Reprinted from *IEEE Transactions on Computers*, 43(12):1445-1450, December 1994.

Bit-Serial Multipliers and Squarers

Paolo Ienne and Marc A. Viredaz

Abstract

Traditional bit-serial multipliers present one or more clock cycles of data-latency. In some situations, it is desirable to obtain the output after only a combinational delay, as in serial adders and subtracters. A serial multiplier and a squarer with no latency cycles are presented here. Both accept unsigned or sign-extended two's complement numbers and produce an arbitrarily long output. They are fully modular and thus good candidates for introduction in VLSI libraries.

1 Introduction

Bit-serial arithmetic is often used in parallel systems with high connectivity to reduce the wiring down to a reasonable level. When multiplications are required, a typical choice is a serial-parallel multiplier. In this device, one factor is stored in parallel, while the other is entered serially. However, this scheme is not always possible, as, for instance, when both factors are input serially at the same time. In such cases a multiplier with two serial inputs is needed.

An example of serial input and output multiplier was presented by R. F. Lyon [1]. Its salient feature is a high throughput obtained at the expenses of a truncated output. Full-precision modular serial multipliers for unsigned numbers were introduced by H. J. Sips [2] and by N. R. Strader and V. T. Rhyne [3]. In a similar paper, R. Gnanasekaran [4] presented the first multiplication scheme for two's complement numbers. It directly takes into account the negative weight of the most significant bit in the two's complement representation. It results in a overcomplicated design which requires the knowledge of the cycle when the sign bit is presented. Moreover, a latency of one clock cycle is required before the output is produced. L. Dadda [5] independently extended the original scheme to perform a signed multiplication. Two solutions are presented. First, the sign-extension of the input is used to produce a result which is correct up to the $2N^{\text{th}}$ cycle, where the N^{th} bit represents the original sign-bit before sign-extension. Second, the negative weight of the most significant bit is directly accounted for. Anyway, a single clock-cycle latency is still present. Furthermore, the modularity issue is not directly addressed.

Finally, Rhyne and Strader [6] presented a further multiplication scheme using Booth recoding and called *Signed Canonical Bit-Sequential multiplier (SCBS multiplier)*. It is modular but far more complicated than the basic schemes derived from the *row & diagonal* algorithm. Dadda [7] pointed out the unnecessary complexity of this scheme and once again presented his multiplier based on sign-extension. The same minor limitations discussed above apply. The purely combinational latency of the SCBS is emphasized by S. G. Smith [8] as the sole reason for

This work was supported by the SPP-IF program and by the EPFL-MANTRA project.

The authors are with the Swiss Federal Institute of Technology, Mantra Centre for Neuro-Mimetic Systems, IN-J Ecublens, CH-1015 Lausanne, Switzerland.

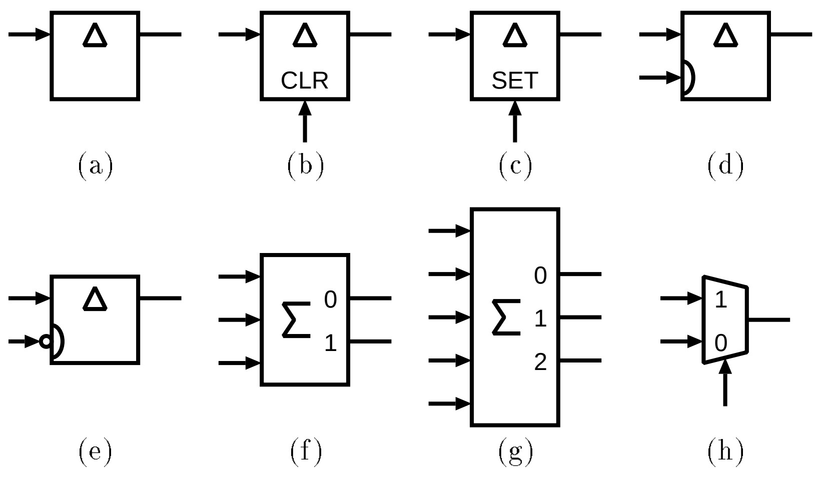


Figure 1: Symbols for the circuit elements. (a) Delay element (D flip-flop). (b) Delay element with active-high synchronous-clear. (c) Delay element with active-high synchronous-set. (d) Delay element with active-high enable. (e) Delay element with active-low enable. (f) (3, 2) counter (full-adder). (g) (5, 3) counter. (h) 2-to-1 multiplexer.

using it instead of simpler circuits. When latency clock-cycles are acceptable, Smith proposes the *flush multiplier*, a very compact design, directly derived from the serial-parallel scheme.

Section 2 presents an original scheme and implementation for serial-serial multiplication of about the same complexity as Sips' design. This multiplier offers four important features: (1) there are no latency clock-cycles between the input presentation and the result output; (2) it can be used for unsigned as well as signed two's complement numbers; (3) it produces a full-precision result (no truncated or rounded bits) which can be extended to an arbitrarily large number of bits (with no extra hardware); and (4) it is fully modular. These features are individually present in other designs but none of the cited schemes has these four characteristics at the same time.

An unsigned serial squarer has been presented by Dadda [9]. It uses the algorithm originally discussed by T. C. Chen [10]. If sign-extension is used, the complexity for signed two's complement numbers is said to be almost twice that for unsigned numbers. Other techniques are described but no really compact or modular solution is shown (the simplest implements a negative weight for the most significant bit). The design presented in section 3 has the same complexity as the Dadda's unsigned squarer but accepts also two's complement numbers and has zero clock-cycle latency.

1.1 Notation and Symbols

The multiplications considered in this correspondence will be performed on two N-bit factors, X and Y. The resulting product is a K-bit number P. Similarly, the square of the N-bit factor X is a K-bit number S. In both cases, to ensure a correct result, K must satisfy $K \geq 2N$.

All numbers are represented by capital letters, the bits of their binary representations are denoted by the corresponding lower-case letter and an index.

