SIMPLE AND EFFICIENT CMOS CIRCUIT FOR FAST VLSI ADDER REALIZATION

Vojin G. Oklobdzija

IBM T.J.Watson Research Center
P.O.Box 218
Yorktown Heights, NY 10598
(914) 945-2607

ABSTRACT

In this paper a simple, yet efficient, scheme for a VLSI implementation of addition (ALU) in CMOS is presented. The implementation of this scheme yields an adder (ALU) with near minimal number of gates, small and regular area, yet outperforming carry-lookahead and recurrence solver schemes. This claim was demonstrated by simulation of the actual implementation of examples. Development of this scheme is based on a more realistic estimate of delay in VLSI-CMOS technology and careful selection of circuits [10]. These estimates are different than the ones traditionaly used. Using the scheme developed 32-bit addition in 14.2 nS was realized and based on it a 64-bit adder performing addition in 16.0 nS made possible.

1. INTRODUCTION

In many VLSI processors the adder (ALU) is in the critical path of the machine - therefore determining the machine cycle. Given that the duration of the machine cycle is directly related to the performance of the machine, the speed of the ALU (adder) is critical in achieving higher performance.

The past research effort had the objective of developing the fastest scheme for addition, and as a result many different schemes and their implementations were created [1]. This development was based on the dominant technologies of that period (TTL, ECL).

A study of VLSI oriented schemes was undertaken by Atkins and Ong implying that CLA scheme (Carry Look Ahead) is yielding the fastest implementation. Their conclusion was supported by partially simulated sections in nMOS technology [9].

A similar study of "recurrence solvers" [11],[13,14,15] was done by Han and Carlson [17,18]. They developed a scheme which they refer to as "Hybrid Prefix Computation" (HPC) claiming it to be faster than all previously reported "recurrence-solvers" [11-16] including CLA and the scheme based on variable sized blocks (VBA), "Variable Block Adder" [5,6]. Their claim was also supported by partial simulation using SPICE as it was done in [9].

In the previous work [10] those representative schemes were implemented for the typical sizes of 16 and 32 bits [1][17][18][5], and the speeds obtained were compared. The results did not confirm the conclusions in [9],[17][18]. More elaborate schemes (CLA, HPC) did not show gain in performance due to larger fan-ins and fan-outs. The implementation of VBA scheme was faster than "recurrence solvers" and CLA for a 32-bit implementation and a close second for a 16-bit implementation.

In this paper we describe implementation features of the VBA adder for sizes of 32 and 64 bits. The technology chosen was 1.5 micron CMOS ASIC. The ASIC cell library is coupled to a simulator developed for that purpose [21,22].

2. PERFORMANCE MODEL

For this model, data was derived from a vendor ASIC-CMOS standard cell library [21]. However, the results can be fairly well generalized to the other CMOS technologies. Dependencies discussed here are valid even for custom design. The ASIC library consists of a collection of gates and cells designed for high speed and low gate count. This particular vendor supplies two versions of each cell:

1. a fast version with high gate count using more power (FG)
2. a standard version designed for small gate count (SG)

2.1. Delay Dependency

Commonly used speed estimates take into account the number of logic levels only [1]. However, the speed of CMOS technology shows a dependency on fan-out loading and fan-in. Wiring delays are attributed to the loading of gate outputs and signal propagation delays. The first is expressed in terms of additional fan-out loads and added to the gate output. The second is added to the gate delay.

Fan-Out

CMOS gates exhibit an almost linear dependency on fan-out loading. Fig.1. shows delay of a NAND gate as a function of fan-out load. For a NOR gate, we observe similar increase (increasing the fan-out from 2 to 8 results in approximately tripling the gate delay). An inverter de-

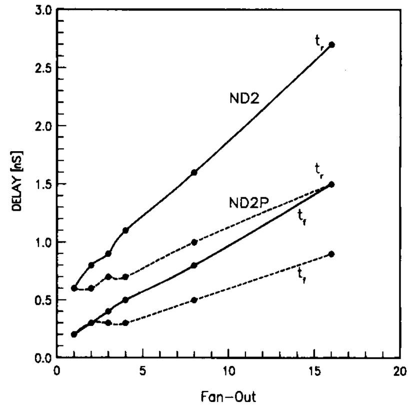

Figure 1. NAND gate delay vs Fan-Out: ND2P is the powered version of a NAND gate: tr - rise time tf - fall time

lay resembles similar dependency on fan-out.

