HIGH-SPEED ADDITION IN CMOS

Nhon T. Quach
Michael J. Flynn

Technical Report: CSL-TR-90-415

FEBRUARY 1990

This work was supported by NSF contract No. MIP88-22961.

HIGH-SPEED ADDITION IN CMOS

by

Nhon T. Quach and Michael J. Flynn

Technical Report CSL-TR-90-415

February 1990

Computer Systems Laboratory

Departments of Electrical Engineering and Computer Science

Stanford University

Stanford, California 94305-4055

Abstract

This paper describes a fully static Complementary Metal-Oxide Semiconductor (CMOS) implementation of a Ling type adder. The implementation described herein saves up to one gate delay and always reduces the number of serial transistors in the worst-case (critical) path over the conventional carry look-ahead (CLA) approach with a negligible increase in hardware.

Key Words and Phrases: Carry look-ahead, conditional-sum adders, modified Ling addition, group-generate, high-speed CMOS binary adder, multiple output Domino logic.

1 Introduction

For high-speed addition, Ling type adders [1,2] have been demonstrated to have advantages over conventional CLA adders in emitter-coupled logic (ECL) [3]. Ling's approach results in a drastic load reduction in the input stage circuitry, thereby allowing direct generation of the *group*-generate from input operands. Because this approach takes advantage of the dot-or capability of ECL, it is not as suitable for CMOS adders. A straightforward application of Ling's scheme to CMOS adders can lead to an increase in hardware or delay time, or both. Also, in fully static CMOS (as opposed to ECL), the designers have to worry about both the N-channel and the P-channel transistor networks. Optimizing on the N-channel transistor network alone is not sufficient.

In this paper, we present an implementation of a fully static CMOS adder using a modified Ling scheme. Our implementation saves up to one gate delay and always reduces the number of serial transistors in the critical path over the conventional CLA approach with *a* negligible increase in hardware. In CMOS, because the speed of a gate is primarily limited by the number of serial transistors connecting the output node to the power or the ground nodes, reducing the number of serial transistors in the critical path, therefore, speeds up the adder.

In section 2, we show, by way of a design, how Ling's approach can be modified for CMOS adders. In section 3, we compare the present adder with other adders reported in the literature. Section 4 contains a summary. In this paper, ***AND*** is denoted by juxtaposition, ***OR*** by $\vee$, ***EXCLUSIVE-OR*** by $\oplus$, negation by overbar, and $\prod_{i=1}^{n} p_i$ by p_{1-n}. Index $i \in (0, 32)$ is used for bits (carry-in is treated as g_0 and actual sums range from 1 to 32), index $j \in (0, 10)$ for groups, and index $k \in (0, 2)$ for blocks, exclusively and respectively.

2 The Adder

The adder is divided into four blocks, of sizes 9-, 9-, 9-, and 6-bits, respectively. Since carry-in is treated as g_0, block 0 actually has only eight bits. Each block is subdivided into three 3-bit groups, except for the last 6-bit block, which has only two 3-bit groups. Within each group and each block, the local sum logic uses the conditional-sum algorithm. Figure 1 shows the structure and numbering convention of the adder and Figure 2 depicts the global carry propagation process. In [2] and [3], the adders have 4-bit groups; owing to the limited fan-in capability of fully static CMOS circuits, 3-bit groups are used in the present adder (Figure 1).

Some definitions:

$$g_i \equiv a_i b_i$$

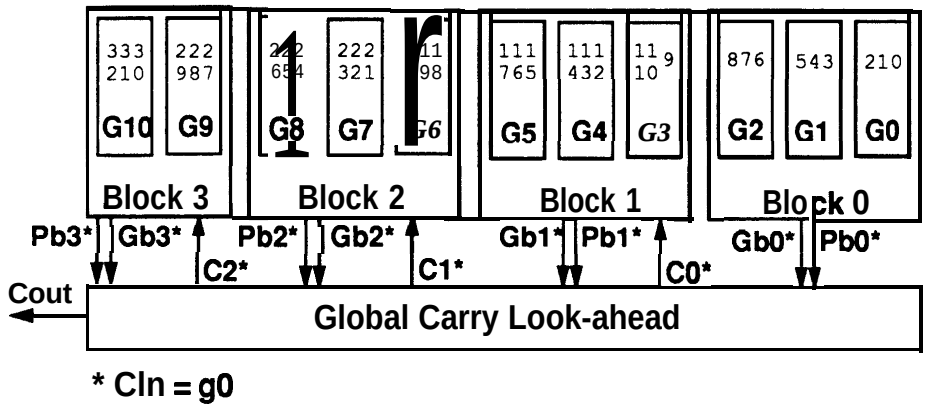

Figure 1: Structure and Naming Convention of Present CMOS Adder.

