

Correspondence

A Recursive Carry-Lookahead/Carry-Select Hybrid Adder

Vitit Kantabutra

Abstract—We present a very fast adder for double-precision mantissas, which is an improvement on Lynch and Swartzlander's Spanning Tree Carry Lookahead Adder or Redundant Cell Adder (*IEEE Trans. Comput.*, Aug. 1992 and IEEE ARITH10 Symp., 1991) which was implemented in AMD's Am29050 microprocessor. Our adder is faster than theirs mainly because we use Manchester carry chains of various lengths instead of chains of all the same length.

Index Terms—Carry-lookahead, carry-select, computer arithmetic circuits, dynamic CMOS, double-precision mantissas, dynamic CMOS, fast adders.

I. INTRODUCTION

We present a very fast adder which is an improvement on Lynch and Swartzlander's Spanning Tree Carry Lookahead Adder (henceforth STCLA), also known as the Redundant Cell Adder [9], [10]. When implemented in AMD's 1- μm dynamic CMOS, the STCLA adds two 56-b operands in 3.2 ns, measured from the clock edge to the slowest sum bits.¹ The STCLA adds numbers quickly by generating carry-in signals for bit positions that are spaced 8 b apart. All such signals are generated within a little less time than 3.2 ns, so that the entire addition process takes 3.2 ns with the help of a carry-select mechanism. In this correspondence, we will present a new, completely laid out network that generates carry-in signals for bit positions that are spaced approximately 16 b apart. According to conservative simulation, this network generates the carry-in signals fast enough that an entire 56-b adder based on the network can add in approximately 1.85 ns, provided that the rest of the adder can quickly provide the conditional output bits to be selected by those carry-in signals computed by the network. We also design (this time without a complete layout) the rest of the adder which is likely to provide such sum bits quickly enough. All of our simulation uses a MOSIS 1- μm process at room temperature, with a rather short effective channel length of approximately 0.712–0.766 μm . We also simulated Lynch and Swartzlander's STCLA using the same technology and transistor sizes (mostly 7- μm drawn width) as the ones used for simulating our network, and found that the STCLA adds in 2.75 ns. Wire capacitances are ignored in our simulation of the STCLA (but not in the simulation of our own main network), and so actually the STCLA would probably be slower than this when implemented in the technology that we have considered.

Before the STCLA, traditional carry-lookahead adders were commonly known as the fastest adders. A fast implementation of such

an adder using especially fast circuit techniques adds two 32-b operands in 3.1 ns using 0.9 μm CMOS [5]. In another paper [11], a slightly modified version of the carry-lookahead adders adds two 64-b numbers in 4.5 ns using 1- μm static CMOS, and can also add two 96-b numbers in the same amount of time. The authors of that paper also showed that a conventional carry-lookahead adder would add two 64-b numbers in 5.4 ns. There was also a new adder [13] that was claimed to be "20–28% faster than the fastest known binary lookahead adder." If we assume that the authors' point of reference of the "fastest known binary adder" is the 3.1-ns 32-b 0.9- μm CMOS adder of [5], then a 32-b version of their design might run as much as 28% faster, i.e., with 2.2 ns of delay. Our 56-b 1- μm CMOS adder would still be faster than theirs. Observe, however, that their speed estimates were based on simple gate delay models and not on simulated delay figures, and thus the comparison may not hold.

Note that the idea of using Manchester carry chains of various lengths may be applicable to designing adders of larger or smaller operands than 56 b as well. The 56-b operand size is an ample size for adding the 52-b mantissas of IEEE Standard-754 double-precision floating-point numbers, with a few extra bits of precision. Note that we have also provided for a carry-out signal in our design.