Figure 1 shows the logic symbols used in this correspondence. The symbols (a) to (e) represent one-cycle delay elements, that is, D flip-flops whose clock inputs have been omitted

					x3	x2	x1	x0
					y3	y2	y1	y0
						x2y0	x1y0	x0y0
-					x3y0			
+					x2y1	x1y1	x0y1	
-				x3y1				
+				x2y2	x1y2	x0y2		
-			x3y2					
-			x2y3	x1y3	x0y3			
+		x3y3						
	p7	p6	p5	p4	p3	p2	p1	p0

Figure 2: Multiplication algorithm for signed two's complement numbers.

for simplicity. Symbol (b) has an active-high synchronous-clear signal, and symbol (c) an active-high synchronous-set signal. Symbols (d) and (e) have respectively an active-high and active-low enable signal. The enable signal must be active for a clock transition to load a value into the element. It is obviously possible to combine clear/set and enable signals on the same delay element. In this case the clear/set signal has higher precedence.

The name *(n, k) counter* will be used to designate a device with n inputs and whose k outputs code a binary word representing the number of active input signals. An alternative view of this device is that of an adder of n one-bit words producing a k-bit number. The number of outputs is obviously equal to $k = \lceil \log_2 (n+1) \rceil$. Some implementations of this class of devices have been discussed by Swartzlander [11]. Symbols (f) and (g) in figure 1 represent respectively a (3, 2) counter and a (5, 3) counter. Finally, symbol (h) is a two input multiplexer.

1.2 Basic Algorithms

Figure 2 shows the multiplication algorithm resulting from a straightforward application of the two's complement integer definition. The $N-1$ least significant bits of each sub-product must first be added, and then the last bit is subtracted, except for the last sub-product, for which additions are replaced by subtractions and vice versa. The complex control required for this algorithm makes it a poor candidate for hardware implementation.

A possible simplification is to sign-extend the sub-products to the desired length K of the final result, and to perform "normal" additions and subtraction, keeping only the first K bits. The resulting algorithm is shown in figure 3, for $K = 9$ bits. This operation can alternatively be viewed as sign-extending the coefficient X to K bits.

However, the algorithm of figure 3 has the disadvantage that the last sub-product must be subtracted while all the previous ones are to be added. This can be overcome by also sign-extending the data Y to K bits and then by performing K additions. Again, only the first K bits of the result should be kept. This algorithm, shown in figure 4, is the one used most

				x3	x3	x3	x3	x3	x3	x2	x1	x0
									y3	y2	y1	y0
				x3y0	x3y0	x3y0	x3y0	x3y0	x3y0	x2y0	x1y0	x0y0
+			x3y1	x3y1	x3y1	x3y1	x3y1	x3y1	x2y1	x1y1	x0y1	
+		x3y2	x3y2	x3y2	x3y2	x3y2	x3y2	x2y2	x1y2	x0y2		
-	x3y3	x3y3	x3y3	x3y3	x3y3	x3y3	x2y3	x1y3	x0y3			
				p7	p7	p6	p5	p4	p3	p2	p1	p0

Figure 3: Multiplication algorithm for signed two's complement numbers with X sign-extended.

often. Both algorithms shown in figures 3 and 4 produce a sign-extended result.

2 Serial-Serial Multiplier

Using the algorithm of figure 4 on sign-extended inputs, the number of partial terms to add grows linearly with the bit position. Most of the terms added to produce all bits of weight above 2^{2N-1} inclusive are due to the sign extension, and are in the form $x_{N-1} \cdot y_{N-1}$. It can be shown that the regularity of these terms makes it possible to ignore them without producing a difference in the output.

Proceeding in a way similar to Strader and Rhyne [3], the partial product after the presentation of the bit number i can be expressed recursively as

$$P_i = X_i \cdot Y_i = P_{i-1} + x_i \cdot Y_{i-1} \cdot 2^i + y_i \cdot X_{i-1} \cdot 2^i + x_i \cdot y_i \cdot 2^{2i}; \tag{1}$$

where X_i and Y_i represent the value of the operands considering only bits from 0 to i, that is, $X_i = X \bmod 2^{i+1}$. The initial values are $P_{-1} = X_{-1} = Y_{-1} = 0$. If the sign extension is performed until the beginning of next multiplication, only the first $N-1$ bits of the operands need to be stored (the sign bits remaining available on the inputs). This may suggest a scheme where, on cycles $i \geq N$, partial operands extend only to bit $N-2$ and the symmetric term is no longer added:

$$\overline{P}_i = \overline{P}_{i-1} + x_{N-1} \cdot Y_{N-2} \cdot 2^i + y_{N-1} \cdot X_{N-2} \cdot 2^i \qquad \text{for } i \geq N, \tag{2}$$

with $\overline{P}_i = P_i$ for $i < N$. The error incurred at cycle $j \geq N$ by using equation (2) instead of equation (1) is:

$$\begin{aligned}(P_j - P_{j-1}) - \left(\overline{P}_j - \overline{P}_{j-1}\right) &= x_j \cdot Y_{j-1} \cdot 2^j - x_{N-1} \cdot Y_{N-2} \cdot 2^j \\ &\quad + y_j \cdot X_{j-1} \cdot 2^j - y_{N-1} \cdot X_{N-2} \cdot 2^j + x_j \cdot y_j \cdot 2^{2j} \\ &= x_{N-1} \cdot y_{N-1} \cdot \left(2^{2j} + 2 \cdot 2^j \cdot \sum_{k=N-1}^{j-1} 2^k\right). \end{aligned} \tag{3}$$