$$P_i \equiv a_i \vee b_i$$

$$g_0 \equiv c_{in}$$

$$s_i \equiv a_i \oplus b_i$$

so that

$$S_i = s_i \oplus c_{i-1}$$

In the definitions, a_i and b_i are the i^{th} bits of the input operands. s_i and S_i are the i^{th} bits of the local and the final sum, respectively.

Global-Carry

For ease of discussion, we illustrate our approach on one group in a block (Figure 1) of the adder (say, group 8 in block 2). The conventional ***group-generate*** equation for the group is [4]

$$G_8 = g_{26} \vee g_{25}p_{26} \vee g_{24}p_{25}p_{26} \quad (1)$$

Using the identity $g_i = p_i g_i$ and extracting p_{26}, we rewrite equation (1) as

$$\begin{aligned} G_8 &= p_{26}(g_{26} \vee g_{25} \vee g_{24}p_{25}) \\ &= p_{26}G_8^* \end{aligned}$$

where

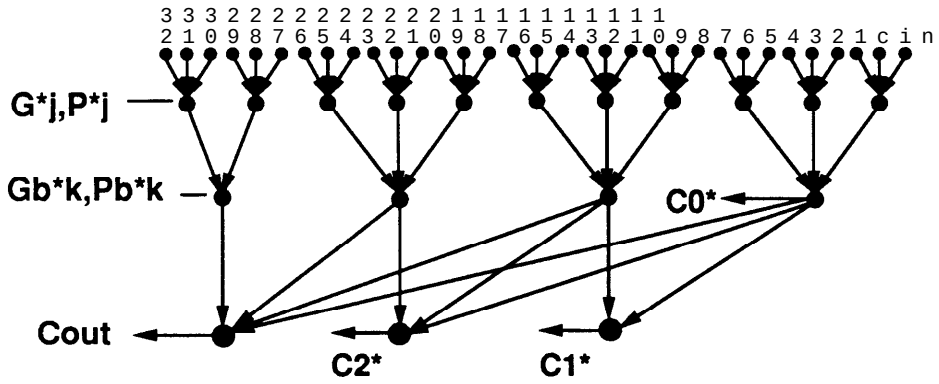

Figure 2: Global Carry Generation Process.

$$G_8^* = g_{26} \vee g_{25} \vee g_{24}p_{25} \tag{2}$$

The essence of Ling's approach is to propagate the G_8^* term only. To see the advantage of his approach, G_8^* can be expanded as

$$\begin{aligned} G_8^* &= a_{26}b_{26} \vee a_{25}b_{25} \vee a_{24}b_{24}(a_{25} \vee b_{25}) \\ &= a_{26}b_{26} \vee a_{25}b_{25} \vee a_{24}b_{24}a_{25} \vee a_{24}b_{24}b_{25} \end{aligned} \tag{3}$$

Equation (3) contains four terms and a total of ten literals, with the largest term having three literals. This can be implemented in CMOS in one complex gate [5]. The conventional ***group-generate*** equation (1), when expanded, contains seven terms and a total of 24 literals, with the largest term containing four literals. In CMOS, the number of literals in a term in a logic equation corresponds to the number of N-channel serial transistors; equation (2) is therefore preferable to equation (1) for direct generation of ***group-generate*** from input operands. Generating ***group-generate*** saves one gate delay in the critical path because the g_i and p_i terms are not implemented.

Readers experienced in CMOS circuit design may have realized that the implementation of equation (3) in fully static CMOS has four P-channel transistors in series, severely limiting its usefulness (Figure 3). A more careful examination of equation (2) suggests an alternative as the relationship of p_i and g_i can again be put to use ($\overline{g}_i\overline{p}_i = \overline{p}_i$). The P-channel transistor network implements the dual of equation (2), which is

$$\overline{G_8^*} = \overline{g}_{26}\overline{g}_{25}(\overline{g}_{24} \vee \overline{p}_{25})$$