The average delay due to wire capacitance is also linearly dependent on the fan-out and its effect is in increasing linear dependency of gate-delay on the fan-out. The average amount of wiring is proportional to the fan-out and it is expressed in terms of additional fan-out loading, Fig.2. The average wire length (expressed in terms of fan-out) is a function of chip size. Both functions exhibit a linear relationship (for chips of moderate area) and the wiring is reflected as additional fan-out in the gate delay calculation.

The relationship between gate-delay and fan-out can be approximated fairly well by a linear function. Increasing the fan-out results in more delay which is neglected by the delay estimates based on counting levels of logic.

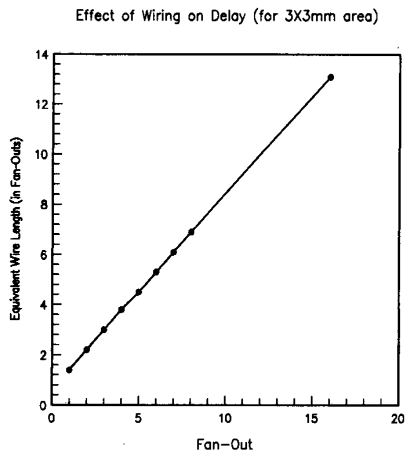

Figure 2. Effect of wire load on delay

Fan-In

Forcing a reduction in the number of logic levels may also result in an increase in fan-in. This increase is reflected in the gate delay since fan-in (FI) is directly proportional (equal) to the number of transistors connected in series in an n-type transistor network (NAND gate) or p-type transistor network (NOR gate). The gate-delay vs fan-in dependency for a NOR gate is shown in Fig.3.

To set the logic state in CMOS, a node must be connected to one of the terminal nodes (Gnd, Vdd) by a string of transistors equal in length to the FI. In case of a NAND gate, these transistors are of n-type (connected to ground node). In case of a NOR gate, these transistors are of p-type (connected to Vdd). Given that a p-type transistor (whose major carriers are holes) is almost twice as slow as an n-type transistor, a NOR gate is slower than an equivalent NAND gate (taking the worse of the two times: t_f , t_r into account) The opposite is true for bipolar technology where a wired-OR implementation is used to obtain a major speed advantage factor. Also, it can be observed that in the case of a NOR gate, the rise time t_r is longer than the fall-time t_f, contrary to the

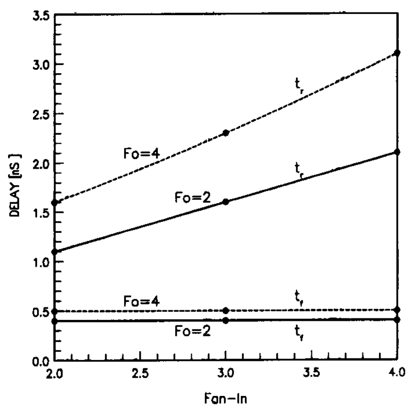

Figure 3. Gate delay as a function of Fan-In: tr - rise time, tf - fall time

NAND gate where the opposite is the case. This is to be expected from the previous observation regarding the transistor paths. Because the time required to change logic states is directly proportional to the number of transistors in the path between the terminal nodes (worst case), which is equal to the FI, the larger the fan-in, the slower the gate.

This simple analysis also challenges the validity of just using the number of logic levels as an adequate estimate of the speed of an implementation as has been traditionally used in development of fast computer arithmetic. However, estimates based on logic levels are still adequate for bipolar technology and other technologies where the driving capabilities of gates are such that the influence of fan-out and wire loading on speed is negligible. With CMOS becoming the dominant VLSI technology, estimates based solely on the number of logic levels are no longer valid. In summary, implementing functions in a minimal number of logic levels does not necessarily yield the fastest implementation.

2.2. Delay Estimates

Simple formulas for estimating speed of a CMOS implementation were derived from the delay tables for the particular ASIC technology [21] and simplified to make them easier to use. All delays are normalized to delay unit of 1 (delay of an inverter) since their purpose has been to provide the relative relationship between the cells. This gives them generality and simplicity for use. The parameters used to estimate the delay δ are: Fo - fan-out of a given gate, (Fi−2) : fan-in in excess of 2.

The formulas for calculating estimates for NAND gate, NOR gate and inverter are:

$$\delta_{NAND} = 1 + 0.3Fo + 0.5(Fi - 2)$$
$$\delta_{NOR} = 1 + 0.5Fo + 0.5(Fi - 2)$$
$$\delta_{INV} = 0.7 + 0.3Fo$$

These estimates have been found adequate for modeling the main contributions to the delay [10]. A NAND gate is slightly less sensitive to fan-out loading than a NOR gate.