The last expression is obtained considering that $x_j = x_{N-1}$, for $j \geq N$, which also implies $Y_{j-1} = Y_{N-2} + y_{N-1} \sum_{k=N-1}^{j-1} 2^k$. Similar relations apply to y_j and X_{j-1}. Therefore, after cycle

									x3	x3	x3	x3	x3	x3	x2	x1	x0
									y3	y3	y3	y3	y3	y3	y2	y1	y0
									x3y0	x3y0	x3y0	x3y0	x3y0	x3y0	x2y0	x1y0	x0y0
+								x3y1	x3y1	x3y1	x3y1	x3y1	x3y1	x2y1	x1y1	x0y1	
+							x3y2	x3y2	x3y2	x3y2	x3y2	x3y2	x2y2	x1y2	x0y2		
+						x3y3	x3y3	x3y3	x3y3	x3y3	x3y3	x2y3	x1y3	x0y3			
+					x3y3	x3y3	x3y3	x3y3	x3y3	x3y3	x2y3	x1y3	x0y3				
+				x3y3	x3y3	x3y3	x3y3	x3y3	x3y3	x2y3	x1y3	x0y3					
+			x3y3	x3y3	x3y3	x3y3	x3y3	x3y3	x2y3	x1y3	x0y3						
+		x3y3	x3y3	x3y3	x3y3	x3y3	x3y3	x2y3	x1y3	x0y3							
+	x3y3	x3y3	x3y3	x3y3	x3y3	x3y3	x2y3	x1y3	x0y3								
									p7	p7	p6	p5	p4	p3	p2	p1	p0

Figure 4: Multiplication algorithm for signed two's complement numbers with both X and Y sign-extended.

i, the total error amounts to

$$
\begin{aligned}
E_i &= x_{N-1} \cdot y_{N-1} \cdot \sum_{j=N}^{i} \left(2^{2j} + 2 \cdot 2^j \cdot \sum_{k=N-1}^{j-1} 2^k \right) = x_{N-1} \cdot y_{N-1} \cdot \sum_{j=N}^{i} \left(2^{2j} + \sum_{k=N}^{j} 2^{j+k} \right) \\
&= x_{N-1} \cdot y_{N-1} \cdot \sum_{j=N}^{i} \left(2^{2j} + \sum_{k=j}^{i} 2^{j+k} \right) = x_{N-1} \cdot y_{N-1} \cdot 2^{i+1} \cdot \sum_{j=N}^{i} 2^j. \qquad (4)
\end{aligned}
$$

It is clear from equation (4) that $E_i \bmod 2^{i+1} = 0, \forall i$. This implies that the missing part of the partial sum never affects a serial output multiplier which—at cycle $i = K - 1$, that is, at the end of the multiplication—has produced only the least significant K bits of output.

Moreover, when unsigned numbers are entered zero-extended, E_i is identically zero. Therefore the expression for $\overline{P}_i$ can be conveniently used to build a multiplier performing equally well on unsigned zero-extended and on sign-extended operands.

The classic *add & shift* technique can be conveniently used introducing

$$
\begin{aligned}
Q_i &= \frac{\overline{P}_i}{2^{i+1}} = \frac{1}{2} \cdot (Q_{i-1} + x_i \cdot Y_{i-1} + y_i \cdot X_{i-1} + 2^i \cdot x_i \cdot y_i) \qquad \text{for } i < N; \qquad (5) \\
Q_i &= \frac{\overline{P}_i}{2^{i+1}} = \frac{1}{2} \cdot (Q_{i-1} + x_{N-1} \cdot Y_{N-2} + y_{N-1} \cdot X_{N-2}) \qquad \text{for } i \geq N. \qquad (6)
\end{aligned}
$$

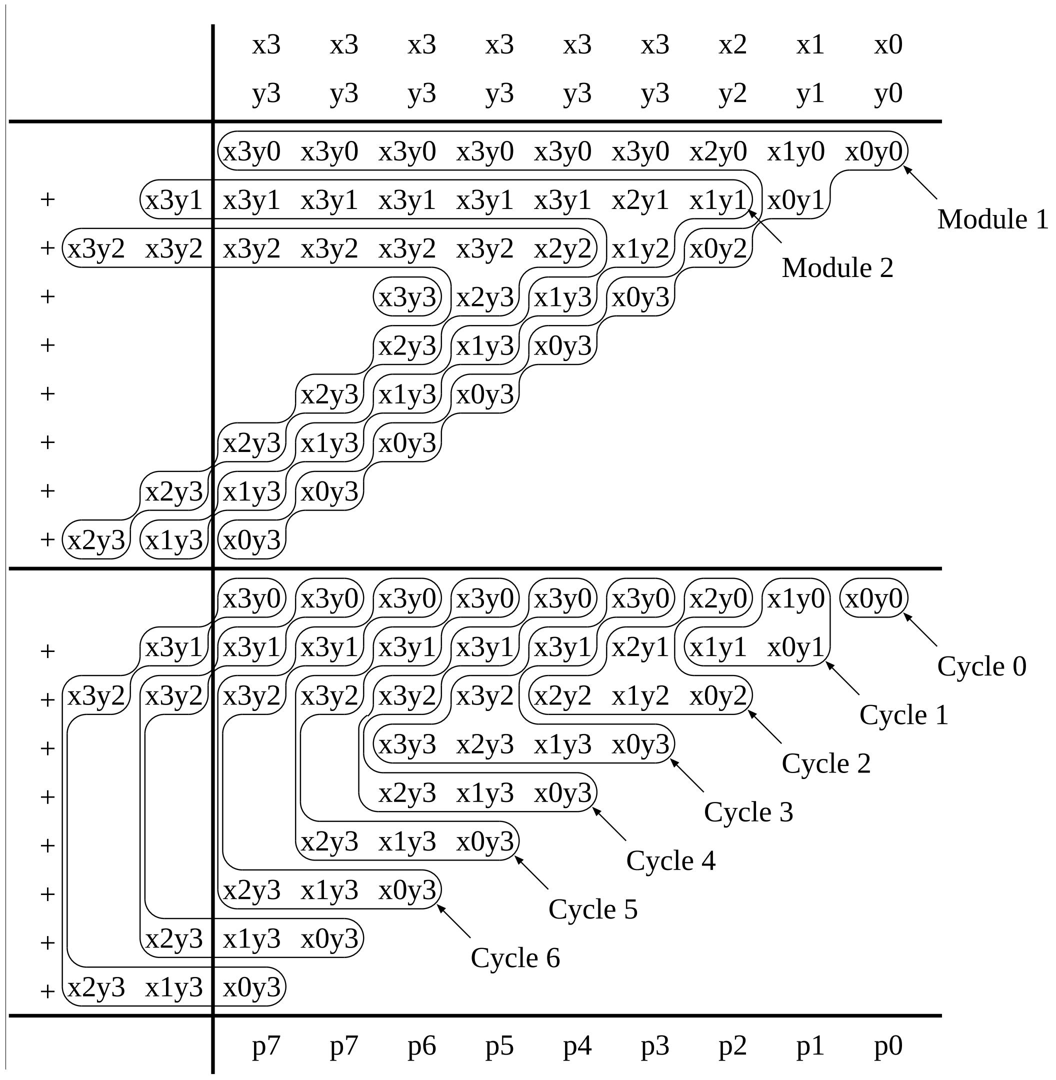
Equation (5) shows that each partial product can be produced with a combinational delay by using the stored values of the operands only up to the previous input bits (X_{i-1} and Y_{i-1}) and the current inputs (x_i and y_i). The symmetric term $x_i \cdot y_i$ should be added on-line to bit number i. The 1/2 term accounts for the shift and the fractional part represents the ready part of the product which has been shifted out. Before shifting, the value added to the partial product in each cycle is an integer value. Thus, if the remainder of the division is extracted from the multiplier at each iteration, an equivalent system can be built up that stores only the integer part of the result:

