01 Nov 2003

Design and Characterization of Null Convention Self-Timed Multipliers

Satish K. Bandapati

Scott C. Smith
Missouri University of Science and Technology

Minsu Choi
Missouri University of Science and Technology, choim@mst.edu

Design and Characterization of Null Convention Self-Timed Multipliers

Satish K. Bandapati, Scott C. Smith, and Minsu Choi
University of Missouri-Rolla

Editor's note:
This article presents various 4-bit × 4-bit unsigned multipliers designed using the delay-insensitive null convention logic paradigm. Simulation results show a large variance in circuit performance in terms of power, area, and speed. This study will serve as a good reference for designers who wish to accomplish high-performance, low-power implementations of clockless digital VLSI circuits.

—Yong-Bin Kim, Northeastern University

FOR THE PAST TWO DECADES, digital design has focused primarily on synchronous, clocked architectures. However, because clock rates have significantly increased while feature size has decreased, clock skew has become a major problem. To achieve acceptable skew, high-performance chips must dedicate increasingly larger portions of their area to clock drivers, thus dissipating increasingly higher power, especially at the clock edge, when switching is most prevalent.

As this trend continues, the clock is becoming more difficult to manage, causing renewed interest in asynchronous digital design. Researchers have demonstrated that correct-by-construction asynchronous paradigms, particularly null convention logic (NCL), require less power, generate less noise, produce less electromagnetic interference, and allow easier reuse of components than their synchronous counterparts, without compromising performance.[1] Furthermore, we expect these paradigms to allow much greater flexibility in the design of complex circuits such as SoCs. Because these circuits are delay insensitive, they should drastically reduce the effort required to ensure correct operation under all timing scenarios, compared to equivalent synchronous designs. Also, the self-timed nature of correct-by-construction SoCs should allow designers to reuse previously designed and verified functional blocks in subsequent designs, without significant modifications or retiming effort within a reused functional block. Such SoCs might also provide simpler interfacing between the digital core and nontraditional functional blocks.

One of the first tasks necessary to help integrate NCL into the semiconductor design industry is to develop and characterize the key components of a reusable-design library. Of fundamental importance are arithmetic circuits, including the multipliers we describe in this article and the ALUs we described elsewhere.[2] Here, we present 4-bit × 4-bit unsigned multipliers that we designed using the delay-insensitive NCL paradigm. They represent bit-serial, iterative, and fully parallel multiplication architectures. The figures depicting each multiplier component are available at http://www.ece.umr.edu/~smithsco.

NCL overview

NCL is a self-timed logic paradigm in which control is inherent in each datum. NCL follows the so-called weak conditions of Seitz's delay-insensitive signaling scheme.[3] Like other delay-insensitive logic methods, the NCL paradigm assumes that forks in wires are isochronic.[4] Various aspects of the paradigm, including the NULL (or spacer) logic state from which NCL derives its name, have origins in Muller's work on speed-independent circuits in the 1950s and 1960s.[5]

Delay insensitivity

NCL uses symbolic completeness of expression to achieve delay-insensitive behavior. A symbolically com-

0740-7475/03/$17.00 © 2003 IEEE Copublished by the IEEE CS and the IEEE CASS

plete expression depends only on the relationships of the symbols present in the expression without reference to their time of evaluation.[6] In particular, dual- and quad-rail signals or other mutually exclusive assertion groups (MEAGs) can incorporate data and control information into one mixed-signal path to eliminate time reference. For NCL and other circuits to be purely delay insensitive, assuming isochronic wire forks, they must meet the input-completeness and observability criteria.[6] Furthermore, when circuits use the bitwise completion strategy with selective input-incomplete components, they must also meet the completion-completeness criterion.[6]

Most multirail delay-insensitive systems,[3,4,7] including NCL systems, have at least two register stages, one at both the input and the output. Two adjacent register stages interact through request and acknowledge lines K_i and K_o to prevent the current DATA wavefront from overwriting the previous DATA wavefront by ensuring that the two are always separated by a NULL wavefront.

Logic gates

NCL differs from other delay-insensitive paradigms,[3,7] which use only one type of state-holding gate, the C-element.[5] A C-element behaves as follows: When all inputs assume the same value, the output assumes this value; otherwise, the output does not change. On the other hand, all NCL gates are state holding. NCL uses threshold gates as its basic logic elements.[8] The primary type of threshold gate is the TH*mn* gate ($1 \le m \le n$). TH*mn* gates have *n* inputs. At least *m* of the *n* inputs must be asserted before the output becomes asserted. Because NCL threshold gates are designed with hysteresis, all asserted inputs must be deasserted before the output is deasserted. Hysteresis ensures a complete transition of inputs back to NULL before asserting the output associated with the next wavefront of input data. NCL threshold gates may also include a reset input to initialize the output. Circuit diagrams designate resettable gates by either a D or an N appearing inside the gate along with the gate's threshold. D denotes the gate as being reset to logic 1; N, to logic 0.

Previous work

Researchers have proposed two approaches to designing self-timed multipliers.[9,10] However, neither of these multipliers is delay insensitive, so changing fabrication processes requires that the multipliers undergo extensive timing analysis. Hence, they are not directly comparable to the delay-insensitive NCL designs presented here. On the other hand, a 4-bit × 4-bit, delay-insensitive, 3D, pipelined array multiplier[11] is directly comparable to our designs.