The STCLA, on which we improved, is a hybrid between a carry-lookahead adder and a carry-select adder, and is significantly faster than both classical carry-lookahead adders and classical carry-select adders in dynamic CMOS. The STCLA uses a "tree" of 4-b Manchester carry-lookahead chains (which is not a tree in the theoretical sense of the word) to generate the carries into bit positions 8, 16, 24, 32, 48, and 56, thus earning the STCLA its name. Two 8-b ripple adders are used for generating the sum outputs for assumed carry inputs of 0 and 1. Once the true carry-in to each 8-b block is known, that carry-in is used to select the correct sum bits from the ripple adders. In order to get the carry-in signals at 8-b intervals, the STCLA needs as inputs to some cells the "propagate" and "generate" signals of overlapping groups of bits, thus earning the STCLA its older name, "Redundant Cell Adder." In addition, the STCLA also needs intermediate "propagate" and "generate" outputs from some of the Manchester chains, making these chains rather heavy loads to the input signals.

The main idea behind our improvement on the STCLA is to employ Manchester chains of various lengths instead of using only 4-b chains. Lynch and Swartzlander already suspected during the conference presentation of their result [9] that this idea might give rise to a faster adder. In their journal paper [10], they mentioned this again, and also mentioned that using fixed-length Manchester chains of lengths 3 exclusively or 5 exclusively would not yield faster adders.

The idea of varying the lengths of bit groups has been used to speed up carry propagation, but only in other types of adders than the hybrid type considered in this correspondence. The techniques used for varying the lengths of bit groups in those other adders certainly do not seem to be applicable to the type of adders considered here. For example, many researchers, including the author of this correspondence [2]–[4], [6]–[8], [12], [14] used the idea to speed up carry propagation in carry-skip adders. Turrini's 32-b, near-optimum multilevel carry-skip adder, implemented in late-1980's

Manuscript received November 17, 1992; revised March 15, 1993, and June 17, 1993.

The author is with the Department of Computer Science, State University of New York, Brockport, NY 14420.

IEEE Log Number 9213763.

¹The STCLA was actually designed to accommodate 64-b operands, but only 56 b are needed. Note also that the delay due to the computation of the bit-level propagate and generate signals is not included in the timing of either the STCLA or the adder to be presented in this correspondence because these computations occur before the clock asserts.

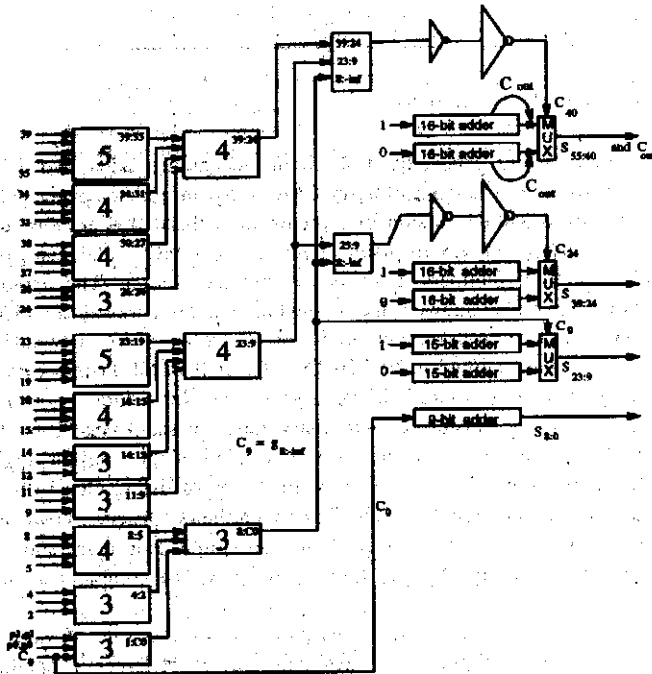


Fig. 1. The main network of Manchester carry chains in the adder presented in this correspondence; the diagram also includes the buffers for driving the multiplexers, as well as the block adders.

BCL technology, added in 1.7 ns.² In [3], [15], the idea of using bit blocks of various lengths was used to speed up traditional block-carry-lookahead adders. However, no one before now has sped up carry-lookahead/carry-select-hybrid adders by using bit groups of various lengths.