$$
\overline{Q}_i = \left\lfloor \frac{1}{2} \cdot \left(\overline{Q}_{i-1} + x_i \cdot Y_{i-1} + y_i \cdot X_{i-1} + 2^i \cdot x_i \cdot y_i \right) \right\rfloor \qquad \text{for } i < N. \qquad (7)
$$

A similar expression for $i \geq N$ can be derived from equation (6).

A better understanding of the situation can be obtained by comparing an example of this algorithm with the basic algorithm for sign-extended operands shown in figure 4. Figure 5 presents an example multiplication of two 4-bit operands, generating a 9-bit output. The terms present in figure 4 and missing in figure 5 correspond to those in the last sum of equation (4).

A single bit-slice implementing directly the algorithm of equation (7) is shown in figure 6. Owing to its structure, this multiplier requires both operands to be of the same size. Should this not be the case, the shorter one is to be extended to the size of the longer one. $N - 1$ such slices are necessary to multiply two N-bit numbers (thus allowing storage of X_{N-2} and Y_{N-2}). They should be connected as shown in figure 7. Perfect modularity, at the expense of some redundant hardware, can be obtained by replicating in every module the AND gate of figure 7 and by replacing the multiplexer with an N^{th} module having signal Pin at zero. Both operands are serially entered, least significant bit first, during the first N clock cycles. As mentioned above, unsigned numbers should be extended with zeroes during the $K - N$ remaining cycles (with $K \geq 2N$ to ensure a correct result) and two's complement numbers should be sign-extended. The FirstBit signal should be set high during the first cycle; subsequently held low.



x3 x3 x3 x3 x3 x3 x2 x1 x0
y3 y3 y3 y3 y3 y3 y2 y1 y0
x3y0 x3y0 x3y0 x3y0 x3y0 x3y0 x2y0 x1y0 x0y0
+ x3y1 x3y1 x3y1 x3y1 x3y1 x3y1 x2y1 x1y1 x0y1
+ x3y2 x3y2 x3y2 x3y2 x3y2 x3y2 x2y2 x1y2 x0y2
+ x3y3 x2y3 x1y3 x0y3
+ x2y3 x1y3 x0y3
+ x2y3 x1y3 x0y3
+ x2y3 x1y3 x0y3
+ x2y3 x1y3 x0y3
+ x2y3 x1y3 x0y3
Module 1
Module 2
x3y0 x3y0 x3y0 x3y0 x3y0 x3y0 x2y0 x1y0 x0y0
+ x3y1 x3y1 x3y1 x3y1 x3y1 x3y1 x2y1 x1y1 x0y1
+ x3y2 x3y2 x3y2 x3y2 x3y2 x3y2 x2y2 x1y2 x0y2
+ x3y3 x2y3 x1y3 x0y3
+ x2y3 x1y3 x0y3
+ x2y3 x1y3 x0y3
+ x2y3 x1y3 x0y3
+ x2y3 x1y3 x0y3
+ x2y3 x1y3 x0y3
Cycle 0
Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5
Cycle 6
p7 p7 p6 p5 p4 p3 p2 p1 p0

Figure 5: Multiplication algorithm implemented by the serial-serial multiplier for signed two's complement numbers.

