, and the carries within a group are generated concurrently, where the number of carries in each group is equal to those already generated. This process naturally doubles the generation of the carry flags in each step (one gate delay), causing an exponential generation of carries. This process is shown to be very effective in reducing the overall delay in the adder, as expected.

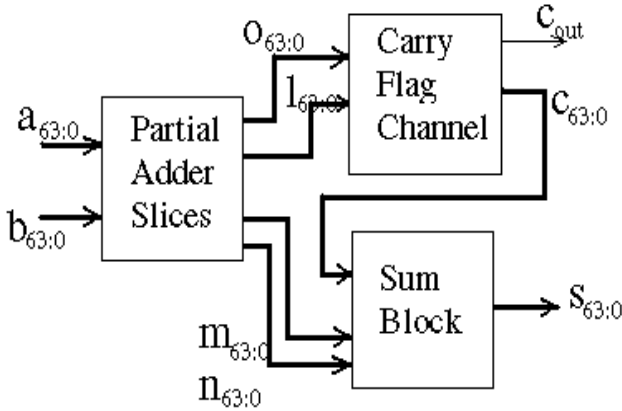


Fig.1 - Block diagram of a 64-bit carry select adder.

A. A 64-bit Adder

Figure 1 shows a block diagram for a 64-bit adder. The adder consists of three circuit blocks: Partial Adder, Carry Flag Channel, and Sum Block. The Partial Adder simply generates single bit partial sums and partial carries for input carries 0 and 1, as given in Eq.(2). The Carry Flag Channel, on the other hand, generates all carry flags, from c_i to c_{63} . Finally, with carry flags available the final sum term, for each bit, is selected within the Sum Block.

B. Carry Generation Block

To understand how the carry flag circuit works we refer to Fig.2. Here the $i+1^{st}$ carry flag, c_{i+1} , is generated through a 2-to-1 MUX that selects between the two primary carry terms O_{i+1} and I_{i+1} by using the i th carry flag c_i . On the other hand, in Fig.3, the $i+2^{nd}$ carry flag c_{i+2} is generated through a MUX-circuit that uses both $i+1^{st}$ and $i+2^{nd}$ primary carry terms (i.e. O_{i+1} , I_{i+1} , O_{i+2} and I_{i+2}) and still uses c_i for the final selection. In fact Fig.3 indicates

that we can simultaneously generate carry flags c_{i+1} and c_{i+2} in exchange for some increase in gate counts. One may conclude that we are then exchanging higher speed for higher gate counts. This is a true statement, but at least we have first achieved to resolve the delay problem in the carry propagation channel. Let us now consider Fig.4. Here the $i+3^{rd}$ carry flag, c_{i+3} , is also generated through an MUX-circuit which is the same circuit shown in Fig.3, except that here we use x_0 and x_1 , generated in the preceding MUX-circuit (Fig.3), as the selecting signals. This indicates that a substantial saving in the hardware could be achieved by sharing partial circuits between the individual carry generation units. Hence, as the number of bits grow the carry delays increase logarithmically while the growth of the carry generation circuit is closer to linear, rather than exponential. There is, of course, a limit to sharing the signals between the carry generation unit. The rule of the game is the *path delay*, as stated below.

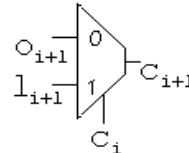


Fig. 2 - Generation of c_{i+1} using c_i , and O_{i+1} and I_{i+1} .

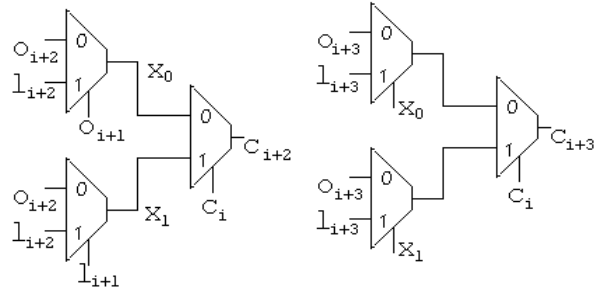


Fig.3-Generation of c_{i+2} using c_i . Fig.4-Generation of c_{i+3} using c_i .

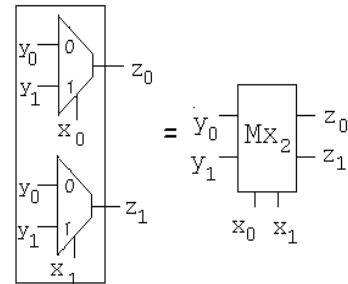


Fig. 5 - Double 2-to-1 MUX module, MX_2

If a carry flag is to be generated with a maximum of n gate delays then any path from a pair of primary carries, O_j and I_j , to the corresponding (generated) carry flag must not exceed n gate delays, as well.