II. THE NEW ADDER

The adder presented in this correspondence is a *carry-select adder*, which for our adder means the following: partition each 56-b operand into several blocks of contiguous bit positions. (Both operands are partitioned in the same way.) For each block, add together the parts of the two operands that are in the block. In fact, there are *two* additions to be done simultaneously, one with an assumed carry-in of 0 and the other with an assumed carry-in of 1. Simultaneously with these additions, compute the true carry-in signal to the least significant bit of each block. This signal will be used to select the correct sum output for each bit position in the block, as well as the carry out of the entire adder (in case the block in question is the most significant one).

Clearly, the speed of the entire 56-b adder depends on the speed of the adders for the bit blocks as well as the speed of the circuit for computing the carry-in signals to the blocks. *Our main contribution* is a very fast circuit for computing these carry-in signals. This circuit, a network of dynamic Manchester Carry Chains, is shown in Fig. 1. The laid out version is shown in Fig. 2. The size of the layout is 930 × 240 μm, and can be made smaller if necessary. The network is simulated, and is found to be significantly faster than the Manchester Carry Chain network used for the STCLA ([10], Fig. 2, not shown in the present correspondence), even with the parts

²Turrini's adder was static, and thus his timing included the computation of the bitwise "propagate" signals which are excluded from consideration in our adder as well as in the STCLA because such computation (as well as the computation of the bitwise "generate" signals) occurs before the clock asserts. (Our timing, like Lynch and Swartzlander's, is from clock edge to sum.) Also, the timing of Turrini's adder included the time to drive a 4 mm result bus. Turrini's 32-b adder was laid out in 2100 × 300 μm, which may be smaller than a 32-b adder with a design similar to ours.

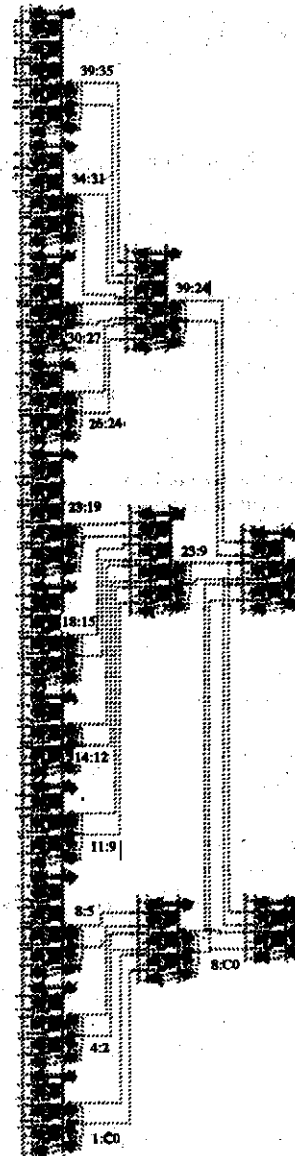


Fig. 2. The layout of the network of Manchester carry chains used in the adder presented in this correspondence using two layers of metal.

pertaining to the most significant 8 b removed, and despite the fact that each of our network's outputs drives up to approximately twice as many multiplexers as an output of the network used for the STCLA. However, our network has the obvious disadvantage that the carry-in signals that it outputs are few and far between, thereby requiring very fast (and large) adders for the large blocks of bit positions. These "block adders," shown in Fig. 3, are also of the carry-lookahead/carry-select hybrid type, which explains the term "recursive" in the name of our 56-b adder. The block adders will be explained in the next section.

III. DETAILS ON THE COMPONENTS

We will now give details on the components of our adder. We start with the Manchester carry chains. In the network of Manchester chains for generating the carry inputs to the blocks, we never need intermediate outputs, and thus a 4-b chain in our circuit would be as shown in Fig. 4, which is the same as [10, Fig. 4(c)]. Our transistors are all 7 μm wide (drawn), except in the two inverters at the output of the chain. Their sizes will be discussed later. A 2-, 3-, or 5-b

