Arithmetic Structures for Inner-Product and Other Computations Based on a Latency-Free Bit-Serial Multiplier Design

Steve Haynal and Behrooz Parhami

Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560, USA

Abstract

Traditional bit-serial multipliers present one or more clock cycles of data latency. When combined with addition operations, as would be needed for an inner product computation, the latency may increase further. In this paper, we extend a design method for latency-free bit-serial multipliers to more powerful bit-serial arithmetic units capable of computing functions of the form $S = VW + X$, $S = VW + X + Y$, $S = VW + X + Y + Z$, $S = VW + XY$, and $S = VW + XY + Z$ with no latency (i.e., with only combinational delay between input and output). We show that the above double multiplication and accumulative capabilities are obtained with small extra cost compared to simple bit-serial multipliers. More specifically, the added cost, contributed mainly by the use of a (7, 3) counter in lieu of a (5, 3) counter in each multiplier cell, is about 50% for the most complex unit, making our designs quite cost-effective. Unsigned or sign-extended 2's-complement numbers may be used to produce arbitrarily long outputs. Since the designs are fully modular, they are easily introduced into VLSI libraries.

Keywords: Bit-serial computation, Convolution, Inner product, Little-endian arithmetic, Multiply-accumulate, On-line arithmetic, Systolic multiplier, Two's-complement multiplication

1. Introduction

Bit-serial arithmetic provides a way to minimize pin count, wire length, and floor space requirements in VLSI designs. However, performing bit-serial arithmetic simply and quickly, especially when all operands are entered serially, poses challenging design and implementation problems. Since bit-serial adders/subtractors are easily realized and on-line bit-serial dividers/square-rooters are not feasible unless a redundant representation and MSD-first or big-endian order is used [3], research in bit-serial arithmetic using conventional binary representations has focused on the design of multipliers and squarers (see, e.g., [1], [2], [5], and the references therein).

In a recent paper, Ienne and Viredaz [4] review past design approaches to bit-serial multiplication and present a new bit-serial multiplier with four important features:

1. No latency cycles between input presentation and output availability.
2. Applicability to both unsigned and 2's-complement operands.
3. Production of full double-precision or longer sign-extended result.
4. Regular and modular designs suitable for VLSI realization.

This new design needs only $N - 1$ modules to produce the $2N$-bit product $P = XY$, given N-bit 2's-complement operands X and Y that are sign-extended to length $2N$. Each module, representing one multiplier slice, incorporates a (5, 3) parallel counter [6] that adds its 5 single-bit inputs to produce a 3-bit binary output representing the sum in the range 0 to 5.

A possible realization of a (5, 3) counter is based on 2 binary full adders and 1 binary half adder, connected in a 3-level structure. By using 4 binary full adders, and with only slight additional delay, viz the difference between one full adder and one half adder delay, a (7, 3) counter can be realized that accepts 2 additional inputs. This provides our motivation to replace the (5, 3) counter with a (7, 3) counter in order to perform more complex computations.

In the remainder of this paper, we show that by changing the (5, 3) counter into a (7, 3) counter and adding a few additional components, the bit-serial multiplier of Ienne and Viredaz [4] can be extended into bit-serial units to compute functions such as $S = VW + X$, $S = VW + X + Y$, $S = VW + X + Y + Z$, $S = VW + XY$, and ultimately $S = VW + XY + Z$. Computation of the two-term inner product, $S = VW + XY$, or inner product and accumulate, $S = VW + XY + Z$, is especially important since it is useful for matrix operations, correlation, and convolution functions. Because of minimal modifications in the overall structure of the bit-serial multiplier, all the important features listed previously for the original design carry over to these extended designs.

2. Background and Notation

We adopt the arithmetic and logic notations used by Ienne and Viredaz [4] for ease of reference and comparison. Numbers are written as capital letters, with the bits of their binary representations denoted by the corresponding lower-case letters. An index associated with a lower-case letter denotes its bit

position, starting with 0 at the least-significant bit. All multiplication operands are considered to be of length N unless otherwise noted. The final computation result is denoted by S which must be of a length $\geq 2N$ to ensure correct evaluation.