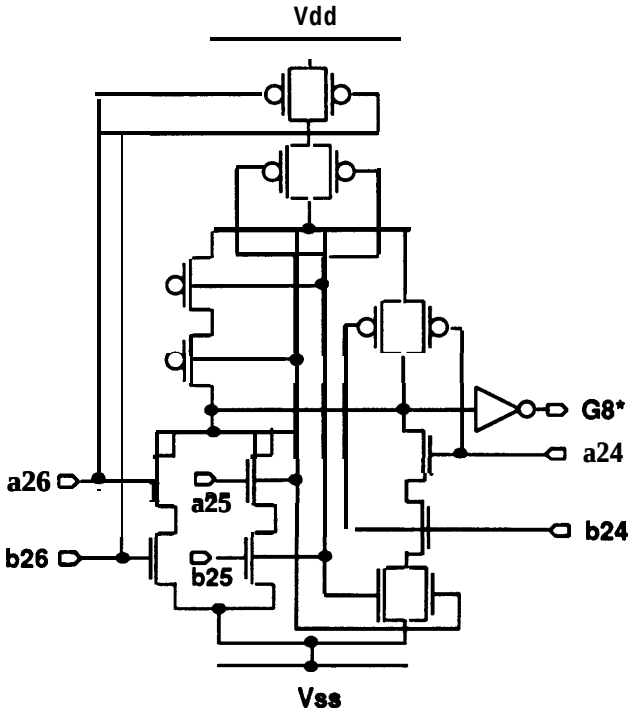

Figure 3: Typical CMOS Implementation of Group-Generate.

$$= \bar{g}_{26}(\bar{g}_{25}\bar{g}_{24} \vee \bar{p}_{25})$$
$$= (\bar{a}_{26} \vee \bar{b}_{26})[(\bar{a}_{25} \vee \bar{b}_{25})(\bar{a}_{24} \vee \bar{b}_{24}) \vee \bar{a}_{25}\bar{b}_{25}] \quad (4)$$

Figure 4 shows an improved implementation of equation (3) using equation (4); only three P-channel transistors are in series.

The equation for ***group-propagate*** in conventional CLA is

$$P_8 = p_{24-26}$$

Ling uses a modified ***group-propagate,*** P_8^*, defined as

$$P_8^* = p_{23-25}$$

Note that G_j^* and P_j^* are, respectively, the reduced and left-shifted versions of the conventional G_j and P_j. **To see the advantage of defining *group-propagate*** this way, consider the ***block-generate*** equation for block 2 (Figure 1)

$$Gb_2 = G_8 \vee G_7P_8 \vee G_6\bar{P}_7\bar{P}_8$$
$$= G_8^*p_{26} \vee G_7^*p_{23}P_8 \vee G_6^*p_{20}\bar{P}_7\bar{P}_8 \quad (5)$$

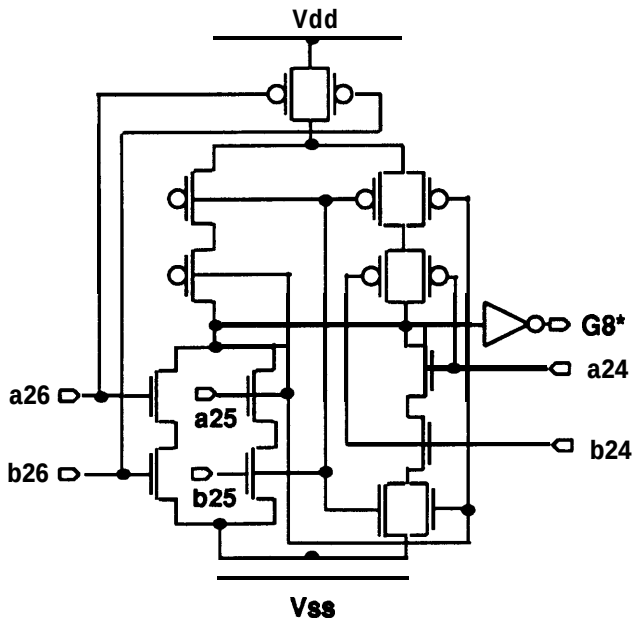

Figure 4: Improved CMOS Implementation of Group-Generate.