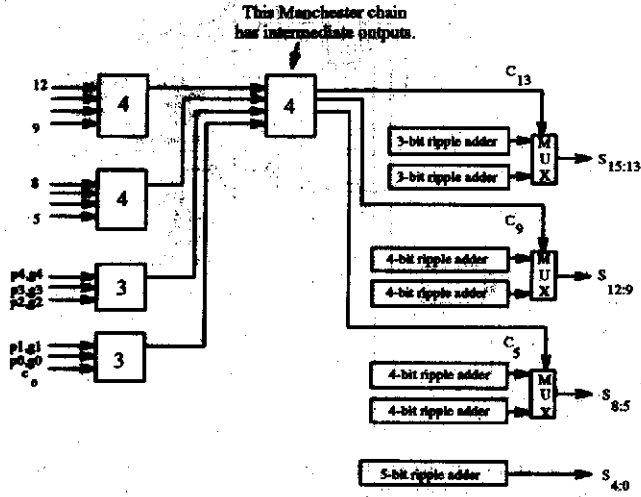


Fig. 3. A block adder.

Manchester chain would basically be the same circuit, but shortened or elongated in the most obvious fashion. The chain with c_0 input is the same as in Lynch and Swartzlander's journal correspondence's Fig. 4(d), but shortened by 1 b. Note that we did not optimize the transistor sizes, but simply picked some reasonable sizes that would yield a reasonably small layout area and would still be large enough not to be excessively affected by wire capacitance. Careful optimization would no doubt yield faster operation. However, we must note that there are many degrees of freedom involved in such optimization, and such optimization would be a substantial piece of research in itself.

All the Manchester chains in our network for generating the block carry-in's have no intermediate outputs, a fact that makes our network faster and smaller than that of Lynch and Swartzlander's STCLA. As stated earlier, the disadvantage of our network of Manchester carry chains is that it only gives carry-in signals at 16-b intervals (approximately) instead of 8-b intervals. This disadvantage forces us to use fast and complex block adders.

We now examine the components of a block adder, which once again is shown in Fig. 3. The components of a block adder are a single 4-b Manchester carry chain with intermediate outputs, four Manchester chains without intermediate outputs, 4- and 5-b ripple adders, and multiplexers. We now discuss the ripple adders. These are made of alternating odd and even ripple cells, as shown in Fig. 5. The transistors we used for these cells are mostly $7\mu\text{m}$ wide. (The cells came from a paper by Guyot *et al.* [4].) These ripple cells are used because they propagate carry signals very quickly. According to our simulation, a carry signal ripples through 2 b in 0.55 ns. See another correspondence by this correspondence's author [6] for simulation details. Notice that we put the longest ripple chain (5 b long) in a place where the sum bits do not have to waste any time going through multiplexers inside the block adder. The sum bits that must be multiplexed inside the block adders are from ripple chains that are at most 4 b long. Such a chain would entail only 1.1 ns of carry propagation time, and a very small amount of extra time to change the sum bits. Note that in the simulation of the ripple cells, no wire length parasitics are included since the part of the adder containing these cells have not been laid out. Thus, we cannot say for sure that the sum signals and the carry-out will be ready for selection by the multiplexers in time. However, we must note that since the ripple cells are static, some or all of the rippling can be done before the clock asserts if it does not disrupt the operation of the system as a whole.

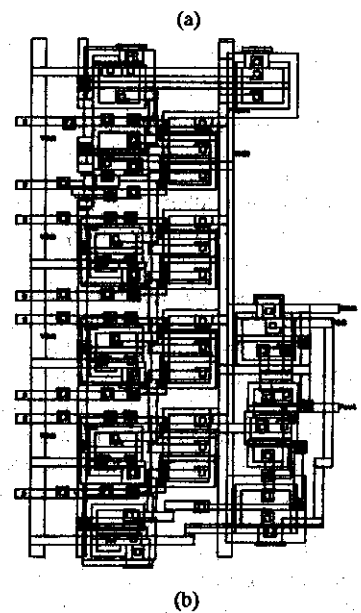
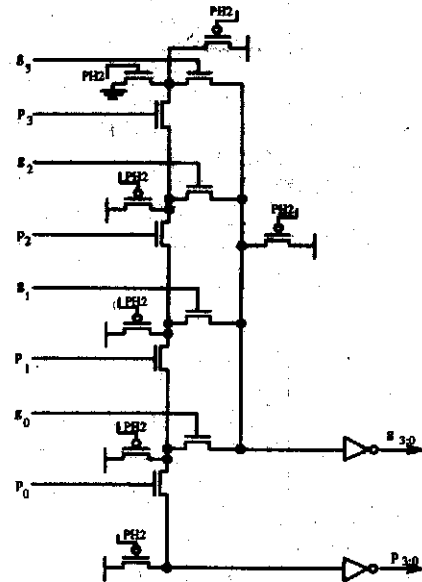