For the fast version of the gates (FG) which uses more powerful driver to drive large fan-out load, we similarly obtained:

$$\delta_{NAND-F} = 1 + 0.125Fo + 0.5(Fi - 2)$$
$$\delta_{NOR-F} = 1 + 0.25Fo + 0.5(Fi - 2)$$
$$\delta_{INV-F} = 0.3 + 0.125Fo$$

These relations are used as a base for derivation of a new scheme.

3. IMPLEMENTATION OF THE SCHEME IN CMOS

For implementation CMOS ASIC technology with 1.5 micron line features was selected. Standard cells from the ASIC library [21] are used for design. ASIC library consists of a collection of gates and cells designed for the maximal speed and small gate count.

Fast version FG of the gates was used wherever it was critical to minimize the delay. Otherwise standard version was used for the purpose of minimizing the gate count and power requirements.

Preliminary pass through all the blocks was applied to identify any nodes which are more heavily loaded (larger fan-outs or driving longer wires). At this nodes gates with stronger driving capabilities are used. Also in every block the combination of gates (NAND-NAND, NAND-NOR, etc.) which yields the fastest implementation of a given function was applied. This was estimated from the gate-delay vs fan-out load tables [21] for each gate. After the network compilation and first simulation the nodes with the rise/fall times longer than 3.0 nS were flagged [22]. At this point the fast version gates (FG) were introduced at the flagged nodes or the nodes were divided into separately driven logic.

In the final pass, the critical path was reexamined replacing the gates with their faster version, FG, where applicable and reducing the loading. Several iterations through the design and simulator brought this process to the point where no further improvements were possible.

The experience gained from nMOS technology showed that the pass-transistor circuit can be used to gain advantages in density, speed or both. Several new circuits were invented that take advantage of pass-transistor in nMOS technology [2]. As far as adder circuits are concerned, Manchester Carry Chain [2,3,5] built from pass-transistors yield an implementation faster than its gate oriented counterparts [11].

To take advantage of this design style in CMOS we identified the component containing pass-transistor structure from the cell library [21]. Incidentally, this component, a two-to one multiplexer, is the fastest component in the library.

To be able to use multiplexer in the carry chain the "carry propagate" term p_i must be implemented as an XOR function, $p_i = a_i\, XOR\, b_i$, so that the table in Fig.4. is obtained:

g_i $a_i b_i$	p_i a_i XOR b_i	Select	Cout
0	0	g_i	0
0	1	Cin	Cin
1	0	g_i	1
-	-	-	-

Figure 4. Truth table for p and g terms

The fourth term of the table $p_i = 1$, $g_i = 1$ can never occur. Such choice of p_i function allows for the multiplexer to be used in the carry path as shown in Figure 4. The multiplexer MUX21 incorporates an inverter which serves the amplification function of the carry signal. The carry signal is therefore inverted in the alternative stages of each "bit-slice" of the adder. For consistency the "carry generate" function, $g_i = a_i b_i$, as well as the resulting sum s_i has to be inverted.

Next we applied our scheme, based on the variable sizes of the carry blocks, which are obtained by the process of optimization of the speed of the carry path. We refer to it as VBA (Variable Block Addition) [5-8],[10]. By simulation we obtained that the time required to skip over a block is twice as long as the time required for a carry signal to ripple across a bit in the carry chain. These times are T=1.6nS and t=0.8nS respectively. The division of the 32-bit carry chain into groups of bits of sizes 1,3,5,7,7,5,3,1, respectively is optimal [10].

The implementation of our scheme is shown on the example of an 32-bit adder (easily modified to an ALU). The size of the blocks vary with the size of the adder [5-8],[10]. The critical path is the carry signal path and it is implemented as a string of multiplexers, Fig.5.

Single carry block is implemented using 4 to 1 multiplexer (actually used as 3 to 1) as the last of the string of 2 to 1 multiplexers. Carry by-pass is connected to the inputs 3,4 of the 4:1 multiplexer (group carry multiplexer) and the selection of carry by-pass is activated by the NAND gate signaling the condition for "group propagate" and activating the group multiplexer in turn. This is shown in Fig.6.