Bit-serial multiplier

Figure 1 shows the logic diagram of the 4-bit × 4-bit serial multiplier we developed using the NCL paradigm. This circuit, like all NCL systems, contains a complete request-acknowledge interface. The multiplier consists of input-complete NCL AND functions, a half adder, and full adders.[12] Other components include a multiplicand interface, a multiplier interface, a sequencer, and dual-rail registers and their associated completion components.[12]

Initially asserting the Reset signal returns the multiplier components to their initial values. The circuit produces the first partial product from the 4-bit parallel multiplicand input and the multiplier's least-significant bit, which is generated by the input-complete NCL AND functions. The circuit then passes these partial-product bits to the adders, which initially add the first partial product to the reset value of DATA0, to produce a combined product along with the least-significant bit of the product output. Then, the circuit produces the next three partial products, using the multiplicand along with each more-significant multiplier bit, and adds them to the combined product, thus generating one additional product bit each cycle. At this time, the multiplicand and multiplier interfaces produce four additional partial products of DATA0, to produce the four most-significant bits of the product. Once the multiplier has produced eight product bits, the inputs to the adders are again DATA0 because of the four DATA0 partial products, and the next multiplication is ready to begin.

This architecture has three registers in the feedback loop so that each adder can feed its sum back to its respective bit position, as required.[7] Two registers between adders store the initial DATA0 combined product and provide the necessary handshaking that allows the combined product to shift to the right each cycle. Finally, there is a register between each AND function and its corresponding adder. Although these registers are not essential, they increase throughput 75% by allowing partial-product generation to take place more independently of the addition circuitry.

Multiplicand interface

The multiplicand interface circuitry initially requests the 4-bit parallel multiplicand *MD* used to produce the first partial product. It then feeds back this multiplicand three more times to produce the remaining three par-

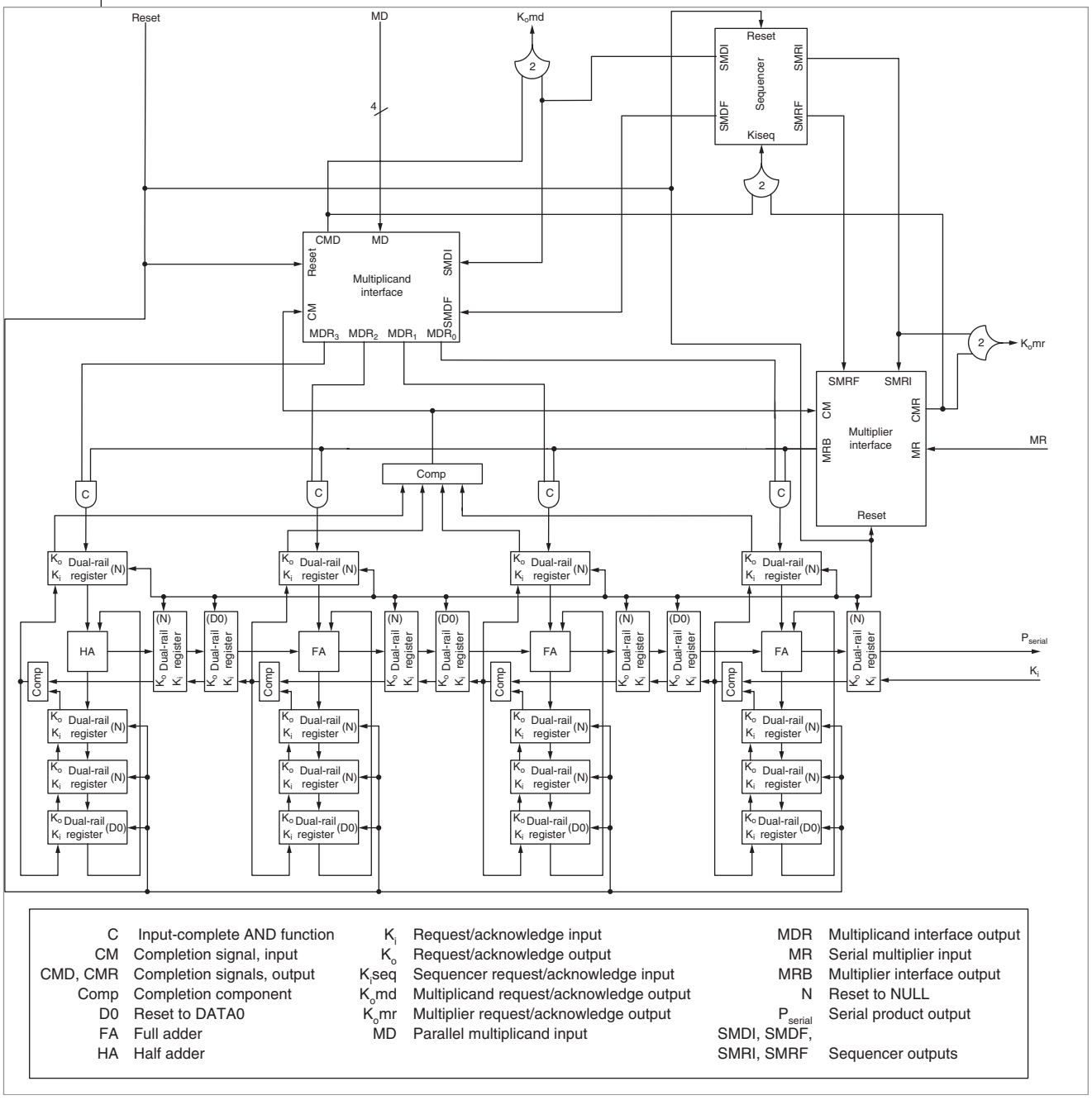

Figure 1. Logic diagram of NCL 4-bit × 4-bit serial multiplier.

tial products, and four more times after that to produce the four DATA0 partial products, as described earlier.