Using the definition of P_j^*, we rewrite equation (5) as

$$\begin{aligned} Gb_2 &= G_8^* p_{26} \vee G_7^* p_{23} p_{24-26} \vee G_6^* p_{20} p_{21-23} p_{24-26} \\ &= p_{26}(G_8^* \vee G_7^* p_{23-25} \vee G_6^* p_{20-22} p_{23-25}) \\ &= p_{26}(G_8^* \vee G_7^* P_8^* \vee G_6^* P_7^* P_8^*) \\ &= p_{26} Gb_2^* \end{aligned} \tag{6}$$

Hence, the use of P_j^* affords a more efficient implementation of ***block-generate;*** equation (6) is easier to implement than equation (5) because p_{26} in equation (6) propagates further down into the ***final-carry*** equation:

$$\begin{aligned} C_2 &= Gb_2 \vee Gb_1 Pb_2 \vee Gb_0 Pb_1 Pb_2 \\ &= Gb_2^* p_{26} \vee Gb_1^* p_{17} p_{18-26} \vee Gb_0^* p_8 p_{9-26} \\ &= p_{26}(Gb_2^* \vee Gb_1^* p_{17-19} p_{20-22} p_{23-25} \vee Gb_0^* p_{8-10} p_{11-13} p_{14-16} p_{17-19} p_{20-22} p_{23-25}) \\ &= p_{26}(Gb_2^* \vee Gb_1^* P_6^* P_7^* P_8^* \vee Gb_0^* P_3^* P_4^* P_5^* P_6^* P_7^* P_8^*) \\ &= p_{26}(Gb_2^* \vee Gb_1^* Pb_2^* \vee Gb_0^* Pb_1^* Pb_2^*) \\ &= p_{26} C_2^* \end{aligned} \tag{7}$$

where

$$C_2^* = Gb_2^* \vee Gb_1^* Pb_2^* \vee Gb_0^* Pb_1^* Pb_2^*) \tag{8}$$

The ***final-carry*** equations C_0, C_1, and C_{out} can be derived similarly as

$$C_0 = p_8 Gb_0^* \tag{9}$$

$$\begin{aligned} C_1 &= p_{17}(Gb_1^* \vee Pb_1^* Gb_0^*) \\ &= p_{17} C_1^* \end{aligned} \tag{10}$$

$$\begin{aligned} C_{out} &= p_{32}(Gb_3^* \vee Gb_2^* Pb_3^* \vee Gb_1^* Pb_2^* Pb_3^* \vee Gb_0^* Pb_1^* Pb_2^* Pb_3^*) \\ &= p_{32} C_{out}^* \end{aligned} \tag{11}$$

The definitions of Gb_k^* and Pb_k^* in equations (7), (9), (10), and (11) follow those of the conventional Gb_k and Pb_k with a simple modification: the P_j and G_j terms are replaced by the P_j^* and G_j^* terms, respectively. For example,

$$Gb_1 = G_5 \vee G_4 P_5 \vee G_3 P_4 P_5$$

$$Gb_1^* = G_5^* \vee G_4^* P_5^* \vee G_3^* P_4^* P_5^* \tag{12}$$

and

$$Pb_1 = P_{3-5}$$

$$Pb_1^* = P_{3-5}^*$$

Ling's scheme calls for either the implementation of C_k from Pb_k^* and Gb_k^* (equations (7), (9), (10), and (11)) or the modification of local sum logic to account for the fact that C_j^* is propagated [2]. Neither options are attractive. The former fails to reduce the number of serial transistors in the critical path and the latter adds complexity to the local sum logic, increasing hardware and delay time.

In the present implementation, only the C_k^*, Pb_k^*, Gb_k^*, P_j^*, and G_j^* terms are implemented in the carry look-ahead circuitry without modifying the local sum logic. The p terms in equations (7), (9), (10), and (11) are implemented in the local ***group-carry*** equations, both of which are non-critical paths. In the following section, we show that this is indeed possible and in fact desirable because of reuse of the P_j^* and G_j^* terms. The ability to reuse the P_j^* and G_j^* terms is one of the salient features of the present adder.

Local Group-Carry

We can prove for the general case that the ***p*** terms in the ***final-carry*** equations (7), (9), (10), and (11) can be implemented in the local ***group-carry*** equations for all blocks in the adder. For ease of discussion, however, we show that this is possible for block 2. Equations for other blocks in the adder can be derived in a similar fashion.