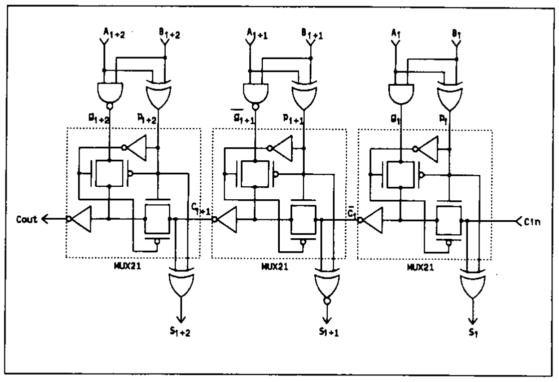

Figure 5. Carry-path implementation with the multiplexer

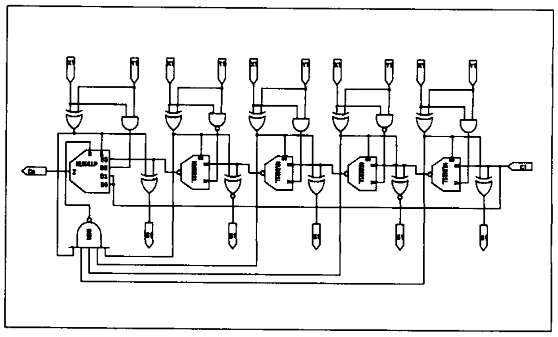

Figure 6. Carry-block of the VBA implementation

This design was simulated for a number of input combinations (test cases) each involving a potential critical path. The simulator used LDS-III [22] takes into the calculation loading caused by the wire of the average length, input transistors, intrinsic gate delay and rise/fall time. The results are very accurate which has been confirmed by experience in a very large number of cases.

The worst case for 32-bit VBA implementation found is t_{cr} = 14.2 nS.

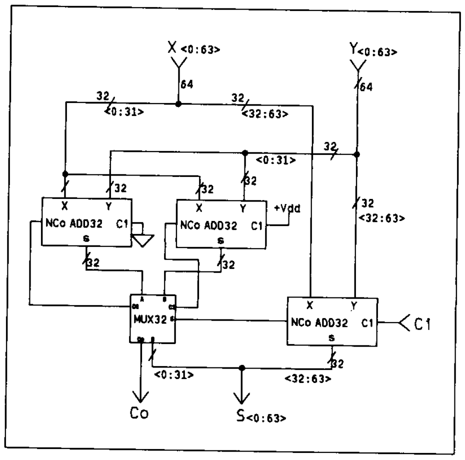

Figure 7. 64-bit Adder Combining CSA and VBA

In our 64-bit implementation we combined VBA and Conditional-Sum Addition scheme (CSA) [12]. The feature of this scheme is the use of duplication in the most significant portion of the adder. Therefore it is very important that the basic block, which is to be replicated, is made simple and yet very fast. The VBA scheme offers this feature. The 64-bit adder is divided in two 32-bit portions, Fig.7. The carry out of the least significant 32-bit portion controls selection of the proper 32-bit sum of the most-significant 32-bits of the adder (bits 32-63). Again, the advantage of having a fast multiplexer as a building block was used in the selection process.

With this implementation 16.0nS 64-bit addition was achieved which is only a small increase compared to the speed of 32-bit scheme. The complexity of the 64-bit adder was only about 60% higher than doubling the 32-bit adder which would result in an adder about twice as slow as 32-bit adder. This 64-bit adder is designed to be used in a multiplier and its addition time is sufficiently fast to allow its operation in 20nS cycle time.

4. COMPARISON WITH OTHER SCHEMES

In [10] we measure the speed of four representative adder implementations. The results reported were obtained by running the entire designs through a very accurate simulator [22].

The schemes implemented were the following: Ripple Carry (RCA), Carry Lookahead (CLA), Recurrence Solver (HPA) and Variable Block Carry (VBA).

All of the examples were implemented for two sizes, 16 and 32 bits.

The respective speed comparison obtained for the selected schemes is shown in Fig.8.

The complexity of each design is measured as a "gate-count", which is a number of equivalent two-input gates (one gate consists of 4 transistors). From Fig.8. we can observe that the implementation of the VBA scheme yields the fastest adder for the size of 32-bits and it is second best in terms of complexity measured in the number of gates used.

This result guided the choice of 32-bit VBA to be implemented in a "Conditional Sum Addition" - CSA, scheme, which takes advantage of being sufficiently fast and yet not very complex in order to justify duplication of the most-significant portion of the 64-bit adder as required by the CSA scheme [12]. This implementation of 64-bit adder uses 1205 equivalent gates and performs addition in 16.0 nS nominal time.

Delay	Simulated		Complexity (no. of equiv. gates)	
Adder type/size	16	32	16	32
RPL	15.7	29.3	160	320
CLA	12.3	15.0	199	443
HPC	9.7	14.4	184	458
VBA	10.8	14.2	174	348

Figure 8. Speed and Complexity of Selected Adders

5. CONCLUSION

In order to achieve a sufficiently fast adder we used the rules modified to reflect the realities of the CMOS technology. The rules (e.g. number of "logic levels") used for the development of various computer arithmetic schemes and their hardware implementations are not very adequate when applied to the new VLSI technologies (such as CMOS).