The multiplicand interface consists of

- an embedded select register, comprised of TH33*n* and TH22*n* gates, to select between the external input and the internal feedback;
- a set of TH12 gates to combine the external and internal paths;
- a set of inverting TH14 gates to generate the completion signal; and
- two additional register stages to complete the three-register feedback loop.

Sequencer outputs *SMDI* and *SMDF* make the selection between the internal and external wavefronts. *SMDI* and *SMDF* are mutually exclusive, thus preventing simultaneous selection of the internal and external wave-

fronts. The multiplicand interface is input-complete with respect to the feedback path; thus, it requires feedback data even when the external input is being selected.

Multiplier interface

The multiplier interface circuitry first requests the four multiplier bits (*MR*), from the least to the most significant, to produce the four partial products. It then requests internal generation of DATA0 to produce the four DATA0 partial products, as described earlier.

The multiplier interface consists of

- an embedded select register, comprised of TH33*n* and TH22*n* gates, to select between the external input and a generated DATA0;
- a TH12 gate to combine the external and DATA0 paths; and
- an inverting TH13 gate to generate the completion signal.

Sequencer outputs *SMRI* and *SMRF* perform the selection between the internal and DATA0 wavefronts. *SMRI* and *SMRF* are mutually exclusive, thus preventing simultaneous selection of the internal and DATA0 wavefronts.

Sequencer

The sequencer is controlled by completion signals *CMD* and *CMR* from the multiplicand and multiplier interface circuits. Sequencer outputs *SMDI*, *SMDF*, *SMRI*, and *SMRF* select between the wavefronts for both the multiplicand and multiplier interface circuits. This sequencer is a 16-stage, single-rail, ring structure with seven tokens and two bubbles. A token is a DATA wavefront with a corresponding NULL wavefront. A bubble is either a DATA or a NULL wavefront occupying more than one neighboring stage. When K_i becomes a request for DATA (*rfd*), the DATA wavefront moves through the two NULL bubbles ahead of it, creating two DATA bubbles in its wake. Likewise, when K_i becomes a request for NULL (*rfn*), the NULL wavefront moves through the two DATA bubbles ahead of it, creating two NULL bubbles in its wake. The DATA/NULL wavefront restricts the forward propagation of the NULL/DATA wavefront for each change of K_i, limiting the forward propagation to only the two bubbles. The cycle for the four outputs is

SMDI = 1000000000000000,
SMDF = 0010101010101010,
SMRI = 1010101000000000, and
SMRF = 0000000010101010.

Iterative multiplier

The iterative multiplier's interface is the same as that of the bit-serial multiplier, except for the product, which is an 8-bit parallel output instead of a serial one. Figure 2 shows the logic diagram of the 4×4 iterative multiplier. It consists of a multiplicand interface, input-complete NCL AND functions, shift circuitry, a carry-save adder, selection circuitry, an input sequencer, an output sequencer, a ripple-carry adder, and registers with associated completion components. The registration stage between the AND functions and the shift circuitry is not essential, but it increases throughput 26% by allowing partial-product generation to take place more independently of the shift circuitry.

Initially asserting the Reset signal returns the multiplier's components to their initial values. The circuit produces the first partial product from the 4-bit parallel multiplicand input and the multiplier's least-significant bit, which is generated by the NCL AND functions. The circuit then passes these partial-product bits to the shift circuitry, which does not shift the first partial product. The first partial product is then input to the carry-save adder, which adds the partial product to the reset value of DATA0 to produce a row of carries and a row of sums. These pass through the selection circuitry, which feeds them back to the carry-save adder for the next iteration.

Subsequently, the circuit produces the next three partial products, using the multiplicand along with each more-significant multiplier bit. The shift circuitry shifts the three partial products left one additional bit position in each iteration, and the carry-save adder sums them. Then, the carry-save adder passes the carry and sum rows to the 10-bit register in the output circuitry, while the selection circuitry sends a DATA0 wavefront to the feedback loop, reinitializing it for the next multiplication. Finally, the ripple-carry adder combines the carry and sum rows from the 10-bit register to produce the 8-bit parallel product.

Multiplicand interface

The iterative multiplier's multiplicand interface is the same as that used in the bit-serial multiplier, but it is controlled differently. In the bit-serial multiplier, the multiplicand interface circuitry initially inputs the multiplicand and then feeds it back seven times to produce four partial products, followed by four DATA0 partial products. In contrast, the iterative multiplier's multiplicand interface circuitry inputs the multiplicand and then feeds it back three times to produce four partial products before inputting the next multiplicand.