The ***final-sum*** equation for bit 24 in group 8 is (see Figure 1)

$$\begin{aligned} S_{24} &= s_{24} \oplus (G_7 \vee P_7 G_6 \vee P_7 P_6 C_1) \\ &= s_{24} \oplus (G_7^* p_{23} \vee p_{21-23} p_{20} G_6^* \vee p_{21-23} p_{18-20} p_{17} C_1^*) \\ &= s_{24} \oplus [p_{23}(G_7^* \vee p_{20-22} G_6^* \vee p_{20-22} p_{17-19} C_1^*)] \\ &= s_{24} \oplus [p_{23}(G_7^* \vee P_7^* G_6^* \vee P_7^* P_6^* C_1^*)] \\ &= s_{24} \oplus \{p_{23}[G_7^* \vee P_7^*(G_6^* \vee P_6^* C_1^*)]\} \end{aligned} \tag{13}$$

Defining

$$gb_7 \equiv p_{23}(G_7^* \vee P_7^* G_6^*) \tag{14}$$

and

$$pb_7 \equiv p_{23}[G_7^* \vee P_7^*(G_6^* \vee P_6^*)] \tag{15}$$

and expanding equation (13) in terms of C_1^* using Shannon's theorem [6], we get

$$S_{24} = \overline{C_1^*}(s_{24} \oplus gb_7) \vee C_1^*(s_{24} \oplus pb_7)$$

or

$$S_{24} = \overline{C_1^*}(\overline{gb}_7 s_{24} \vee gb_7 \overline{s}_{24}) \vee C_1^*(\overline{pb}_7 s_{24} \vee pb_7 \overline{s}_{24})$$

Equations for S_{25} and S_{26} can be derived similarly and are given below:

$$S_{25} = \overline{C_1^*}[\overline{gb}_7(s_{25} \oplus g_{24}) \vee gb_7(s_{25} \oplus p_{24})] \vee C_1^*[\overline{pb}_7(s_{25} \oplus g_{24}) \vee pb_7(s_{25} \oplus p_{24})]$$

and

$$\begin{aligned} S_{26} = {} & \overline{C_1^*}\{\overline{gb}_7[s_{26} \oplus (g_{25} \vee p_{25} g_{24})] \vee gb_7[s_{26} \oplus (g_{25} \vee p_{25} p_{24})]\} \vee \\ & C_1^*\{\overline{pb}_7[s_{26} \oplus (g_{25} \vee p_{25} g_{24})] \vee pb_7[s_{26} \oplus (g_{25} \vee p_{25} p_{24})]\} \end{aligned} \tag{16}$$