Fig. 4. (a) A Manchester carry chain with four inputs and no intermediate outputs. (b) A layout of the Manchester carry chain diagrammed in (a).

The multiplexers used in our adder are very simple 2-to-1 multiplexers, which are quite standard in CMOS circuits.

IV. THE SIMULATIONS AND THEIR RESULTS

We start by simulating the main network of Manchester carry chains shown in Figs. 1 and 2. We first made rough estimates of each path's delay by taking the sum of the squares of the lengths of each Manchester chain in the path. For example, the path from bit position 9 to upper right "generate" output before the buffers is roughly $3^2 + 4^2 + 2^2 = 29$. Since this happens to be one of the longest paths under this estimate, we will simulate it first. (In fact, the sizes of the Manchester chains in the network were determined so as to minimize the maximum signal delay under this estimate first, and then fine-tuned after some actual signal delays had been determined by simulation. This minimization involved balancing the various path lengths so that they are approximately equal.)

The setup for the simulation of the path from bit position 9 all the way to the most significant 16 b of the sum outputs, including the

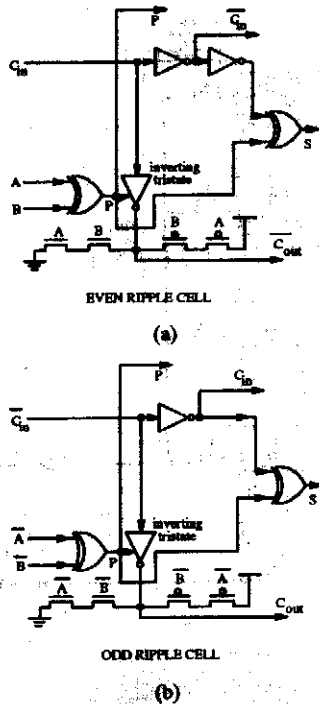


Fig. 5. The two types of fast ripple cells.

delays of the buffers and the multiplexers, is shown in Fig. 6.³ Note that all parts of the circuit are realistically loaded, except the sum output which is only loaded with an inverter ($7\ \mu\text{m}$ PMOS, $4\ \mu\text{m}$ NMOS). Initially, we assume zero-length wires. The effect of long wires will be taken care of later, where we also change the sizes of inverters to help alleviate the effect of wire parasitics. The inverters at the output of each Manchester chain will now be assumed to have a PMOS width of $7\ \mu\text{m}$ and an NMOS width of $4\ \mu\text{m}$. The large inverters shown at the right-hand end of the network of Manchester carry chains are of size e and e^2 times the size of the regular inverters found at the outputs of the chains. The SPICE parameters of the MOS transistors are those of the HP CMOS26B process (an n-well triple metal process, $1.0\ \mu\text{m}$ drawn features, although we did not take advantage of the third level of metal), which can be obtained from MOSIS. The simulated delay using PRECISE⁴ between the point at which the clock input reaches $2.5\ \text{V}$ and the point at which the sum output bits reach $2.5\ \text{V}$ is $1.58\ \text{ns}$.⁵

There is a path in the main network of Manchester carry chains in which the rough estimate of the delay up to before the buffers is 30, which is more than any other path's estimated delay. This is the path from bit position 19 to the upper right-hand "generate" output. The delay through that path and all the way to the 16 sum output bits 40–55 turns out to be $1.7\ \text{ns}$, which will turn out to be the longest path in the entire adder, just as predicted by the rough estimate.