This is an essential and very important statement in designing the carry select channel. In our algorithm we are

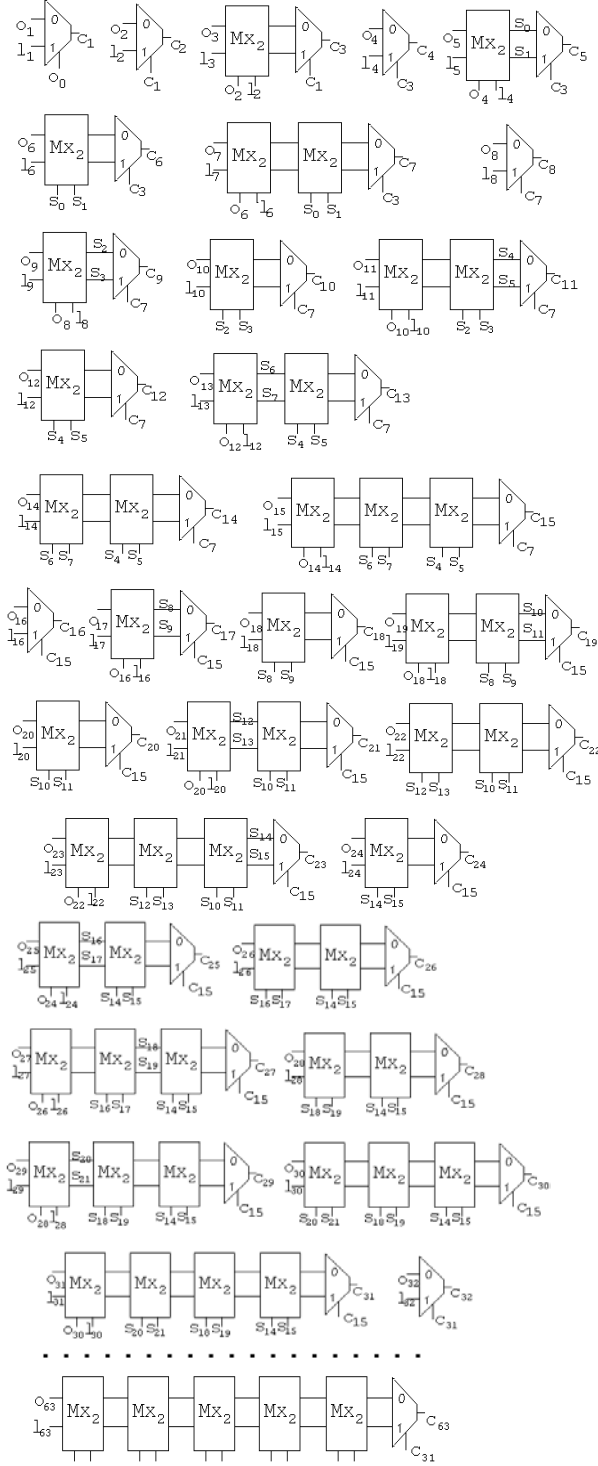


Fig. 6 - Carry generation circuit modules.

proposing a method that has favorably responded to an effective generation of carries, fully implementing the *path delay* rule. For instance, in generating the carry $C_1, C_3, C_7, C_{15}, C_{31},$ or C_{63} all path delays (in gate unit) from the

corresponding primary carries to the generated carry flag are the same and equal to 1, 2, 3, 4, 5, or 6, respectively.

C. Design of Carry Generation Channel

To start designing the carry flag channel, we first introduce a double 2-to-1 MUX module, MX_2 , as shown in Fig.5. This module is a fundamental element used to build the carry-select channel. Multiple pairs of primary carry signals (O_i and I_i) are transferred through multiple MX_2 modules to create each final carry flag, as shown in Fig.6.

Figure 6 symbolically illustrates the carry generation channel for a 64-bit adder. As indicated earlier, the carry flags in each group are generated simultaneously, and the groups are specified in Table 1. Table 1 also specifies the path delays (in scale of a gate delay) of different carry flags generated in the channel.

Table 1

C_1	1 gate delay
C_2 and C_3	2 gates delay
C_4, C_5, \dots and C_7	3 gates delay
C_8, C_9, \dots and C_{15}	4 gates delay
C_{16}, C_{17}, \dots and C_{31}	5 gates delay
C_{32}, C_{33}, \dots and C_{63}	6 gates delay

Table 2 provides the gate counts for the carry flag channel and per carry flag, for different size operands. Note that the growth of the hardware (gate counts) for higher density operands is not much, as specified earlier, and this growth is less than one gate for each doubling of the operand size.

Table 2

Operand Size	Carry generation Channel Gate Counts	Gate Counts Per Carry Flag
2	2	1
4	6	1.5
8	18	2.25
16	50	3.125
32	130	4.0625
64	322	5.03125

D. The Sum Block

The final stage in the addition process is the Summing Block (SB). This block is just another array of 64 2-to-1 MUXes. The inputs to the i th MUX are m_i and n_i (see Eq. (2)), the select signals to the MUX are the carry flags C_{i-1} , and the corresponding outputs are sum bits s_i , for $i = 0, 1, \dots, 63$, and $C_{out} = C_{63}$.