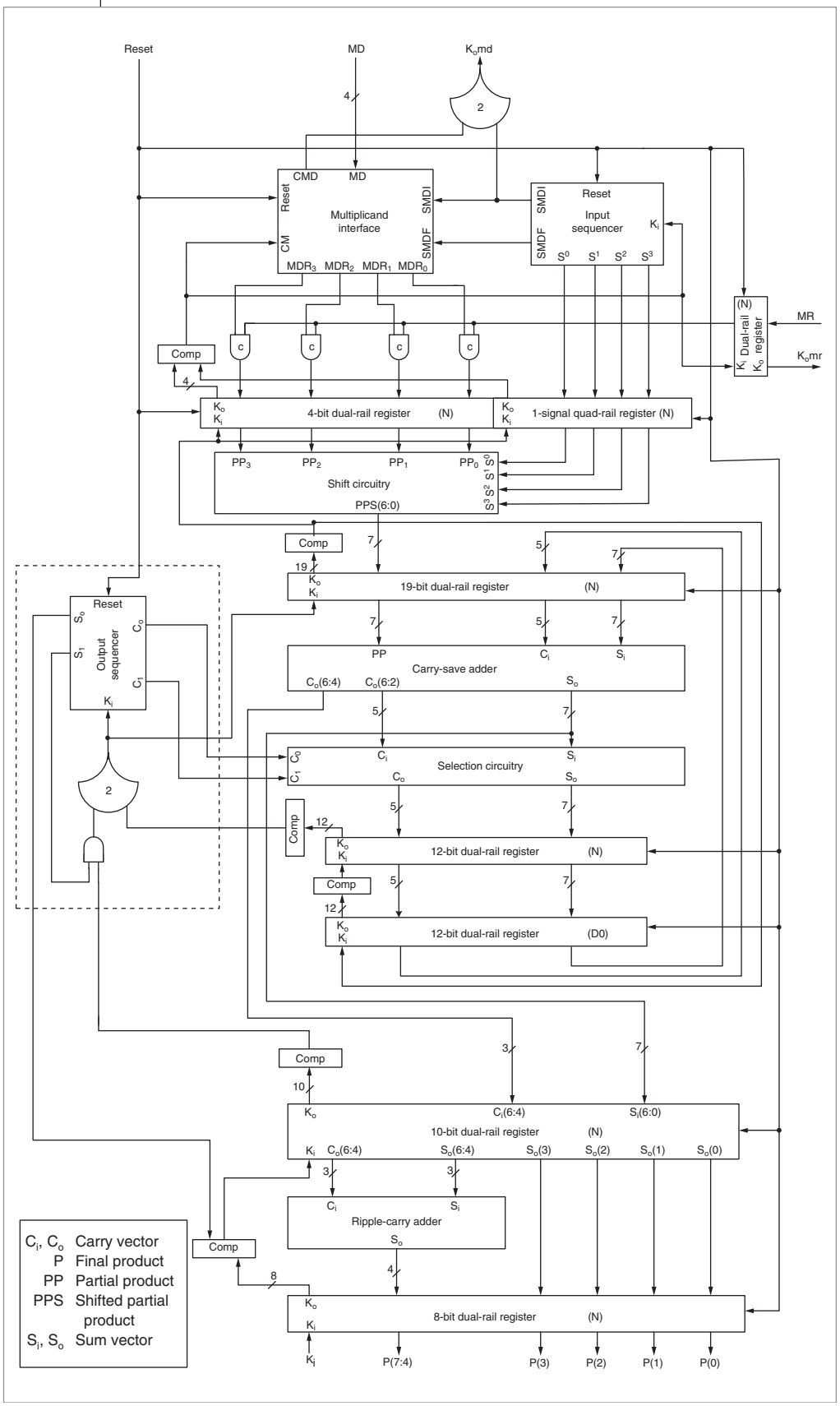

Figure 2. Logic diagram of 4-bit × 4-bit iterative multiplier.

Shift circuitry

The shift circuitry consists of two levels of logic that generate a 7-bit partial product consisting of DATA0 and the 4-bit partial product generated by the AND functions. The shift circuitry shifts the generated partial product left one additional bit position in each iteration. The input sequencer controls the shifting.

Carry-save adder

The carry-save adder consists of a specialized circuit that passes the least-significant bit of the first partial product to the selection circuitry, a half adder, full adders, and a specialized circuit that passes the most-significant bit of the last, or fourth, partial product to the selection circuitry. The specialized LSB circuit replaces a half adder, allowing its use in the second bit position and reducing the number of gates required. This is possible because the least-significant bit of the 7-bit partial product input can only be logic 1 for the first partial product; therefore, this bit will always be logic 0 for the remaining three partial products.

Likewise, the specialized MSB circuit replaces a full adder to reduce the number of gates required. This is possible because the most-significant bit of the 7-bit partial product input can only be logic 1

for the last, or fourth, partial product. Therefore, this bit will always be logic 0 for the first three partial products, and the carry-save addition of the first three partial products will never result in a carry into this bit position. Both specialized circuits are complete with respect to all their inputs, and together they require four fewer gates and 98 fewer transistors. The carry-save adder sends its outputs to both the selection circuitry and the 10-bit register in the output circuitry.

Selection circuitry

The iterative multiplier's selection circuitry consists of one level of logic controlled by the output sequencer; its output feeds back to the carry-save adder. For the first three iterations, the sum row and carry row simply pass through the circuit. In the fourth iteration, the circuit generates a DATA0 wavefront. The circuit is complete with respect to all sum and carry bits for the first three iterations. It is complete only with respect to the carry-save adder output, $C_o(3:2)$, for the fourth iteration. These bits are always logic 0 for this iteration and are therefore not required in the subsequent ripple-carry addition.