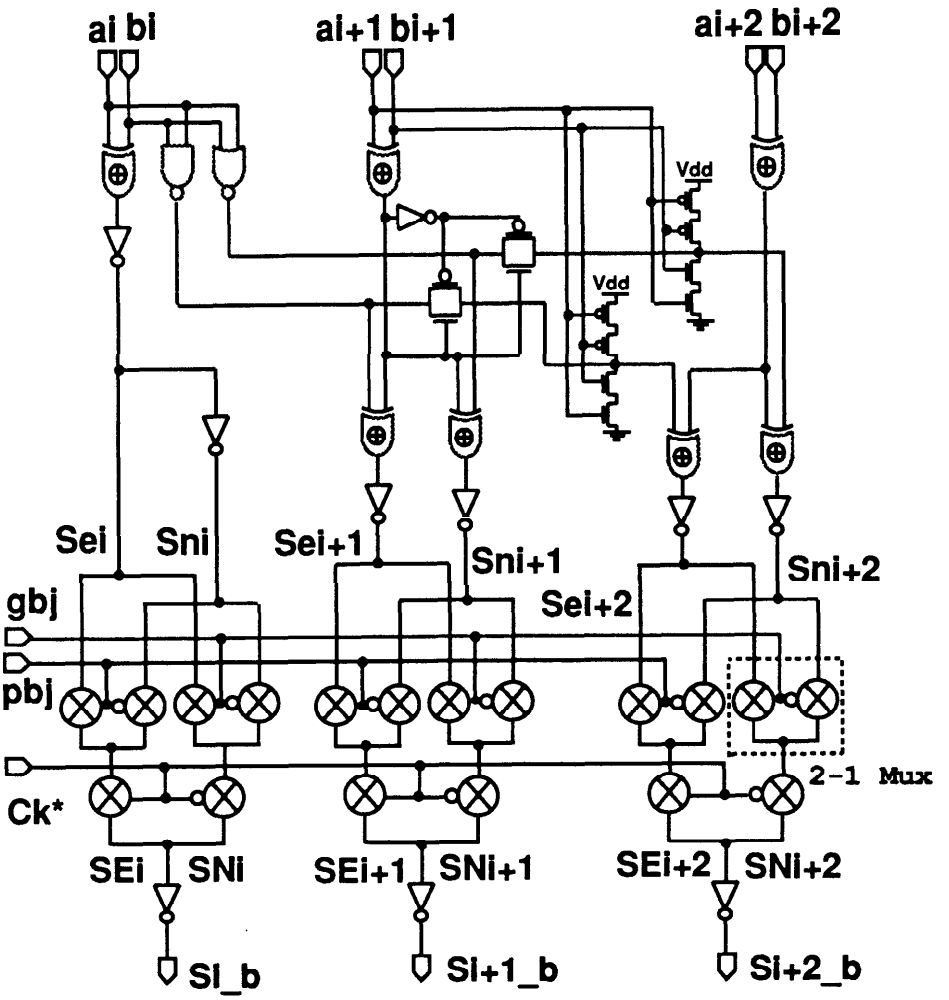

Figure 5: Logic Diagram of Group 8.

Hence, only C_1^* in equation (10) needs to be propagated globally to block 2. p_{17} can be accounted for locally in equations (14) and (15). Both equations (14) and (15) can be implemented in two complex gate delays since P_j^* and G_j^* are available in one complex gate delay. In the present implementation, all complex gates have at most 3 serial transistors and are roughly of the same complexity as that shown in Figure 4. The only exception is the complex gate in the non-critical path used to generate C_{out}, which has 4 serial transistors (Figure 6). One complex gate delay is roughly equal to two and a half 2-input ***NAND*** gate delays. The $G_6^* \vee P_6^*$ term in equation (15) is actually available from group 7 within the same block and can be reused at a cost of one (complex) gate delay, increasing the number of (complex) gate delays of pb_7 from two to three. Figure 5 shows an implementation of group 8.

In Figure 5, it is interesting to note that we have used $s_{25} (= a_{25} \oplus b_{25})$ as the local propagate

(globally, we used $p_i(=a_i \vee b_i)$), allowing a more efficient implementation of $(g_{25} \vee p_{25}g_{24})$ and $(g_{25} \vee p_{25}p_{24})$ in equation (16). Because the present approach does not modify the local sum logic, there is no increase in hardware in Figure 5 when compared with an implementation that uses the conventional conditional-sum (CSA) algorithm.'

In terms of number of (complex) gate delays, S_{32} is no worse than S_{26}. The equation for S_{32} is similar to equation (16):

$$S_{32} = \overline{C}_2^*\{\overline{gb}_9[\overline{g}_{30}(s_{32} \oplus g_{31}) \vee g_{30}(s_{32} \oplus p_{31})] \vee gb_9[\overline{g}_{30}(s_{32} \oplus g_{31}) \vee g_{30}(s_{32} \oplus p_{31})]\} \vee C_2^*\{\overline{pb}_9[\overline{p}_{30}(s_{32} \oplus g_{31}) \vee p_{30}(s_{32} \oplus p_{31})] \vee pb_9[\overline{p}_{30}(s_{32} \oplus g_{31}) \vee p_{30}(s_{32} \oplus p_{31})]\} \quad (17)$$

where

$$gb_9 = p_{29}G_9^*$$

and

$$pb_9 = p_{29}(G_9^* \vee P_9^*)$$

From the previous discussion, P_j^* and G_j^* are available in one complex gate delay, Pb_k^* and Gb_k^* in two, and C_k^* in three. The final sum selection multiplexor is counted as one gate delay. Hence, the present adder has a total of four complex gate delays (three complex gates and *a* multiplexor).

3 Comparison with Other Adders

Table 1 compares the present adder with the conventional CLA, conditional-sum, carry-select, and the Multiple-Output Domino Logic (MODL) adders [7] in terms of complex gate delays and number of serial transistors in the critical path. The present adder has fewer complex gate delays than other adders. In the comparison, we have assumed that the CLA adder is implemented in a complex gate oriented media (i.e., MOS LSI or VLSI) and the carry-select adder uses 4-bit groups and conventional carry look-ahead to propagate the global carry. To be fair, we have further assumed that the conditional-sum adder has a similar organization as the present adder but without using the modified Ling approach as our adder did and that the MODL adder uses conditional-sum logic locally.