Therefore it would be a mistake to simply map the designs of these schemes directly into their VLSI-chip implementations.

In this paper it has been demonstrated that by a careful selection of fast cell libraries and adherence to the a more adequate model a simple yet sufficiently fast adder can be designed. The examples given are 32-bit adder with the complexity of 348 equivalent gates performing addition in 14.2 nS nominal time and a 64-bit adder combining VBA and CSA schemes implemented with 1205 equivalent gates and adding in 16.0 nS nominal time.

REFERENCES

[1] Kai Hwang , *Computer Arithmetic: Principles Architecture and Design*, John Wiley and Sons, 1979.

[2] C.Mead, L.Conway, *Introduction to VLSI Systems*, Addison Wesley, 1980.

[3] T.Kilburn, D.B.G.Edwards and D.Aspinall, *Parallel Addition in Digital Computers: A New Fast "Carry" Circuit*, Proc. IEE, Vol.106, Pt.B. P.464, September 1959.

[4] M.Lehman and N.Burla, *Skip Techniques for High-Speed Carry-Propagation in Binary Arithmetic Units*, IRE Transactions on Electronic Computers, December 1961, p.691.

[5] V.G.Oklobdzija, E.R.Barnes, *Some Optimal Schemes for ALU Implementation in VLSI Technology* , Proceedings of 7th Symposium on Computer Arithmetic, June 4-6, 1985, University of Illinois, Urbana, Illinois.

[6] E.R. Barnes, V.G. Oklobdzija, *Method for Fast Carry-Propagation for VLSI Adders* , submitted: March 22, 1983, IBM Technical Disclosure Bulletin, Vol. 28, No. 4, September 1985.

[7] E.R. Barnes, V.G. Oklobdzija, *New Scheme for VLSI Implementation of Fast ALU* , submitted: January 24, 1984 , IBM Technical Disclosure Bulletin, Vol. 28, No. 3, August 1985.

[8] E.R. Barnes, V.G. Oklobdzija, *New Multilevel Scheme for Fast Carry-Skip Addition* , submitted: March 22, 1984 , IBM Technical Disclosure Bulletin, Vol. 27, No.11, April 1985.

[9] S.Ong and D.E.Atkins, *A Comparison of ALU Structures for VLSI Technology*, Proc. of the 6th Symposium on Computer Arithmetic, June 20-22, 1983, Aarhus University, Aarhus, Denmark.

[10] V.G.Oklobdzija, E.R.Barnes, *On Implementing Addition in VLSI Technology*, IEEE Jo. of Distributed and Parallel Computing, to be published in the Special Issue on Computer Arithmetic, 1988.

[11" M.Pomper et al, *A 32-Bit Execution Unit in an Advanced NMOS Technology*, IEEE Journal of Solid State Circuits, Vol. SC-17, No.3, June 1982.

[12] A.Bilgory and D.D.Gajski, *Automatic Generation of Cells for Recurrence Structures*, Proc. of 18th Design Automation Conference, Nashville, Tennessee 1981.

[13] A.Bilgory and D.D.Gajski, *A Heuristic for Suffix Solutions* , IEEE Transactions on Computers, Vol. C-35, No.1, January 1986.

[14] R.P.Brent, H.T.Kung, *A Regular Layout for Parallel Adders* , IEEE Transactions on Computers, Vol. C-31, No.3, March 1982.

[15] P.M.Kogge, H.S.Stone, *A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations* , IEEE Transactions on Computers, Vol. C-22, No.8, August 1973.

[16] T.F.Ngai, M.J.Irwin, *Regular, Area-Time Efficient Carry-Lookahead Adders* , Proceedings of 7th Symposium on Computer Arithmetic, June 4-6, 1985, University of Illinois, Urbana, Illinois.

[17] T.Han, D.A.Carlson, *Fast Area-Efficient VLSI Adders* , Proceedings of 8th Symposium on Computer Arithmetic, May 19-21, 1987, Como, Italy.

[18] T.Han, D.A.Carlson, S.P.Levitan, *VLSI Design of High-Speed Low-Area Addition Circuitry* , submitted to ICCD'87, October 5-8, 1987, Rye, New York.

[19] T.Han, *Theory of Hybrid Prefix Algorithm* , PhD proposal, February, 1987.

[20] T.Han, *Private Communications* , February, 1987.

[21] *Compacted Array Product Databook*, LSI Logic Corp., February, 1987.

[22] *LDS Software Guidelines*, LSI Logic Corp., 1985, 1986.