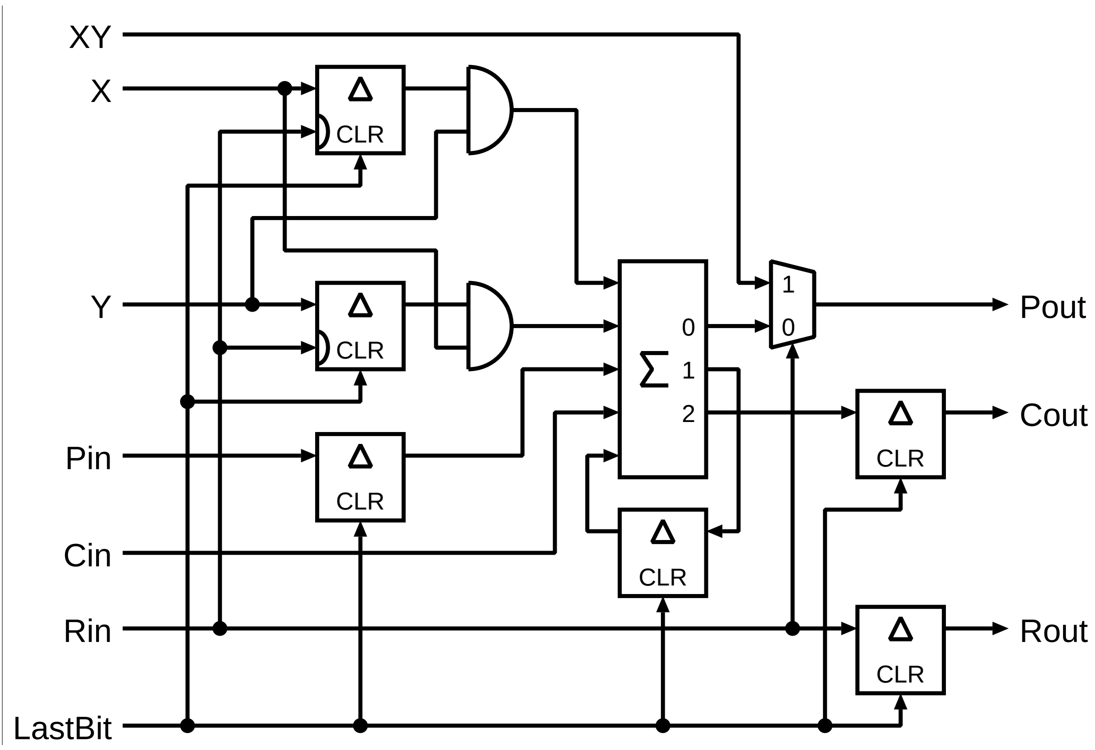


Figure 6: One-bit serial-serial multiplier slice.

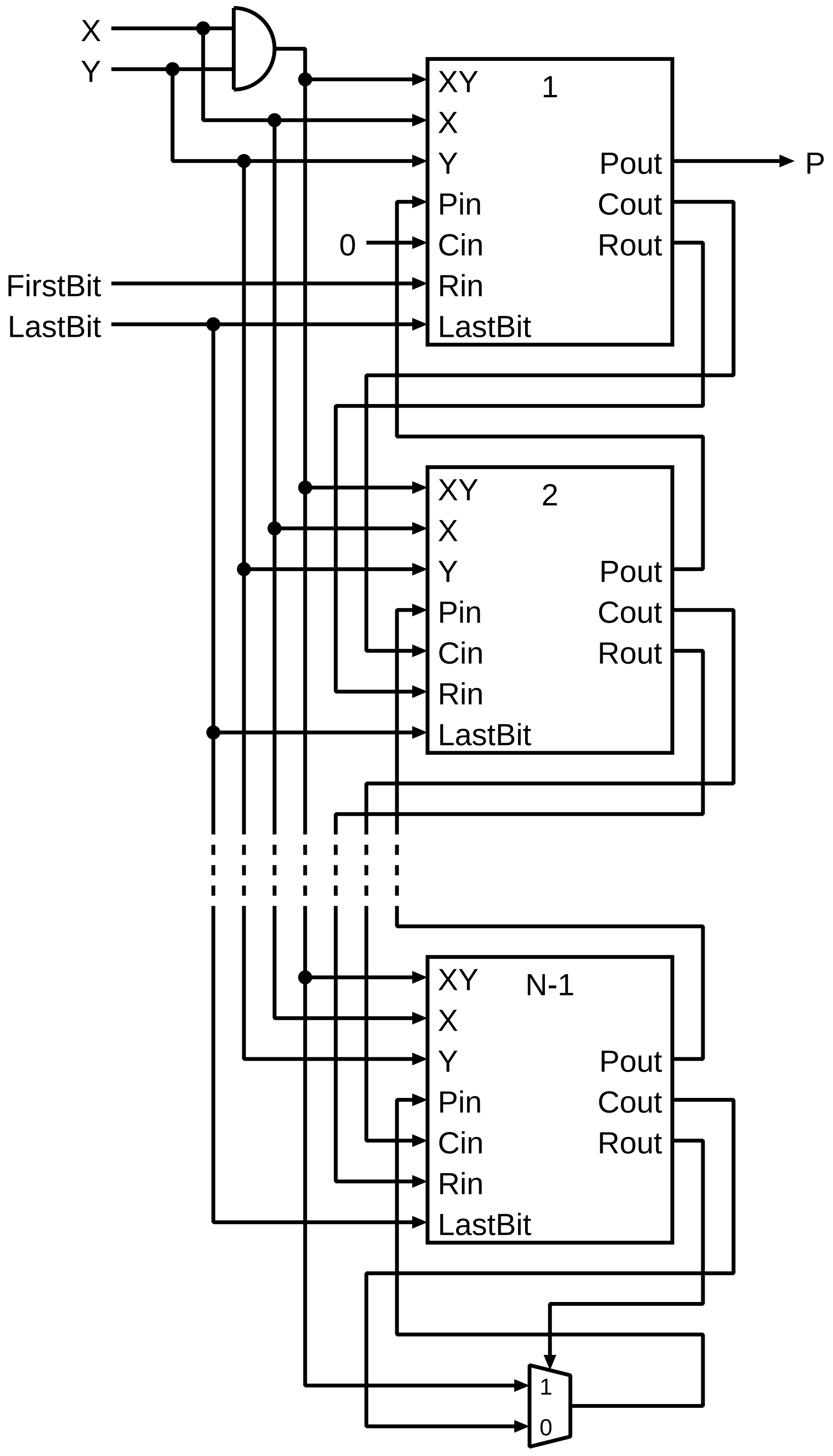


Figure 7: Serial-serial multiplier for unsigned and signed two's complement numbers.

Similarly, the LastBit signal should be activated only during the last cycle of a multiplication or, more precisely, during the cycle preceding a new multiplication. The output is serially available starting with the first cycle. The values outlined by the sets in the upper portion of figure 5 correspond to the terms added by each module of the multiplier; while the elements of the sets in the lower portion of the figure are added to the partial sum in the same cycle.

Usually, two new terms are added in each bit-slice at every cycle. This suggests the use of a (5, 3) counter with a *far carry* (weight $\times 4$) back-propagated to the module that precedes the generating one (delay element on the output Cout). Another delay element is provided to store the local carry. The output multiplexer of the $(i+1)^{\text{th}}$ slice is used to add the symmetric term $x_i \cdot y_i$ to the appropriate bit of the partial sum. The delay elements with enable signals progressively store the operands (X_i and Y_i), and are used to produce the two new terms added to the partial sum ($x_i \cdot Y_{i-1}$ and $y_i \cdot X_{i-1}$). Finally the delay elements on the path Pin-Pout constitute the register to store and shift the partial sum $\overline{Q}_i$.