Input sequencer

The iterative multiplier's input sequencer has a similar structure to that of the bit-serial multiplier's sequencer. However, the iterative multiplier's input sequencer is an 8-stage, single-rail ring structure with three tokens and two bubbles, and it has different outputs. This sequencer is controlled by its K_i input; it controls the multiplicand interface with its *SMDI* and *SMDF* outputs and the shift circuitry with its S^0, S^1, S^2, and S^3 outputs, which together form a quad-rail signal. The cycle for these six outputs is

SMDI = 10000000,
SMDF = 00101010,
S^0 = 10000000,
S^1 = 00100000,
S^2 = 00001000, and
S^3 = 00000010.

Output sequencer

The output sequencer is the same as the input sequencer, except for its outputs. This sequencer is controlled by its K_i input. It controls the selection circuitry with its C_0 and C_1 outputs, and it controls loading of the 10-bit register in the output circuitry and associated completion with its S_0 and S_1 outputs. As a result of using S_0 as an extra input to the input completion component for this register, the multiplier lets DATA inputs pass to the ripple-carry adder only when S_0 is asserted in the fourth iteration, in which they are added to produce the final product output. The cycle for the four outputs is

C_0 = 10101000,
C_1 = 00000010,
S_0 = 00000010, and
S_1 = 01010111.

Together, the output sequencer, the TH22 gate, and the AND gate (in the dotted box in Figure 2) preserve the multiplier's delay insensitivity, despite the 10-bit register's accepting DATA only every fourth iteration. With the initial reset, the 10-bit register is reset to NULL such that it requests DATA and S_1 is reset to logic 1. This asserts K_i, thus starting the sequencer's cycle. S_0 controls loading of the 10-bit register, and S_1 controls masking of the register's request signal K_o and mimics the requesting of DATA/NULL wavefronts for the first three iterations.

S_0 is asserted only in cycle 7; therefore, the sum and carry rows can pass through the 10-bit register only after the fourth iteration, when the carry-save adder has added all four partial products. S_1 is asserted in cycles 2, 4, and 6 to mimic the requests for DATA and NULL from the 10-bit register. The AND gate masks the 10-bit register for the first three iterations because this register does not receive the DATA wavefronts, which feed back to the carry-save adder; thus, K_o does not change. Instead, only the feedback loop controls the output sequencer and the addition iterations.

S_1 is again asserted in cycle 7 to ensure that the 10-bit register receives the DATA wavefront. This occurs when K_o becomes an *rfn*, thus deasserting the AND gate. S_1 remains asserted in cycle 8 to ensure that the 10-bit register receives the NULL wavefront. This occurs when K_o becomes an *rfd*, thus asserting the AND gate and requesting the first iteration of the next multiplication operation. Next, K_o is once again masked, because the outputs of the next three iterations do not go to the 10-bit register. Therefore, this structure retains delay insensitivity in two ways: First, it ensures that only the feedback loop controls the sequencer and addition iterations when the intermediate results do not go to the output circuitry's 10-bit register. Second, it ensures that both the feedback loop and the 10-bit register control the sequencer and addition iterations during the fourth iteration when the carry and sum rows go to the 10-bit register and to the feedback loop to reset it to DATA0.