Figure 1 shows the symbols used in our logic diagrams. Symbols (a) and (b) are D flip-flops, with clock inputs omitted for simplicity. They both have a one-cycle delay and active-high synchronous-clear lines. Symbol (b) also has an active-high enable. Symbol (c) is a standard two-input multiplexer. Finally, symbol (d) is a (7, 3) counter that outputs a 3-bit binary number (output bit positions 0, 1, and 2) indicating how many of its 7 inputs are high.

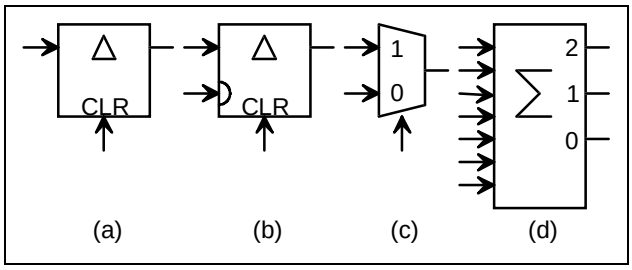

Figure 1: Circuit symbols. (a) delay element (D flip-flop) with active-high synchronous-clear, (b) same as (a) but with active-high enable, (c) 2-to-1 multiplexer, (d) (7,3) counter.

Rather than presenting a separate design for computing each of the desired and possible functions, we will only examine the case of $S = VW + XY + Z$ in detail. Other cases can be derived by pruning or simplifying the design for this most complex case.

3. Theory of Operation

The algorithm for computing $S = VW + XY + Z$ is depicted in Figure 2. In the example shown, all multiplication operands are signed 2's-complement binary numbers having $N = 4$ bits. To perform the computation correctly, these must be sign extended as suggested by Dadda [1]. The additive operand Z, however, can be a signed 2's-complement number of length $2N$. With the above assumptions, the maximum anticipated value of a positive result S is

$$S^{\max} = 2(-2^{N-1})^2 + (2^{2N-1} - 1) = 2^{2N} - 1 \qquad (1)$$

In Equation (1), the first term containing the squared negative value represents the sum of the largest possible positive products VW and XY, when each of the four operands involved is a maximal 2's-complement negative number, and the second term represents the largest possible positive value for Z. Similarly, the magnitude of the most negative result $S^{\min}$ can be computed which is slightly less than the positive bound. Thus, the result S is a 2's-complement number with at most $2N + 1$ bits and the terms to the left of the vertical line in Figure 2 are superfluous.

The boxed terms in bit positions 7 and 8 of Figure 2 can also be ignored. Consider the underlined v_3w_3 terms present in bit positions 7 and 8. These add up to form a result

$$2v_3w_3 \times 2^7 + 3v_3w_3 \times 2^8 = v_3w_3 \times 2^{10} \qquad (2)$$

The result in Equation (2) can alter S starting at bit position 10. More generally, ignoring these terms only affects bit positions