'Since we used CSA for the local sum logic, it is fair to compare our adder with CSA, knowing that CSA consumes more hardware than CLA [4]. Also, by using $s_i(=a_i \oplus b_i)$ as the local propagate, the present implementation actually saves hardware.

Table 1: Comparison of CMOS Adders in Terms of Gate Delays and Number of Serial Transistors in Critical Path.

Adders	Gate Delay* from c_{in} to S_{32}	Number of Serial Transistors from c_{in} to S_{32}
CLA	6	21
Condition-Sum Adder	5	16
Carry-Select Adder	5	18
MODL Adder	5	18
Present Adder	4	14

* Complex gate delay. As explained in the text, this is not *a* good measure of adder speed. Number of serial transistors is a better measure.

Because CLA, conditional-sum adder, and carry-select adder do not generate ***group-generute*** directly, they have one more gate delay than the present adder. CLA requires another gate delay to generate the local sum, increasing its total number of gate delays from 5 to 6. The MODL adder though generates ***group-generate*** directly, it does so by using a small 2-bit group [7], requiring more levels in the global carry generation process than the present adder.

Comparison of complex gate delays in CMOS adders can be misleading because they depend on both fan-in and fan-out. A better measure is the number of serial transistors which a signal must traverse in the critical path. This means that for fully static CMOS circuits, we evaluate both P-channel and N-channel transistors for critical paths. For dynamic CMOS circuits [8], which include DOMINO circuits, we only evaluate the N-channel transistors. Hence, this comparison scheme is slightly biased against fully static CMOS circuits.

Admittedly, comparing CMOS adders in terms of serial transistors in the critical path is crude but it does allow us a quick way to evaluate the potential performance of an algorithm for further study. A fair comparison scheme should consider area, power consumption, speed, and design turnaround time; such an elaborate scheme is far more time consuming than can be afforded during the algorithm selection phase of a study.

In counting the number of transistors, the discharge N-channel transistors in DOMINO logic are not included. Inverters are counted as one transistor and ***XOR*** gates as two. The number of serial transistor count for ***NAND, NOR,*** and complex gates is the number of transistors in

the longest N-channel or P-channel chain for static and dynamic CMOS circuits. When there are pass gates [8] involved, the situation is a little more complicated. The source-drain (input-output) path of a pass gate is counted as 0.5 transistors and the gate-drain (control-output) path as one transistor. By the same token, 2-l multiplexors are counted as two transistors from the selection-output path, but as 0.5 from the input-output path. A similar comparison scheme has been suggested by Oklobdzija and Barnes [9], but the accounting details were not given.

From Table 1, the present adder has fewer number of serial transistors in the critical path than others. To count the number of serial transistors in the critical path of the present adder, equation (15) can be used. The input signal must traverse G_j^* in four transistors (Figure 4), Gb_k^* in another four transistors (equation (12) plus an inverting buffer), and C_2^* in yet another four transistors (equation (8) plus an inverting buffer), giving a total of twelve transistors. All other terms in equation (15) arrive sooner than C_2^* . The final sum selection multiplexors contribute two more transistors. Hence, the critical path from c_{in} to S_{32} in the adder has fourteen transistors, as indicated in Table 1. The inverting output buffers in Figure 5 are not needed in the adder and are therefore not counted as in the critical path.

The path from c_{in} to C_{out} has the same number of serial transistors as the path from c_{in} to S_{32}. Gb_k^* and Pb_k^* are available in eight transistors, hence C_{out}^* is available in twelve transistors and C_{out} in fourteen from equation (7) (Figure 6). The path from c_{in} to C_{out}, however, is not the critical path in terms of actual delay because there is much less capacitive loading on this path than on the path from c_{in} to S_{32}.

We have laid out a 9-bit block of the adder in an advanced bipolar/CMOS process with 1.0μm drawn channel length. The block was laid out in a standard-cell fashion and occupied roughly 700x450 μm^2. No automatic compaction was performed on the layout. The delay of the whole adder operated at 5 volts driving a 300fF load at room temperature is estimated using SPICE to be around 3.0ns.

4 Summary

In summary, we have presented a fully static CMOS implementation of a Ling type adder. The implementation has fewer number of complex gate delays and fewer number of serial transistors in the critical path than other conventional adders (i.e., conditional-sum adder, CLA, carry-select adder, and multiple-output domino logic adder). Compared with a conventional conditional-sum adder, the increase in hardware in the present implementation is negligible.