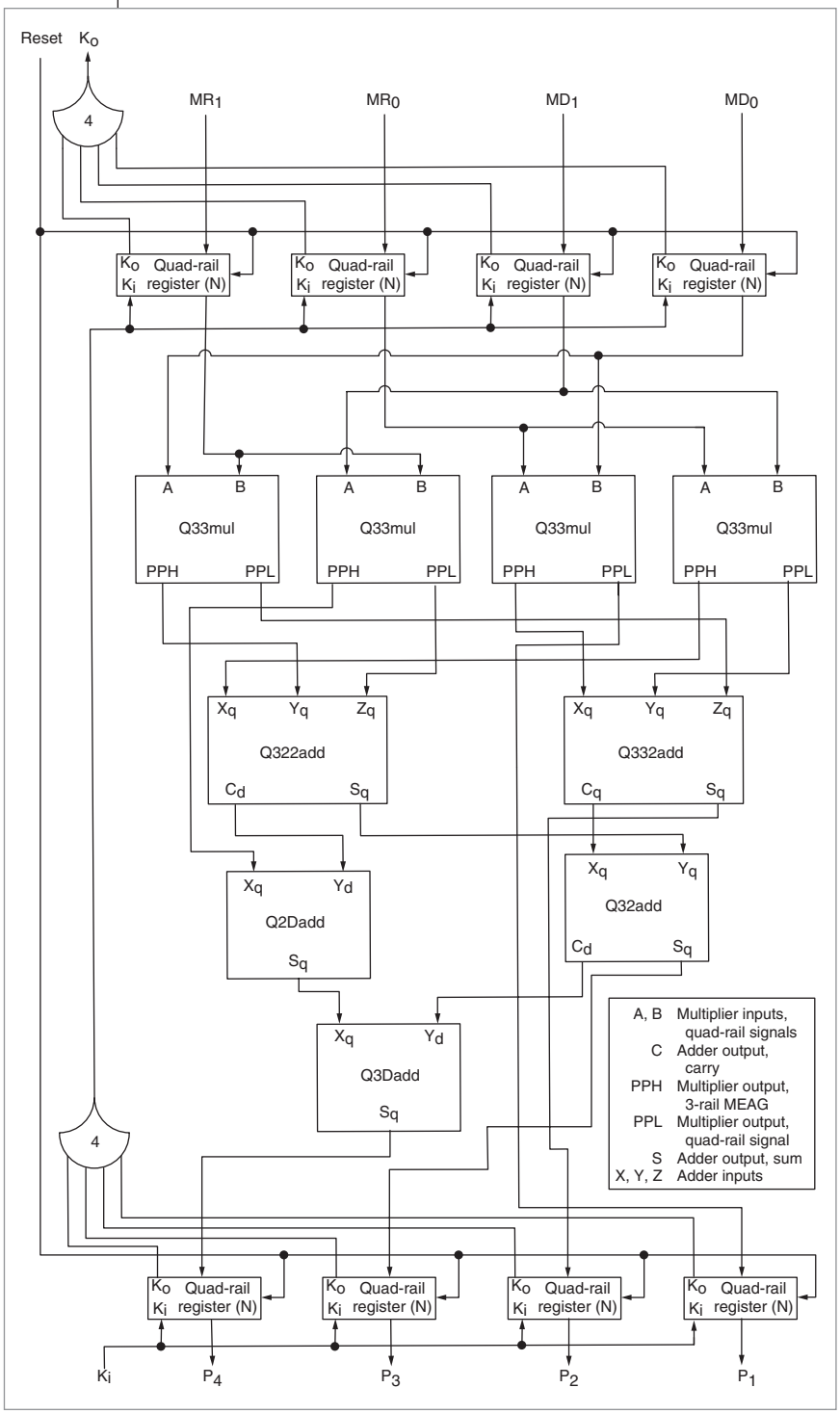

Figure 3. Logic diagram of parallel, nonpipelined, 4 × 4 quad-rail multiplier.

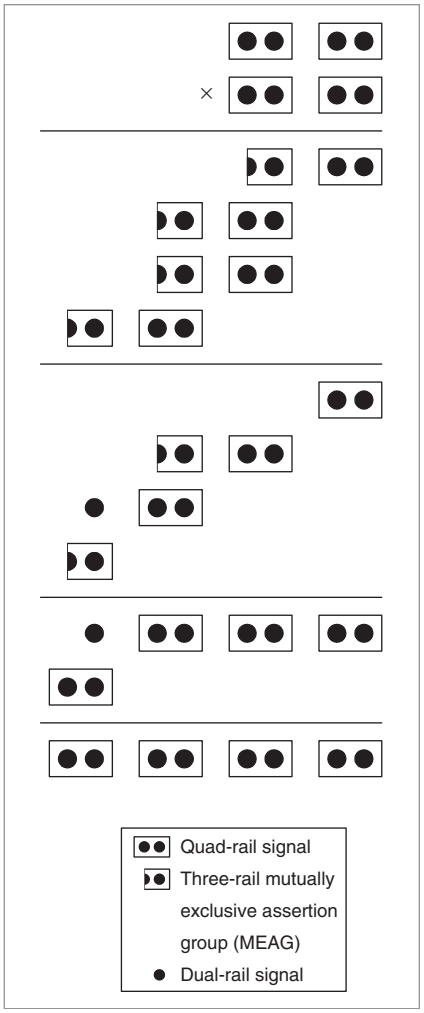

Figure 4. Dot diagram of quad-rail multiplication.

Parallel quad-rail multiplier

Figure 3 shows the logic diagram of a fully parallel, nonpipelined, 4-bit × 4-bit quad-rail multiplier. Both the multiplicand input and the parallel multiplier input consist of two quad-rail signals, and the parallel product input consists of four quad-rail signals. The request-acknowledge interface includes K_o to request both the multiplier and the multiplicand and K_i to acknowledge the product output. This design consists of quad-rail multipliers, denoted Q33mul; an assortment of adders, denoted Q332, Q322, Q32, Q3D, and Q2D; and four quad-rail registers at both the input and output, along with their associated completion components.

Figure 4 shows a dot diagram of the quad-rail multiplication operation. It begins with the parallel generation of all partial products. The multiplication of two quad-rail signals to produce a partial product results in two outputs: less-significant signal *L* and more-significant signal *M*. The largest quad-rail × quad-

rail multiplication is 3×3, which results in an output of 9, represented as $M = 2$, $L = 1$. M has a range of only 0 through 2, so it is representable by a three-rail MEAG, instead of a quad-rail signal, thus requiring one fewer wire. On the other hand, L has the range 0 through 3 and thus must be represented as a quad-rail signal. The next three multiplication levels add the partial products in a Wallace tree fashion. This scheme uses various quad-rail carry-save adders to take advantage of the reduced range of the three-rail MEAGs, thus producing the product consisting of four quad-rail signals.

This multiplier's design has a worse-case path delay of eight gates in the combinational logic and one gate in the completion logic. For an NCL circuit, we estimate worse-case throughput as the worst-case data path delay plus the completion delay, for both the DATA and NULL wavefronts, which comprise one complete DATA/NULL cycle. This calculation is equivalent to twice the sum of the worst-case data path delay and completion delay. The completion delay is calculated as $\lceil \mathrm{Log}_4 N \rceil$, where N is the number of dual-rail or quad-rail signals in a stage's output register. So in this case, the completion delay is one and the initial throughput is (one cycle)/(18 gate delays). However, with a gate-level pipelining method, we can optimally pipeline it, using bitwise completion and a maximum stage delay of three gates.[12] In this method, we insert a register between each level in the dot diagram to increase the circuit's throughput from (one cycle)/(18 gate delays) to (one cycle)/(eight gate delays). If throughput is the main design concern, however, we should choose the parallel dual-rail multiplier because it can be pipelined more finely, with a stage delay of only two gates and a throughput of (one cycle)/(six gate delays), thus resulting in a faster circuit.[12]

Q33mul

The Q33mul circuitry multiplies two quad-rail signals, A and B, to produce a two-signal partial product consisting of the more-significant three-rail MEAG, *PPH*, and the less-significant quad-rail signal, *PPL*. We ensured that this circuit is input complete by adding additional terms to the equation for PPL^0 such that both inputs, A and B, are required even when either is logic 0. The *PPL* circuitry consists of two levels of logic, and the *PPH* circuitry consists of only one level.