						v_3	v_3	v_3	v_3	v_3	v_3	v_2	v_1	v_0
					x	w_3	w_3	w_3	w_3	w_3	w_3	w_2	w_1	w_0
+						x_3	x_3	x_3	x_3	x_3	x_3	x_2	x_1	x_0
					x	y_3	y_3	y_3	y_3	y_3	y_3	y_2	y_1	y_0
+						z_7	z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0
+						z_7	z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0
+	← These terms can be ignored. →					v_3w_0	v_3w_0	v_3w_0	v_3w_0	v_3w_0	v_3w_0	v_2w_0	v_1w_0	v_0w_0
+						x_3y_0	x_3y_0	x_3y_0	x_3y_0	x_3y_0	x_3y_0	x_2y_0	x_1y_0	x_0y_0
+					v_3w_1	v_3w_1	v_3w_1	v_3w_1	v_3w_1	v_3w_1	v_2w_1	v_1w_1	v_0w_1	
+					x_3y_1	x_3y_1	x_3y_1	x_3y_1	x_3y_1	x_3y_1	x_2y_1	x_1y_1	x_0y_1	
+				v_3w_2	v_3w_2	v_3w_2	v_3w_2	v_3w_2	v_3w_2	v_2w_2	v_1w_2	v_0w_2		
+				x_3y_2	x_3y_2	x_3y_2	x_3y_2	x_3y_2	x_3y_2	x_2y_2	x_1y_2	x_0y_2		
+			v_3w_3	v_3w_3	v_3w_3	$\underline{v_3w_3}$	$\underline{v_3w_3}$	v_3w_3	v_2w_3	v_1w_3	v_0w_3			
+			x_3y_3	x_3y_3	x_3y_3	x_3y_3	x_3y_3	x_3y_3	x_2y_3	x_1y_3	x_0y_3			
+		v_3w_3	v_3w_3	v_3w_3	v_3w_3	$\underline{v_3w_3}$	$\underline{v_3w_3}$	v_2w_3	v_1w_3	v_0w_3				
+		x_3y_3	x_3y_3	x_3y_3	x_3y_3	x_3y_3	x_3y_3	x_2y_3	x_1y_3	x_0y_3				
+	v_3w_3	v_3w_3	v_3w_3	v_3w_3	v_3w_3	$\underline{v_3w_3}$	v_2w_3	v_1w_3	v_0w_3					
+	x_3y_3	x_3y_3	x_3y_3	x_3y_3	x_3y_3	x_3y_3	x_2y_3	x_1y_3	x_0y_3					
						S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

Figure 2: Algorithm to perform $S = VW + XY + Z$ with sign-extended two's complement numbers.

$2N + 2$ and beyond, and in no way changes our $(2N + 1)$-bit result. Similar reasoning shows that the x_3y_3 terms in bit positions 7 and 8 can be ignored.

The algorithm in Figure 2 can be implemented using a modified classic *add & shift* technique. Simple manipulation leads to the following recurrence for the computation, with $S_0 = 0$:

$$S_i = \tfrac{1}{2}\,[S_{i-1} + v_i W_{i-1} + x_i Y_{i-1} + v_i w_i 2^i + x_i y_i 2^i + z_i] \quad \text{for } i < N$$
$$S_i = \tfrac{1}{2}\,[S_{i-1} + v_{N-1} W_{N-2} + x_{N-1} Y_{N-2} + z_i] \quad \text{for } i \geq N \quad (3)$$

Besides noting that W_j and Y_j represent the values of W and Y up to bit position j (i.e., bits already received and stored in the cells), there are four main points to make with regard to Equation (3). First, the symmetric terms $v_i w_i$ and $x_i y_i$ are added only for bit positions $i < N$. Second, for the inputs V, W, X, and Y, only N–1 bits must be stored, provided that the inputs continue to supply the sign-extended values for bit positions $i \geq N$. Third, the output depends on the current inputs and previous bit values. Therefore, a new result bit is produced only after a combinational delay. And finally, the ½ term in Equation (3) implies that the least-significant result bit is shifted out and the remaining integer is all that is needed to compute further results.

4. Modular Implementation

Figure 3 shows a modular implementation of a serial arithmetic unit designed to compute the function $S = VW + XY + Z$. All signals are shown and labeled except for the clock. This is a synchronous design and it is assumed that flip-flops latch on a clock edge. With N-bit operands V, W, X, and Y, the design consists of N identical modules ($N = 4$ in Figure 2's example).

To begin a computation, "clear" must be held high for at least one cycle. After "clear" is brought low, computation begins by presenting the least significant bits of all the operands at the appropriate inputs. Also, in the same cycle that the least significant bits are presented and only for that one cycle, "token" must be set high. This token is held by a module for one cycle before it is passed onto the module below. While in possession of the token, a module computes only the symmetric term $v_j w_j + x_j y_j$, where j is the module number. This takes care of the necessary symmetric terms for $i < N$ as shown in Equation (3).