Two key ideas presented in this paper that allows Ling scheme to be used in CMOS are: (1) the identity $\overline{g}_i\overline{p}_i = \overline{p}_i$ can be used on the P-channel transistor network (equation (4)) and

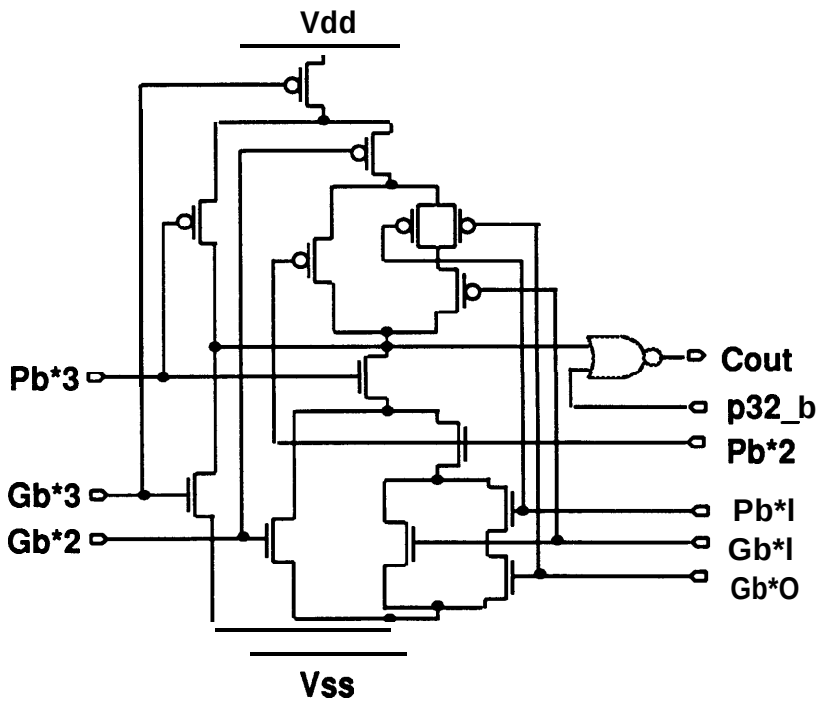

Figure 6: CMOS Implementation of Carry-out.

(2) the factored p term in Ling's equation can be propagated locally (equations (14) and (15)). Together, they allow a reduction of serial transistors in the critical path without any hardware increase. A minor observation is that by using $s_i (= a_i \oplus b_i)$ as the local propagate, the present implementation saves hardware in the local sum logic over the conventional conditional-sum adder (Figure 5).

5 Acknowledgement

The authors wish to thank Mark Horowitz and Antonio R. Todesco for their valuable comments on an early version of this paper.

References

[1] H. Ling, "High Speed Binary Parallel Adder," *IEEE Trans. Comput.*, pp. 799–802, Oct. 1966.

[2] H. Ling, "High Speed Binary Adder," ***IBM Journal of Res. and Dev.,*** no. 3, pp. 156-166, May 1981.

[3] G. Bewick, G. D. P. Song, and M. J. Flynn, "Approaching a Nanosecond: A 32-Bit Adder," in *Proc.* ***of International Conference on Computer Design,*** pp. 221–224, 1988.

[4] S. Waser and M. J. Flynn, ***Introduction to Arithmetic for Digital Systems Designers.*** New-York: Holts, Rinehart and Winston, 1982.

[5] L. A. Glasser and D. W. Dobberpulh, ***The Design and Analysis of VLSI Circuits.*** Readings, Massashusetts: Addison-Wesley, 1985.

[6] E. J. McCluskey, ***Logic Design Principles with Emphasis on Testable Semicustom Circuits,*** ch. 2. Englewood Cliffs, New Jersey: Prentice-Hall, 1986.

[7] I. S. Hwang and A. L. Fisher, "A 3.1ns 32b CMOS Adder in Multiple Output DOMINO Logic," in ***IEEE International Solid-State Circuit Conference,*** pp. 140-141, 1988.

[8] J. P. Uyemura, ***Fundamentals of*** *MOS* ***Digital Integrated Circuits,*** ch. 6-9. Readings, Massachusetts: Addison-Wesley, 1988.

[9] V. G. Oklobdzija and E. R. Barnes, "Some Optimal Schemes for ALU Implementation in VLSI Technology," in *Proc.* 7^{th} ***Symposium on Computer Arithmetic,*** pp. 2-8, Jun. 1985.