Adders

Various quad-rail carry-save adders, which take advantage of the three-rail MEAGs' reduced range to decrease gate count and delay, perform the partial-product addition. A further optimization of the Q3D adder is that it accounts for the fact that the multiplication of two 4-bit unsigned numbers results in an 8-bit product; therefore this adder does not require a carry output. Table 1 lists the input and output types of the various adders. All adder circuits discussed here are inherently input complete.

Table 1. I/O specifications for quad-rail adders. Q3 represents a quad-rail signal of range 0 through 3, Q2 represents a three-rail MEAG of range 0 through 2, and D represents a dual-rail signal of range 0 through 1.

		Output types	
Adder type	**Input types**	**Carry**	**Sum**
Q332add	Q3, Q3, Q2	Q2	Q3
Q322add	Q3, Q2, Q2	D	Q3
Q32add	Q3, Q2	D	Q3
Q2Dadd	Q2, D	—	Q3
Q3Dadd	Q3, D	—	Q3

Other multiplier architectures

Two other NCL multiplier architectures are of interest: a fully parallel dual-rail multiplier, and a three-dimensional pipelined multiplier.

Parallel dual-rail multiplier

The full description and the logic diagram of the fully parallel, nonpipelined, 4-bit $\times$ 4-bit, dual-rail multiplier using full-word completion appear in another article.[12] Both the multiplicand and multiplier consist of four dual-rail signals, and the product consists of eight dual-rail signals. This design contains

- NCL AND functions to generate the partial products,
- carry-save adders consisting of half and full adders to intermediately sum the partial products,
- a ripple-carry adder to produce the final combined product, and
- eight dual-rail registers at the input and output, along with their associated completion component, to provide the necessary handshaking signals.

The multiplier has a worse-case path delay of 10 gates in the combinational logic, but it can be optimally pipelined using bit-wise completion with a maximum stage delay of two gates.[12] This will increase the circuit's throughput from (one cycle)/(24 gate delays) to (one cycle)/(six gate delays).

Table 2. Comparison of NCL multipliers.

Multiplier architecture	Gate count	Transistor count	T_{DD} (ns)	P_{DD} (nW)
Bit-serial	203	2,598	69.71	—
Iterative	418	5,478	33.05	—
Parallel, quad-rail, nonpipelined	245	3,630	9.92	0.74
Parallel, quad-rail, pipelined	315	4,610	5.92	—
Parallel, dual-rail, nonpipelined	145	2,004	9.21	3.34
Parallel, dual-rail, pipelined	320	4,292	3.84	—
Three-dimensional, pipelined	583	7,004	6.77	—

Three-dimensional pipelined multiplier

Taubin, Fant, and McCardle developed a dual-rail, 3D, pipelined multiplier to increase throughput by eliminating broadcasting and completion trees.[11] This architecture uses gate-level pipelining of Manchester adders, combined with a 2D cross-pipeline mesh for multiplicand and multiplier propagation and partial-product bit calculation. The structure is like a two-story building whose second floor sums the partial-product bits generated by the first floor. The first floor also propagates the multiplicand bits in the *y* direction and the multiplier bits in the *x* direction, thus producing the partial-product bits, which propagate in the *z* direction. The second floor consists of Manchester adders connected in carry-save fashion, which sum the partial-product bits and propagate the carry bits in the *x* direction and the sum bits in the *y* direction. The completion signals are local and move in directions opposite those of the data.

Taubin, Fant, and McCardle's multiplier is a 4-bit × 4-bit signed multiplier, so we designed an unsigned version to compare with the other 4-bit × 4-bit unsigned multipliers discussed here. Also, Taubin, Fant, and McCardle's multiplier uses a different technology library, further necessitating our redesign.

Simulation results

We simulated the circuits compared here using a 0.5-micron CMOS process operating at 3.3 V. Table 2 summarizes the characterizations of the various multipliers in terms of speed, area, and power. Gate count is one measure of area; however, because NCL gates vary greatly in size (from two transistors for an inverter to 26 transistors for a TH24 gate), transistor count provides a better means of comparison. Also, because NCL circuits are delay insensitive, speed is data dependent; therefore, we used average cycle time, T_{DD}, for comparison. We calculated T_{DD} as the arithmetic mean of the cycle times corresponding to all 256 possible pairs of input operands. Furthermore, we calculated average power per operation, P_{DD}, for the nonpipelined dual-rail and quad-rail multipliers to compare their encoding schemes. We did this by running a Spice simulation of both designs performing three randomly selected multiplication operations, calculating the total power for these operations (subtracting reset power), and then dividing the total power by 3.