The top half of Figure 4 shows what part of the computation is performed by each module, while the bottom half indicates when each computation step is performed. For brevity, the bit-level inner product computation $v_a w_b + x_a y_b$ is represented as i_{ab}. Notice that module 0, the first module to receive a token, computes $v_0 w_0 + x_0 y_0 + z_0$ during the first cycle. Since it stores values for v_0, w_0, x_0, and y_0 during the first cycle, it will be responsible for all subsequent terms of $v_0 w_j + x_0 y_j$ and $v_j w_0 + x_j y_0$ shown in the algorithm of Figure 2. Computation proceeds in a similar manner for the remaining modules as the token is passed downward.

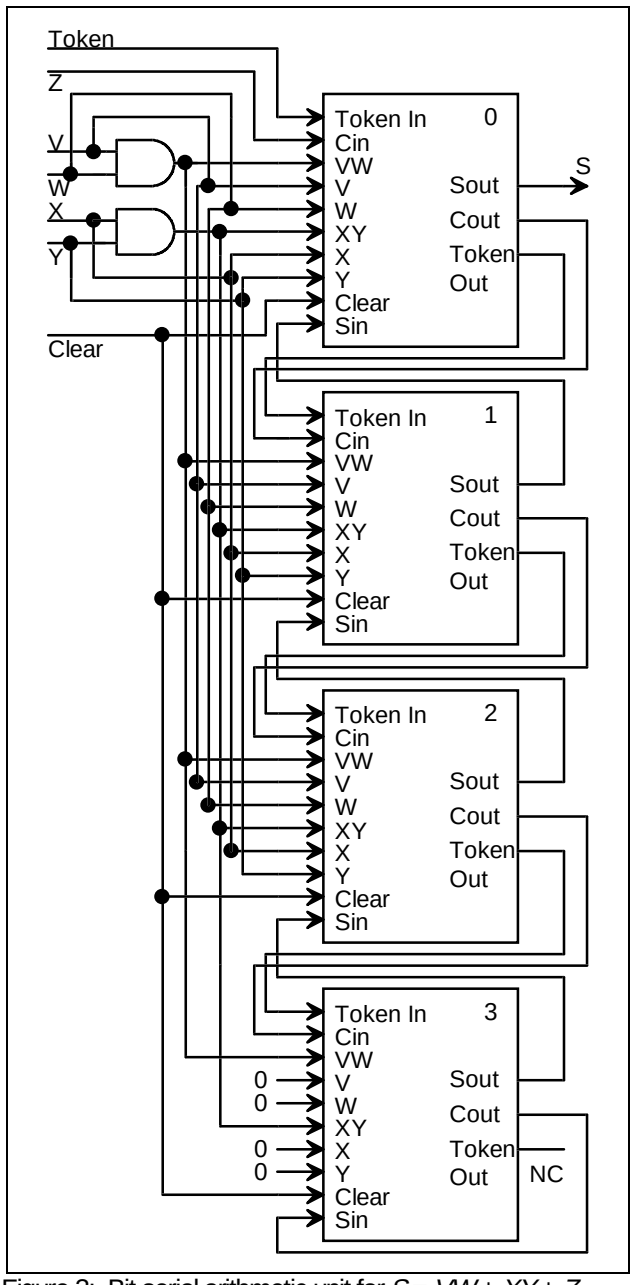

Figure 3: Bit-serial arithmetic unit for $S = VW + XY + Z$.

Note that even though Figure 4 shows modules computing some terms to the left of the vertical line separating bits positions 8 and 9, including these terms does not alter the result. These redundant computations are introduced to keep the design modular. Effects of these terms are flushed out of their respective modules by the clear signal preceding a new computation. Following an analysis similar to that of Ienne and Viredaz [4], we have shown that these terms will not corrupt proper result sign extension even if the arithmetic unit is operated beyond $2N + 1$ cycles, provided that all operands are sign extended for the entire duration of the computation.