³In the simulation without wire capacitances, we did not include the mux for selecting the carry output from the entire adder. However, this mux was included for all simulation in which wire capacitances were included.

⁴A fast circuit-level simulation software package by Electrical Engineering Software based on SPICE2G.

⁵We used the curve tracing facility of PRECISE to ensure the accuracy of this reading. According to the curve tracing facility, the sum output bits S_{40} – S_{55} reach $2.558\ \text{V}$ at $3.077\ \text{ns}$, which is $1.577\ \text{ns}$ or about $1.58\ \text{ns}$ after the time at which the input clock has reached $2.5\ \text{V}$. The curve tracing facility does not have a reading for the time at which the output was exactly $2.50000\ \text{V}$. Later, when we consider the wire parasitics and do the final simulation, we will take two readings and interpolate. The two readings will be made just before and just after the output voltage reaches exactly $2.5\ \text{V}$.

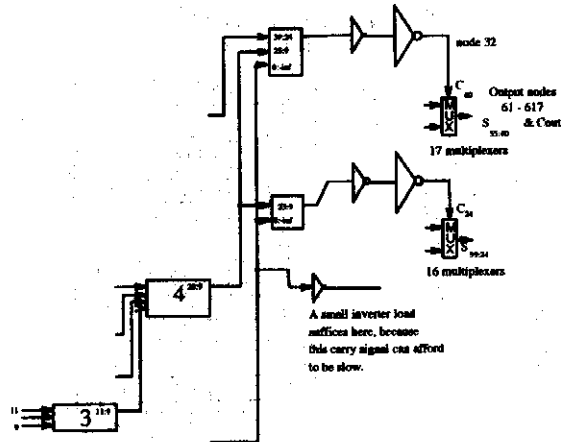


Fig. 6. The setup for simulating the path from bit position 9 to the most significant 16 sum output bits and the carryout of the entire adder.

Source Bit Position	Delay from Clock Rising Edge to Sum Bits (40–55) $2.5\ \text{V}$ point to $2.5\ \text{V}$ point (No wires cap. 70 inverters)	Delay from Clock Rising Edge to Sum Bits (40–55) & Cout $2.5\ \text{V}$ point to $2.5\ \text{V}$ point (2007 isolated wire cap. 70 inv & 500 wire from 3rd level mux output)	Same as previous column except that the parallel-sized inverter at the MCC network output has been removed.
9	1.58ns	1.86ns	
19	1.7ns	1.88ns	
C_0	1.64ns	1.92ns	
5	1.66ns	1.92ns	
27	1.58ns	1.87ns	
15	1.63ns	1.86ns	
35	not tested	1.97ns (longest)	1.85ns (longest)
31	not tested	1.87ns	
24	not tested	1.82ns	

Fig. 7. Table showing delays in the main Manchester carry chain network.

The other paths simulated were the paths from the following bit positions to the most significant 16 b of the sum: C_0 , bit 5, bit 27, and bit 15. The simulation results for all of the six paths in the network that we simulated without wire parasitics are tabulated in Fig. 7, second column. The other paths are either clearly subpaths of some simulated paths, or are roughly estimated to be significantly shorter than 30. Thus, we will not need to simulate them as long as wire lengths have not been considered. For example, the paths from bits 16, 17, and 18 to the most significant 16 sum and the carry-out bits is shorter than the path from bit 15 that we have tested. And the path from bit position 35 to the most significant 16 sum and the carry-out bits is shorter than the path from bit 19. Also, the path from each bit position i to each of the least significant 40 b is shorter than the path from the same bit i to the most significant 16 sum and the carry-out bits.

As pointed out by a referee, our method of adding up the squares of the lengths of Manchester chains is only an approximate method for comparing path lengths. Thus, if we estimate p_1 to be longer than p_2 by only a small amount by this method, it does not necessarily mean that p_1 has to be longer than p_2 at all, unless p_2 is truly a subpath of p_1 or p_2 is exactly the same circuit as some subpath of p_1 with exactly the same load. As an example, the path from C_0 to the most significant sum bits has an estimated length of 27, while the path from bit 5 has an estimated length of 26. Yet, the former path has a simulated delay of $1.64\ \text{ns}$, which is shorter than the $1.66\ \text{ns}$ delay of the latter path. These two paths may have similar loads, but neither is a subcircuit of the other, and thus they are not comparable, except in the approximate sense, that is, they are approximately the same length.