The shift register produced by chaining the signals Rin and Rout of the various stages activate successive modules of the multiplier; this allows the addition of the symmetric term and enables the memorization of the input bits. It can be replaced by a binary counter followed by a decoder. The counter should be synchronously reset before the first bit is entered. This alternative scheme is valid also for the proposed squarer (see section 3). Essentially, it trades modularity for a possible small hardware reduction. Otherwise the two approaches are equivalent.

The module shown in figure 6 has a critical path composed of an AND gate, a (5, 3) counter, and a 2-input multiplexer. Compared to other designs (e.g., Dadda [7]), this design increases the critical path delay by the propagation time through the multiplexer.

3 Serial Squarer

There are many systems where a serial unit should produce the square of its input; for instance, when computing the Euclidean distance between two vectors. If two different numbers are not required to be multiplied in the same unit, then it would be a waste of hardware to implement such a unit with the serial-serial multiplier presented in section 2. The fact that the two operands are identical facilitates in the simplification of the multiplication algorithm.

Equation (7), for $i < N$, now becomes

$$\overline{Q}_i = \left\lfloor \frac{1}{2} \cdot \left(\overline{Q}_{i-1} + 2 \cdot x_i \cdot X_{i-1} + 2^i \cdot x_i \right) \right\rfloor \qquad \text{for } i < N. \tag{8}$$

An expression for $i \geq N$ can be obtained by limiting the register storage and avoiding the addition of the term x_i. The remarks on the irrelevance of the incurred error E_i hold equally true.

Figure 8 shows graphically the algorithm used to square an unsigned number of $N = 4$ bits. The upper part of the figure shows the conventional unsigned multiplication algorithm with both operands being identical. The lower part presents the simplified squaring algorithm. Similarly, figure 9 presents the algorithm used to square signed two's complement numbers. In this example, the input has $N = 4$ bits and the output $K = 9$ bits.

A single bit-slice of the proposed serial squarer is shown in figure 10. $N - 1$ such slices—connected as shown in figure 11—are used to square an N-bit operand. For improved modularity, as in the case of the multiplier, the AND gate can be replaced with an N^{th} module having signal Sin at zero. The operand is serially entered, least significant bit first, during the N first clock cycles, and is extended with zeroes (unsigned number) or sign-extended (two's complement

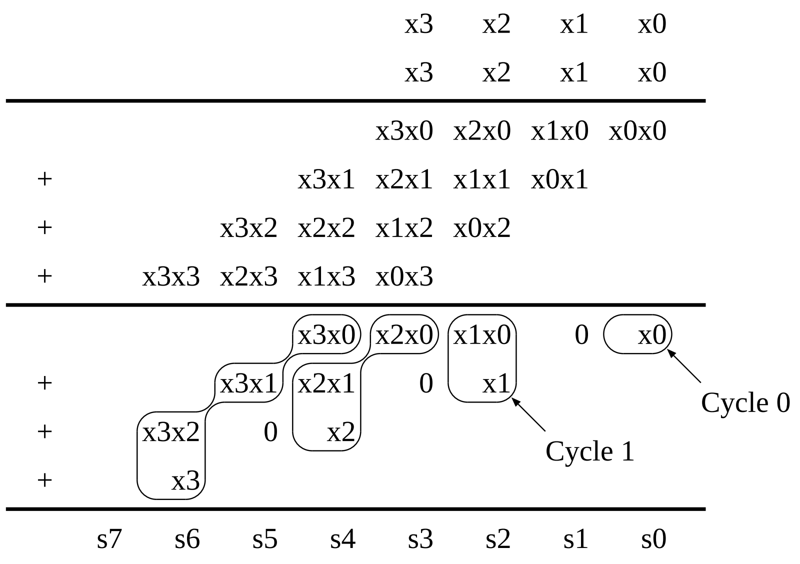


Figure 8: Squaring algorithm for unsigned numbers.

number) during the $K - N$ cycles (with $K \geq 2N$ to ensure a correct result). As with the serial-serial multiplier, the FirstBit and the LastBit signals should be set high during the first and the last cycle respectively; otherwise they are set low. The output is serially available with no latency. The same hardware can be used for both unsigned and two's complement numbers.

The values outlined in the sets in figures 8 and 9 are added to the partial result at successive iterations, the terms on the n^{th} line being produced by module n. The terms x_i are added through the output multiplexers. The operand is progressively stored (X_i) in the register made up of the delay elements with enable. The output of the AND gate is added to the partial sum, stored in the shift register composed of the delay elements on the path Sin-Sout. Comparing figure 10 with figure 6, it may be noted that the delay element holding $\overline{Q}_i$ is here moved after the adder. This doubles the weight of the added term (output of the AND gate) and accounts for the 2 in front of the term $x_i \cdot X_{i-1}$ found in equation (8).

4 Conclusions

As detailed in the introduction, the multiplication scheme discussed above presents some similarities with other designs, but the proposed implementation is the most compact one. An interesting feature, extremely important for VLSI designs, is its high degree of modularity. The delay between an input bit presentation and the corresponding output bit has been reduced to the delay of a combinational circuit, whose critical path contains only a (5, 3) counter and a few gates. Furthermore, a novel and important feature has been added: the result remains correct (i.e., the sign is extended) as long as the inputs are sign-extended, hence allowing the production of an arbitrarily long output. In some cases, this can be a key feature. Supposing that a sum of products is being computed with all bit-serial components and that the partial-sum size is larger than that of the single products, no special circuitry has to be added between the multiplier and the serial adder, provided that the operands are appropriately sign-extended.