Note that the average cycle time for the nonpipelined, parallel, dual-rail multiplier is less than that of the nonpipelined, parallel, quad-rail multiplier, even though the worse-case delay is less for the quad-rail version. The reason is that average cycle time is based on average-case delay, not worse-case delay; and the dual-rail version has a smaller average-case delay because of the ripple-carry adder's average-case logarithmic behavior. Also, the quad-rail multiplier requires less power per operation than the dual-rail version because there are half as many signal transitions per operation for the quad-rail multiplier (that is, two dual-rail signals transition for each corresponding quad-rail signal transition).

COMPARING THE VARIOUS ARCHITECTURES shows that when speed is the main design goal, an optimally pipelined, parallel, dual-rail multiplier is the best choice. When area is the main concern, a nonpipelined, parallel, dual-rail multiplier is preferable. And, when a design requires minimal power, a nonpipelined, parallel, quad-rail multiplier is best. The architecture that best balances area and speed is the nonpipelined, parallel, dual-rail multiplier, which requires the least area and has the highest speed of the nonpipelined designs. The nonpipelined, quad-rail multiplier best balances speed and power because it is only slightly slower than the dual-rail version but requires significantly less power.

Designers would rarely choose the bit-serial and iterative multipliers because they require more area than the nonpipelined, parallel, dual-rail multiplier and are much slower. These multipliers have more area than the fully parallel version because of the extra circuitry needed to ensure delay insensitivity, such as the three-register feedback loop(s), the sequencer(s), and the interface circuit(s). Also, designers would seldom use

either the parallel, pipelined, quad-rail multiplier or the 3D, pipelined multiplier because both require more area than the parallel, pipelined, dual-rail multiplier, and neither is as fast. The pipelined, quad-rail version is not as fast as its dual-rail counterpart because the worse-case delay of its primary components is greater (three versus two gate delays), and these primary components cannot themselves be pipelined without violating the input-completeness criterion. Therefore, the quad-rail version cannot be as finely pipelined, thus restricting throughput enhancement.

On the other hand, the 3D, pipelined multiplier takes more area because it requires substantially more registers, associated completion components, and larger adder cells. It is slower because of the increased dependence of the completion signals. However, for substantially larger designs, the pipelined, dual-rail multiplier's throughput would decrease because of the extra levels of logic required in the completion components for partial-product generation. In contrast, throughput would remain about the same for the 3D, pipelined design because of its extremely fine-grained, localized completion strategy. ■

Acknowledgment

We thank the University of Missouri Research Board for the funding that made this work possible.

■ References

1. J. McCardle and D. Chester, "Measuring an Asynchronous Processor's Power and Noise," *Proc. Synopsys User Group Conf.*, 2001, http://www.theseus.com/Pdf/snug_boston.pdf.
2. S.K. Bandapati and S.C. Smith, "Design and Characterization of NULL Convention Arithmetic Logic Units," *Proc. 2003 Int'l Conf. VLSI* (VLSI 03), CSREA Press, 2003, pp. 178-184; http://www.ece.umr.edu/~smithsco/ALU.pdf.
3. C.L. Seitz, "System Timing," *Introduction to VLSI Systems*, C. Mead and L. Conway, eds., Addison-Wesley, 1980, pp. 218-262.
4. A.J. Martin, "Programming in VLSI," *Developments in Concurrency and Communication*, C.A.R. Hoare, ed., Addison-Wesley, 1991, pp. 1-64.
5. D.E. Muller, "Asynchronous Logics and Application to Information Processing," *Switching Theory in Space Technology*, H. Aiken and W.F. Main, eds., Stanford Univ. Press, 1963, pp. 289-297.
6. S.C. Smith, "Completion-Completeness for NULL Convention Digital Circuits Utilizing the Bit-Wise Completion Strategy," *Proc. 2003 Int'l Conf. VLSI* (VLSI 03), CSREA Press, 2003, pp. 143-149; http://www.ece.umr.edu/~smithsco/Completeness.pdf.
7. J. Sparso and J. Staunstrup, "Design and Performance Analysis of Delay-Insensitive Multi-Ring Structures," *Proc. 26th Hawaii Int'l Conf. System Sciences* (HICSS-26), vol. 1, IEEE CS Press, 1993, pp. 349-358.
8. G.E. Sobelman and K.M. Fant, "CMOS Circuit Design of Threshold Gates with Hysteresis," *Proc. IEEE Int'l Symp. Circuits and Systems* (ISCAS 98), IEEE Press, 1998, pp. 61-65.
9. A.J. Acosta et al., "Design and Characterization of a CMOS VLSI Self-Timed Multiplier Architecture Based on a Bit-Level Pipelined-Array Structure," *IEE Proc., Circuits, Devices, and Systems*, vol. 145, no. 4, Aug. 1998, pp. 247-253.
10. G.A. Ruiz and M.A. Manzano, "Self-Timed Multiplier Based on Canonical Signed-Digit Recoding," *IEE Proc., Circuits, Devices, and Systems*, vol. 148, no. 5, Oct. 2001, pp. 235-241.
11. A. Taubin, K. Fant, and J. McCardle, "Design of Three Dimension Pipeline Array Multiplier for Image Processing," *Proc. IEEE Int'l Conf. Computer Design: VLSI in Computers and Processors* (ICCD 02), IEEE CS Press, 2002, pp. 104-111.
12. S.C. Smith et al., "Delay-Insensitive Gate-Level Pipelin-