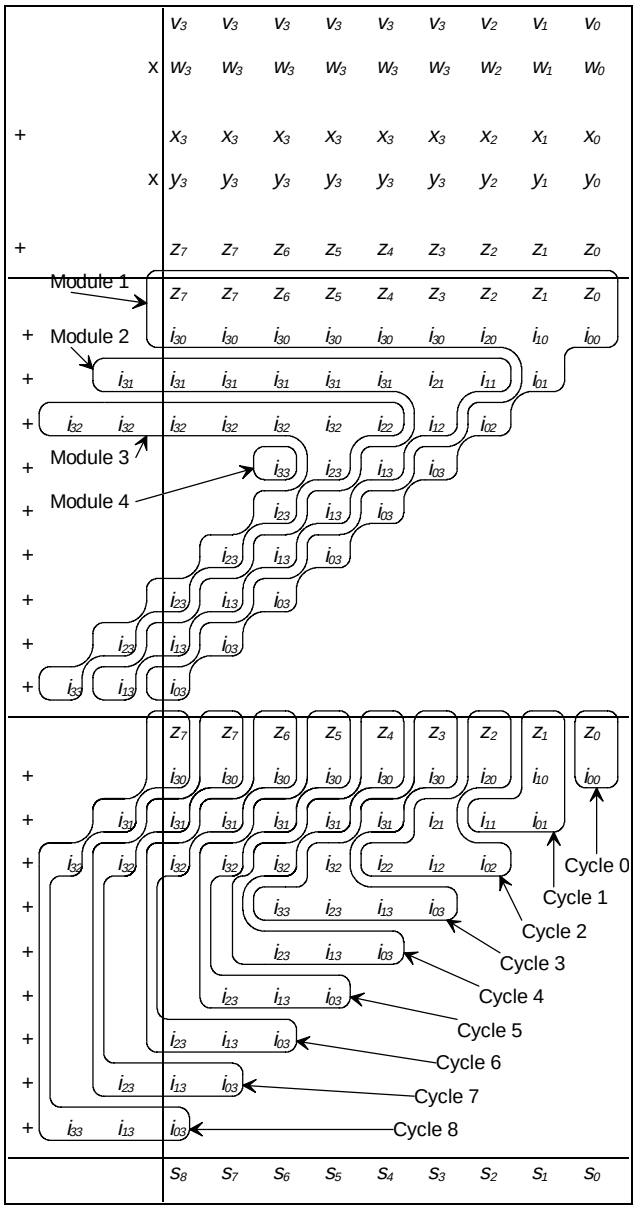

Figure 4: Module and time assignment for each bit-level inner product $i_{ab} = v_a w_b + x_a y_b$.

The final result in Figure 4 is a valid signed 2's-complement number of length $2N + 1$. This is the maximum length expected for $S = VW + XY + Z$. Unfortunately, $2N + 1$ is a rather odd length in most applications dealing with data words whose lengths are multiples of 8 or 4 bits. Typically, one knows the expected length of a result before computation. If this is the case, the user only has to compute the result up to the anticipated length. Bits beyond this length are all sign extensions. This suggests that results of the more convenient length $2N$ can be produced if the higher overflow probability is tolerable. Overflow detection would still be possible by examining the output bit at position $2N$ after each computation step.

5. Detailed Module Design

Figure 5 shows the complete implementation of a module. When the token input is high, the multiplexers present the (7, 3) counter with the product terms $v_j w_j$ and $x_j y_j$. The token signal also latches v_j, w_j, x_j, and y_j for future computations. The inverted token signal input to two AND gates is necessary to prevent any of the currently latching data from altering the result during this cycle. For the lowest order module, C_{in} carries one bit of Z.

Once the token is passed on and a new cycle i has begun, the (7, 3) counter will be presented with, in order from top to bottom input, $v_j w_i$, $v_i w_j$, $x_j y_i$, $x_i y_j$, a sum bit from module $j + 1$, a far carry from module $j - 1$, and a near carry from its own previous cycle. The carries from position j should go to positions $j + 1$ and $j + 2$, with the sum staying at position j. However, because of the multiplicative ½ term in Equation (3), everything is shifted up and each module will work on the next higher significant position during the following cycle. The number of 1s among the 7 inputs to the (7, 3) counter dictates the cell result for the current cycle. The flip-flops on the S_{in}-S_{out} path form the register used to store and shift the partial result S_i .