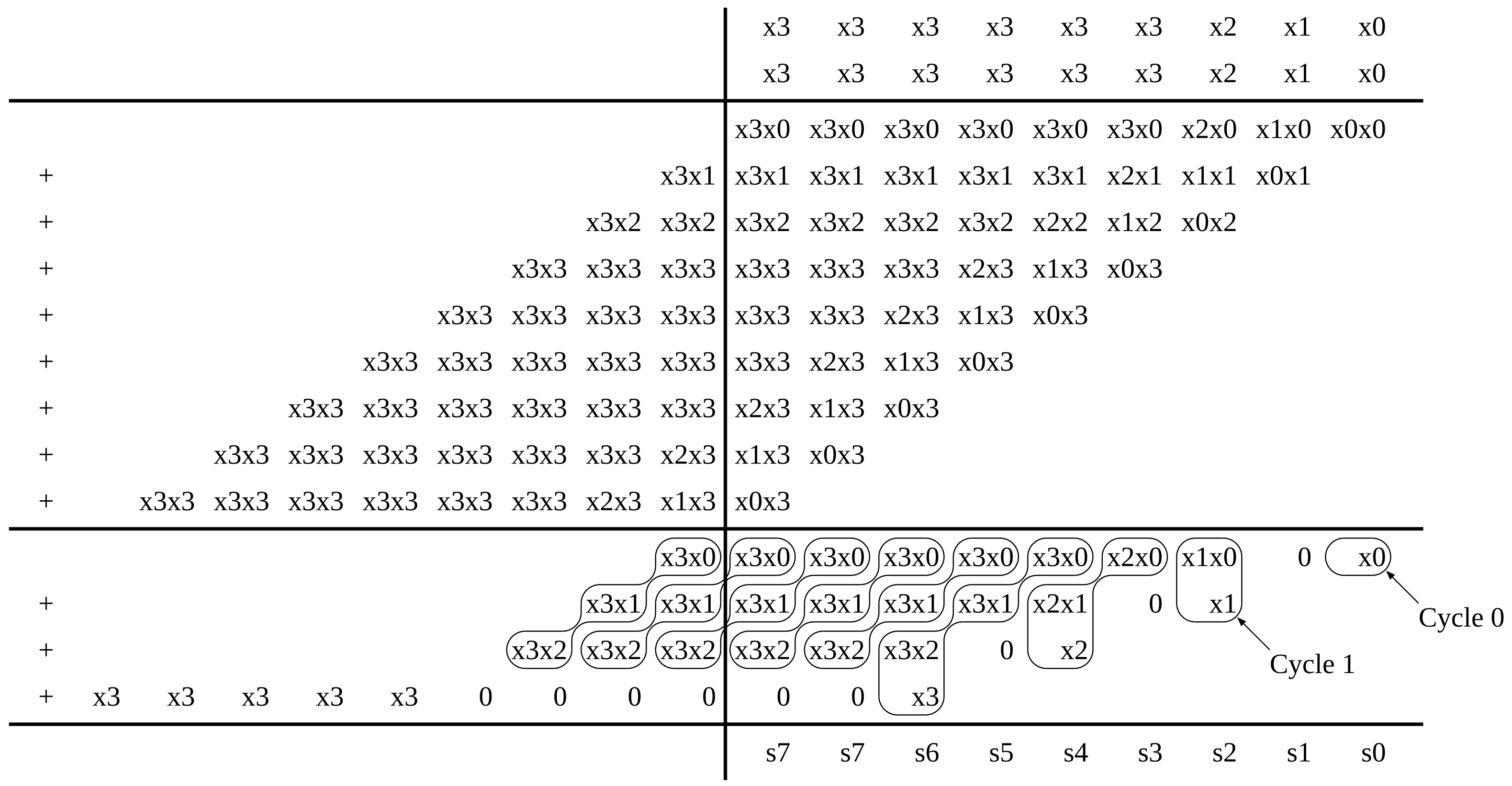

x3 x3 x3 x3 x3 x3 x2 x1 x0
x3 x3 x3 x3 x3 x3 x2 x1 x0
x3x0 x3x0 x3x0 x3x0 x3x0 x3x0 x2x0 x1x0 x0x0
+ x3x1 x3x1 x3x1 x3x1 x3x1 x3x1 x2x1 x1x1 x0x1
+ x3x2 x3x2 x3x2 x3x2 x3x2 x3x2 x2x2 x1x2 x0x2
+ x3x3 x3x3 x3x3 x3x3 x3x3 x3x3 x2x3 x1x3 x0x3
+ x3x3 x3x3 x3x3 x3x3 x3x3 x3x3 x2x3 x1x3 x0x3
+ x3x3 x3x3 x3x3 x3x3 x3x3 x3x3 x2x3 x1x3 x0x3
+ x3x3 x3x3 x3x3 x3x3 x3x3 x3x3 x2x3 x1x3 x0x3
+ x3x3 x3x3 x3x3 x3x3 x3x3 x3x3 x3x3 x2x3 x1x3 x0x3
+ x3x3 x3x3 x3x3 x3x3 x3x3 x3x3 x2x3 x1x3 x0x3
x3x0 x3x0 x3x0 x3x0 x3x0 x3x0 x2x0 x1x0 0 x0
+ x3x1 x3x1 x3x1 x3x1 x3x1 x3x1 x2x1 0 x1
+ x3x2 x3x2 x3x2 x3x2 x3x2 x3x2 0 x2
+ x3 x3 x3 x3 x3 0 0 0 0 0 0 x3
s7 s7 s6 s5 s4 s3 s2 s1 s0
Cycle 0
Cycle 1

Figure 9: Squaring algorithm for signed two's complement numbers.

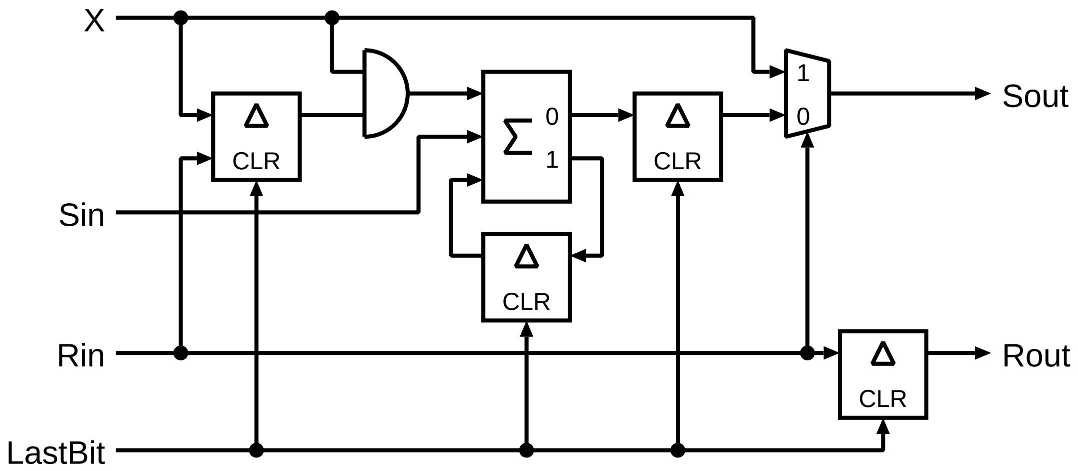


Figure 10: One-bit serial squarer slice.