Now, we deal with the wire parasitics in our main network of Manchester carry chains. We laid out the network (see Fig. 2) so that

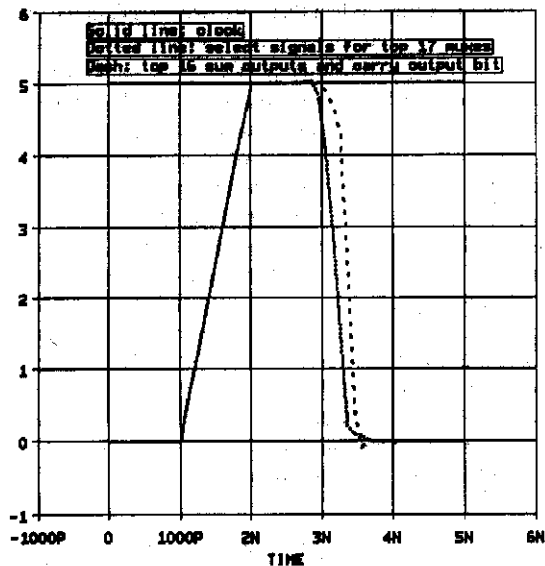


Fig. 8. Simulation results for the longest path—the path from bit 35 to the 16 most significant sum outputs and the carry-out of the entire adder.

the longest path according to our wire-free simulation (the path from bit 19) would have the shortest wires. In an attempt to deal with long wires, we increase the widths of the output inverters of each Manchester chain to $9\ \mu\text{m}$ (PMOS) and $5\ \mu\text{m}$ (NMOS). Now, note that even the longest wire is short enough to be modeled accurately as a capacitance. To determine how much capacitance each wire should have, we note that an isolated, $1.5\text{-}\mu\text{m}$ -wide metal-2 wire on substrate (each wire indeed lies almost entirely on substrate) would have a capacitance of $0.111\ \text{fF}$. We assume quite conservatively that the same wire in parallel with other wires in this technology would have a capacitance no more than twice that much. Thus, we assume the capacitance of a wire of length l to be $2 * 0.111 * l\ \text{fF}$. Additionally, we assume that the carry signals that are output from the two rightmost Manchester chains in Fig. 2 are fed into wires that are up to $500\ \mu\text{m}$ long (isolated). The results of this simulation are tabulated in the third column of Fig. 7. The first thing we notice about these new results is that the path from bit 19 is no longer the longest path. Among the six paths originally simulated (the first six lines in the table in Fig. 7), the paths from bit 5 and from C_0 are now the longest at $1.92\ \text{ns}$ due to long wires. And we note that due to long wires, the paths from bits 31 and 35 to the 16 most significant sum and the carry output of the adder can no longer be assumed to be shorter than any previously simulated paths. Thus, we must simulate them. By doing so, we indeed found that the path from bit 35 is the longest overall, at $1.97\ \text{ns}$.

In the final simulation step, we eliminate the inverters immediately connected to the outputs of the rightmost two Manchester carry chains, leaving only the large inverting buffers. This shortens the longest path (the path from bit 35) to $1.85\ \text{ns}$. This new delay figure is tabulated in the last column of the table in Fig. 7, and the simulation graph appears in Fig. 8. The other paths are clearly shorter than this since they all go through the same set of inverters which has just been shortened. Thus, the adder delay is $1.85\ \text{ns}$, provided that the conditional sum and carry outputs can be generated fast enough to be selected by the output multiplexers.