E. Adder Design, Simulation and Implementation

The proposed adder for 64-bit operands is designed, simulated and implemented. The total gate counts for the

core is 642, being split into three blocks: 256 gates for the Partial Adder Slices, 322 of 2-to-1 MUXes for the Carry Flag Channel, and 64 of 2-to-1 MUXes for the Sum Block.

A 32-bit adder is implemented using XC4010 Xilinx FPGA. For 64-bit adder the work is in progress to implement a modified version of the design.

Figure 7 shows the simulation results for a 32-bit adder. For more than 1000 random numbers not a single error (error1) was reported.

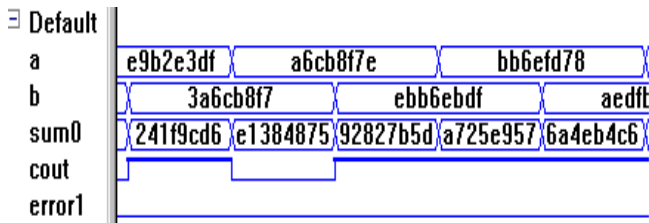


Fig. 7 - Simulation results for a 32-bit adder.

III. MODIFICATIONS AND IMPROVEMENTS

There are possibilities to modify the adder design, and in some cases, improve the performance of the adder. A modification is to resize the slicing for partial adders. For example, for two bit slicing, rather than the single bit, the delay time stays almost unchanged (8 gate delays for a 64-bit adder), but the hardware increase. The change is 480 gates and 194 2-to-1 MUXes bringing to a total of 674, compare to 642 gates in the single bit slicing.

A second modification that is very important is the *load distribution*. As noticed in our carry generation channel, some of the critical carry flags, C_3 , C_7 , C_{15} and C_{31} , are extensively used to generate other carries. For example, C_{15} and C_{31} are used 17 and 33 times, respectively. This loading will definitely reduces the overall speed causing a longer delay. To avoid this some of the non-critical carry flags can be used to generate other non-critical carries, and leave the critical carries lightly loaded. Larger buffers for the critical carries may also help to reduce the delay.

A third modification is to extend the design to cover both addition and subtraction. To cover this option we need to make two modifications. First, we need to allow carry-in, C_{in} , into the adder, such that for the subtraction we could assign C_{in} to 1. The second modification is to use XOR for each bit of operand b in order to be able to invert its bits by assigning $C_{in} = 1$. The procedure increases the delay by another gate delay, and also it increases the gate counts by 64 more XORs, for a 64-bit adder/subtractor.

IV. CONCLUSION

An algorithm and design procedure is introduced with logarithmic carry propagation delay. This development is particularly important for high-density adders. The design

is aimed at the construction of 32-bit and 64-bit adders with maximum path delay equivalent to 7 and 8 gate delays, respectively. The algorithm is based on the carry select technique with operands partitioned into single bit slices for both quick response and low gate counts. Another property observed in this algorithm is resource (hardware) sharing which is due to regularity of the carry channel structure. The design is coded in Verilog, simulated and implemented for 32-bit adder using XC4010E Xilinx FPGA technology.

REFERENCES

- [1] N. R. Scott, *Computer Number Systems and Arithmetic*. Englewood Cliffs, NJ: Prentice-Hall, 1985, pp. 54-57.
- [2] B. Parhami, *Computer Arithmetic, Algorithm and Hardware Design*, Oxford University Press, New York, 2000, pp. 114-123.
- [3] J.O. Bedrij, "Carry-Select Adder," *IRE Trans. Electronic Computers*, Vol. 11, pp. 3403-346, 1962.
- [4] Nigaglioni, R. H., and E. E. Swartzlander, "Variable Spanning Tree Adder," *Proc. Asilomar Conf. Signals, Svstems, and Computers*, 1995, pp. 586-590.
- [5] B. Gilchrist, J. H. Pomerence, and S. Y. Wong, "Fast carry logic for digital computers," *IRE Trans. El. Comp.*, vol. EC-4, no. 4, pp. 133-136, Dec. 1955.
- [6] H. L. Garner, "A survey of some recent contributions to computer arithmetic," *IEEE Trans. Comput.*, vol. C-25, pp. 1277-1282, 1976.
- [7] R. Hashemian, "A New Algorithm to Construct Parallel Adder for High Density Codes," *Proceedings of the 36th Midwest Symposium on Circuits and Systems*, pp. 257-260, August 16-18, 1993.
- [8] R. Hashemian, "An Algorithm and Design Procedure for High Speed Carry Select Adders," *Proceedings of the 37th Midwest Symposium on Circuits and Systems*, pp. 851-854, August 2-5, 1994.
- [9] J. Rose, A.E. Gamal, and A. Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays," *Proc. IEEE*, vol. 81, no. 7, July 1993, pp.1013-1029.
- [10] J. Rose, A.E. Gamal, and A. Sangiovanni Vincentelli, "Architecture of Field-Programmable Gate Arrays," *Proc. IEEE*, vol. 81, no. 7, July 1993, pp.1013-1029.
- [11] N.H.E. Weste and K. Eshraghian, "*PRINCIPLES OF CMOS VLSI DESIGN*", Addison-Wesley Pub. Comp., 1993