The proposed squarer, an original implementation of the algorithm presented by Chen [10], also allows the use of signed two's complement numbers. The implementation shares the same advantages as the companion multiplier: it is compact, modular, and has no latency. It is moreover much simpler than the signed squarer described by Dadda [9].

The described serial-serial multiplier and serial squarer have been designed in VLSI and simulated using the Compass ASIC development environment [12]. The multiplier has also been included in the GENES IV circuit [13], which has been manufactured in CMOS 1.0 μm technology and successfully tested.

Acknowledgments

The authors wish to thank Prof. Jean-Daniel Nicoud for his constant support. They also gratefully acknowledge the constructive criticism of the referees and extend their appreciation for having stimulated the development of formal proofs of the algorithms.

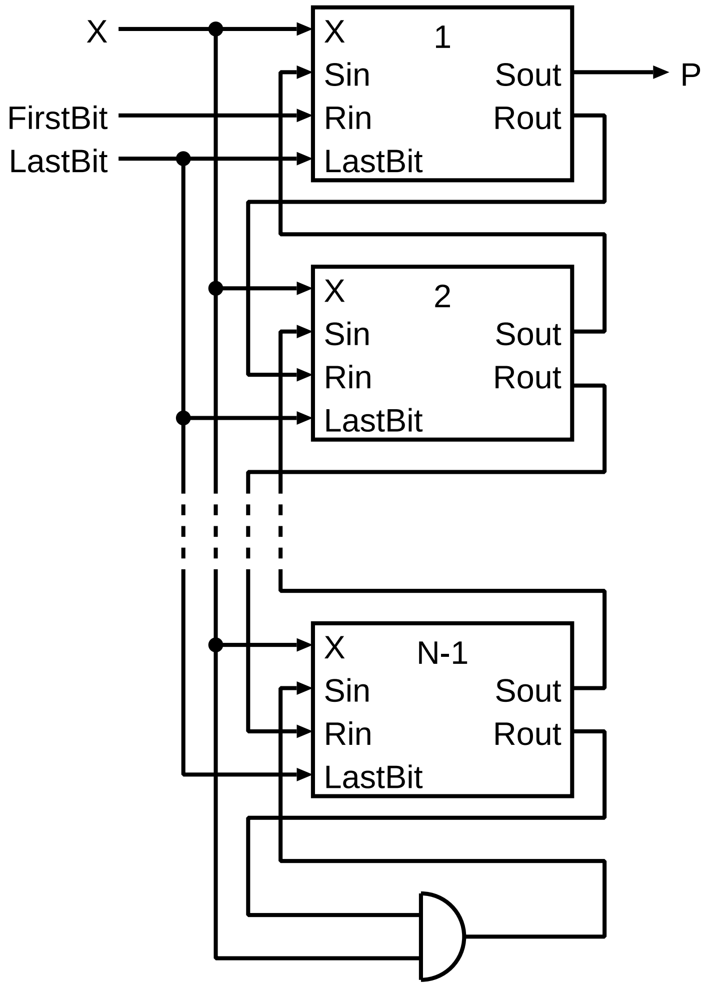


Figure 11: Serial squarer for unsigned and signed two's complement numbers.

References

[1] R. F. Lyon. Two's complement pipeline multipliers. *IEEE Transactions on Communications*, COM-24(4):418–25, April 1976.

[2] H. J. Sips. Comments on "An O(n) parallel multiplier with bit-sequential input and output". *IEEE Transactions on Computers*, C-31(4):325–27, April 1982.

[3] Noel R. Strader and V. Thomas Rhyne. A canonical bit-sequential multiplier. *IEEE Transactions on Computers*, C-31(8):791–95, August 1982.

[4] R. Gnanasekaran. On a bit-serial input and bit-serial output multiplier. *IEEE Transactions on Computers*, C-32(9):878–80, September 1983.

[5] Luigi Dadda. Fast multipliers for two's-complement numbers in serial form. In *IEEE 7th Symposium on Computer Arithmetic*, pages 57–63, 1985.

[6] Tom Rhyne and Noel R. Strader, II. A signed bit-sequential multiplier. *IEEE Transactions on Computers*, C-35(10):896–901, October 1986.

[7] Luigi Dadda. On serial-input multipliers for two's complement numbers. *IEEE Transactions on Computers*, C-38(9):1341–45, September 1989.

[8] S. G. Smith. Comments on "A signed bit-sequential multiplier". *IEEE Transactions on Computers*, C-38(9):1328–30, September 1989.

[9] Luigi Dadda. Squarers for binary numbers in serial form. In *IEEE 7th Symposium on Computer Arithmetic*, pages 173–80, 1985.

[10] Tien Chi Chen. A binary multiplication scheme based on squaring. *IEEE Transactions on Computers*, C-20(6):678–80, June 1971.

[11] Earl E. Swartzlander, Jr. Parallel counters. *IEEE Transactions on Computers*, C-22(11):1021–24, November 1973.

[12] Paolo Ienne and Marc A. Viredaz. Bit-serial input and output multipliers and squarers. Technical Report 92/7, École Polytechnique Fédérale de Lausanne - DI, October 1992.

[13] Paolo Ienne and Marc A. Viredaz. GENES IV: A bit-serial processing element for a multi-model neural-network accelerator. In *Proceedings of the International Conference on Application Specific Array Processors*, Venezia, October 1993.