This design is highly modular and can easily be implemented in VLSI. Figure 3 shows a pair of AND gates producing the terms $v_j w_j$ and $x_j y_j$ for all modules. If strict modularity is desired,

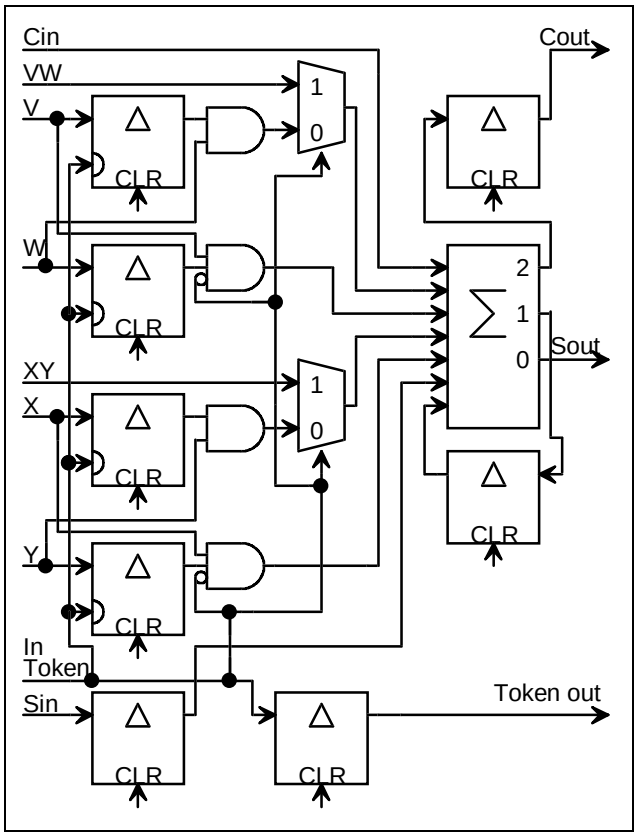

Figure 5: Bit-Slice to implement $S = VW + XY + Z$. All clears are common.

these AND gates can be replicated in each module. On the other hand, if uniformity is not an issue, then the bottom module in the series, module *N*, can be simplified. This last module does not need to store any bits for future computations. Accordingly, the *V*, *W*, *X* , and *Y* flip-flops along with their attached AND gates can be removed. Also, the multiplexers can be replaced with AND gates, with token-in as the other enabling input, and the token-out flip-flop can be removed. Finally, the (7, 3) counter can be replaced with a simpler (5, 3) counter.

The bit-slice in Figure 5 can be pruned to compute $S = VW + X + Y + Z$ by removing the flip-flops, multiplexer and gates associated with *X*, *Y*, and *XY* and then directly connecting *X* and *Y* to the (7, 3) counter. Since only the lowest-order module receives inputs for *X*, *Y*, and *Z*, the higher-order modules don't need (7, 3) counters but only (5, 3) counters. Finally, the inputs *X*, *Y*, and *Z* can be of arbitrary length, even $> 2N + 1$, as long as they are sign-extended to the maximum anticipated result length.

The computation $S = VW + XY$ is a special case of $S = VW + XY + Z$, with *Z* set to 0 at all times. If uniformity is not an issue, a (6, 3) counter could then be used for the first module in this design. Likewise, $S = VW + X + Y$ and $S = VW + X$ are special cases of $S = VW + X + Y + Z$. Again, only the first module needs as many inputs as dictated by the computed function.