Now, we test the 16-b block adders shown in Fig. 3. The 15- and 9-b adders are obviously no slower than the 16-b ones. The two critical paths from the time the clock asserts are clearly the paths

from the input carry (which is the lowest "generate" input), and the generate input at bit position 5. The delay from these two paths to the sum outputs at positions 13–15 are found to be 1.4 and $1.22\ \text{ns}$, respectively. Thus, although these sum outputs are not loaded, it seems likely that even when they are loaded by the wires and the multiplexers of the main network of Manchester carry chains, they would still provide their signals to those multiplexers in ample time. Thus, the only thing that could delay the entire adder beyond $1.85\ \text{ns}$ would be the static ripple chains which have fixed carry inputs and have sum inputs that are ready before the clock asserts.

V. CONCLUSION

We have designed and simulated a very fast $1\text{-}\mu\text{m}$ (drawn) dynamic CMOS adder for double-precision mantissas. This adder is faster than Lynch and Swartzlander's Spanning Tree Carry-Lookahead Adder. The main technique we used for obtaining the speedup was to use various-sized Manchester carry chains in order to achieve a balance among several worst-case carry path lengths.

ACKNOWLEDGMENT

The author would like to thank the anonymous referees for their help in improving this correspondence. He would also like to thank T. Lynch for some helpful advice and Prof. J.-M. Delosme of Yale University for suggesting that the author simulate Lynch and Swartzlander's adder in order to obtain a comparison.

REFERENCES

- [1] M. Annaratone, *Digital CMOS Circuit Design*. Boston, MA: Kluwer Academic, 1986.
- [2] P. K. Chan and M. D. F. Schlag, "Analysis and design of CMOS Manchester adders with variable carry-skip," *IEEE Trans. Comput.*, Aug. 1990.
- [3] P. K. Chan, M. D. F. Schlag, C. D. Thornborson, and V. G. Oklobdzija, "Delay optimization of carry-skip adders and block carry-lookahead adders," *IEEE Trans. Comput.*, Aug. 1992.
- [4] A. Guyot, B. Hochet, and J.-M. Muller, "A way to build efficient carry-skip adders," *IEEE Trans. Comput.*, Oct. 1987.
- [5] I. S. Hwang and A. L. Fisher, "A $3.1\ \text{ns}$ 32B CMOS adder in multiple output domino logic," *IEEE J. Solid-State Circuits*, vol. 24, pp. 358–369, Apr. 1989.
- [6] V. Kantabutra, "Designing one-level carry-skip adders," *IEEE Trans. Comput.*, vol. 42, no. 6, pp. 759–764, June 1993.
- [7] —, "Accelerated two-level carry-skip adders: A type of very fast adders," *IEEE Trans. Comput.*, vol. 42, no. 11, pp. 1389–1393, Nov. 1993.
- [8] M. Lehman and N. Burla, "Skip techniques for high-speed carry propagation in binary arithmetic units," *IRE Trans. Electron. Comput.*, p. 691, Dec. 1961.
- [9] T. Lynch and E. E. Swartzlander, Jr., "The redundant cell adder," in *Proc. 10th IEEE Symp. Comput. Arithmetic*, Grenoble, France, 1991.
- [10] —, "A spanning tree carry lookahead adder," *IEEE Trans. Comput.*, vol. 41, Aug. 1992.
- [11] A. Naini, D. Bearden, and W. Anderson, "A $4.5\ \text{ns}$ 96B CMOS adder design," in *Proc. IEEE 1992 Custom Integrated Circuits Conf.*, Boston, MA, May 1992.
- [12] V. G. Oklobdzija and E. R. Barnes, "Some optimal schemes for ALU implementation in VLSI technology," in *Proc. 7th Symp. Comput. Arithmetic*, 1985.
- [13] H. R. Srinivas and K. K. Parhi, "A fast VLSI adder architecture," *IEEE J. Solid-State Circuits*, vol. 27, May 1992.
- [14] S. Turrini, "Optimal group distribution in carry-skip adders," in *Proc. 9th IEEE Symp. Comput. Arithmetic*, 1989.
- [15] T. Vo and P. P. Gelsinger, "Optimally partitioned regenerative carry lookahead adder," U.S. Patent 4 737 926, U.S. Cl. 364-787, filed Jan. 21, 1986, issue date Apr. 12, 1988.