6. Discussion and Conclusion

We have shown how Ienne and Viredaz's scheme for bit-serial multiplication [4] can be extended to perform $S = VW + X$, $S = VW + X + Y$, $S = VW + X + Y + Z$, $S = VW + XY$, and ultimately $S = VW + XY + Z$, using a small amount of added hardware. The extended design may require *N* modules, rather than $N - 1$ modules, but the *N*th module can be significantly simpler than the rest. The only increase in delay was due to the somewhat slower (7, 3) counter compared to a (5, 3) counter. As in the original design, results are produced without any latency cycles. Furthermore, both unsigned and signed 2's-complement numbers are accepted as long as the inputs are sign extended for the duration of the computation. Full precision outputs of arbitrary length are possible. Finally, the design is modular, allowing for easy VLSI implementation.

The critical path for the design of Figure 5 contains an AND gate, a 2-input multiplexer, and a (7, 3) counter. Compared to the original design of Ienne and Viredaz [4], this represents an increase corresponding to the difference in delay between a (7, 3) and a (5, 3) counter. Assuming 4 (2) gate levels of delay per full (half) adder and 2 per multiplexer, the delay of our extended design is 15 gate levels for an increase of about 15% over the 13 gate levels of the original design. The difference in throughputs is less pronounced since the same latch delay and clock safety margin will have to be figured in for both implementations.

Hardware complexity is increased by the difference in gate counts between a (7, 3) counter and a (5, 3) counter, one additional multiplexer, 2 AND gates, and 2 flip-flops. Counting each full (half) adder as having 9 (4) gates, a (7, 3) counter built of 4 full adders will have 36 gates compared to 22 gates for a (5, 3) counter composed of 2 full adders and 1 half adder. If additionally we take each flip-flop to have 4 gate-equivalent of complexity and each multiplexer as 3 gates, our cell complexity of 78 gates is 53% higher than that of a simple bit-serial multiplier cell at 51 gates. Here, comparison of gate counts is a fair measure of relative costs since the two designs have substantially the same interconnection patterns and wire lengths.

In many applications in signal processing and high-performance computing, the additional capabilities of double multiplication and accumulation is well worth the added complexity. If we compare the two implementations using the composite measure of cost × delay, we are paying an overhead of about 75% to do more than twice the computation.

The designs described in this paper were verified in two stages. In the prototype stage, we began by describing the basic components (latches, AND gates, counters, and multiplexers) as behavioral models in VHDL and carried out the process until complete arithmetic units were encompassed and subsequently tested in a VHDL test-bench. Once the correctness of the designs and their timing properties were established, minor adjustments were made and the full refined designs were modeled in structural VHDL using Cascade Epoch's standard cell library. The model's behavior was then verified with Mentor Graphic's QVSIM. Finally, complete VLSI circuits in a 2.0-micron process with 2 metal layers were synthesized with Epoch. Timing and area data from the synthesis confirmed our gate-level cost/performance estimates to be within 3 percentage points of actual design values (Table I).

Table I: Area and delay results

Description of the Design	Area (μm)	Delay (ns)
Design of Ref. [4] for $S = XY$	568 × 321	11.89
Our cell for $S = VW + XY + Z$	637 × 437	13.26

References

[1] Dadda, L., "On Serial-Input Multipliers for Two's Complement Numbers", *IEEE Transactions on Computers*, Vol. 38, No. 9, pp. 1341-1345, Sep. 1989.

[2] Denyer, P. and D. Renshaw, *VLSI Signal Processing: A Bit-Serial Approach*, Addison-Wesley, 1985.

[3] Ercegovac, M.D. and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*, Kluwer, Boston, 1994.

[4] Ienne, P. and M.A. Viredaz, "Bit-Serial Multipliers and Squarers", *IEEE Transactions on Computers*, Vol. 43, No. 12, pp. 1445-1450, Dec. 1994.

[5] Strader, N.R. and V.T. Rhyne, "A Canonical Bit-Sequential Multiplier", *IEEE Transactions on Computers*, Vol. C-31, No. 8, pp. 791-795, Aug. 1982.

[6] Swartzlander, E.E., "Parallel Counters", *IEEE Transactions on Computers*, Vol. C-22, No. 11, pp. 1021-1024, Nov. 1973.