Efficient Implementation of a Complex ±1 Multiplier

Boris D. Andreev, Eby G. Friedman, and Edward L. Titlebaum

Department of Electrical and Computer Engineering
University of Rochester
Rochester, New York 14627

{bandreev, friedman, tbaum} @ece.rochester.edu

ABSTRACT

A complex ±1 multiplier is an integral element in modern CDMA communication systems, specifically as a pseudonoise code scrambler/descrambler. Therefore, an efficient implementation is essential to reduce the critical path delay, power, and area of wireless receivers. A new architecture is proposed to achieve this complex multiplier function. Tradeoffs and design solutions as well as the interface with subsequent arithmetic circuits are discussed. Simulations exhibit a significant speed improvement as compared to alternative architectures. These results are also applicable to other arithmetic circuits.

Categories and Subject Descriptors

B.2.4 [**Arithmetic and Logic Structures**] High-Speed Arithmetic: *Algorithms*

B.7.1 [**Integrated circuits**] Types and Design Styles: *Algorithms implemented in hardware, VLSI*

I.1.1 [**Symbolic and Algebraic Manipulation**] Expressions and Their Representation: *Representations, Simplification of expressions*

General Terms

Design, Algorithms

Keywords

VLSI, CDMA, PN code, scrambler, redundant arithmetic

1. INTRODUCTION

Modern CDMA cellular systems employ spread spectrum technology to provide multiuser access. In addition to the spreading operation, an integral part of the transceiver is the scrambling operation, which involves the multiplication of the chip sequence with a pseudonoise (PN) code in order to distinguish signals from asynchronous users. In the Third Generation Partnership Project (3GPP) wireless standard [1], the scrambling code is complex, therefore, a corresponding complex multiplication operation is required in both the transmitter and the receiver. The standard transmission scheme is shown in Fig. 1, where after spreading and scaling operations, a complex signal is formed and multiplied by a complex PN code. Since the components of the complex PN code take binary values in the set {−1, +1}, the scrambling multiplier should be optimized to reduce the critical path delay, power, and area of wireless transceivers. Since no such circuits have been reported to date, both conventional and novel architectural solutions are presented here.

The bit-width of the input and output operands is among the primary characteristics of any arithmetic circuit. A sufficient fixed-point number representation is dependent on both the parameters of the cellular system and the particular detection algorithms. As described in [2] and [3], the implementation of certain multiuser algorithms with 8-bit to 16-bit representation of the received signal suffers negligible performance degradation as compared to a system implemented with floating point precision. Therefore, arithmetic circuits on the order of 8 to 16 bits are discussed in this paper with particular attention focused on an 8-bit representation.

These results are not limited to the application of a complex ±1 multiplier as a scrambler in wireless transceivers. The ideas and relations may also be used as a basis for the efficient implementation of other arithmetic circuits.

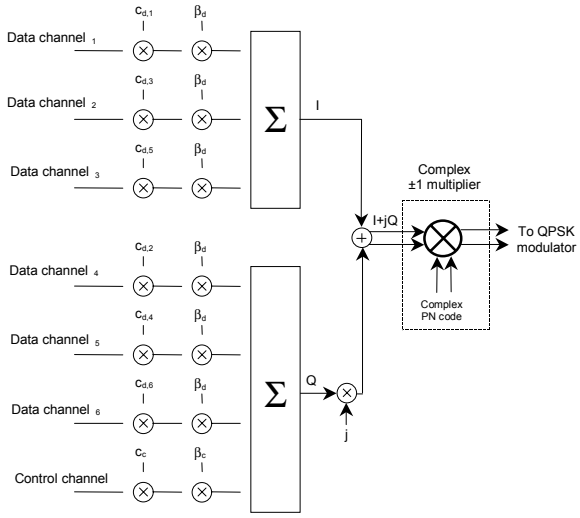


Fig. 1: 3GPP standard transmission scheme [1]

* This research is supported in part by the Semiconductor Research Corporation under Contract No. 99-TJ-687, the DARPA/ITO under AFRL Contract F29601-00-K-0182, grants from the New York State Office of Science, Technology & Academic Research to the Center for Advanced Technology – Electronic Imaging Systems and to the Microelectronics Design Center, and by grants from Xerox Corporation, IBM Corporation, Intel Corporation, Lucent Technologies Corporation, Eastman Kodak Company, and Photon Vision Systems, Inc.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GLSVLSI'02, April 18-19, 2002, New York, New York, USA.
Copyright 2002 ACM 1-58113-462-2/02/0004…$5.00.

In section 2, architectural tradeoffs and solutions are described along with some background on signed-binary arithmetic. Logic level design issues are discussed in section 3 while simulation data and a comparison of the proposed implementation with more standard alternatives are summarized in section 4. Some conclusions are offered in section 5.

2. ARCHITECTURAL DESIGN

Conventional architectural solutions to the complex multiplier problem are formally introduced in section 2-A. Some background on signed-binary (SB) arithmetic is presented in section 2-B. This approach is applied in section 2-C to the development of a proposed complex multiplier architecture.

A. Conventional Solutions

A symbolic description of a ±1 complex multiplier is shown in Fig. 2 where each of the outputs can take on one of

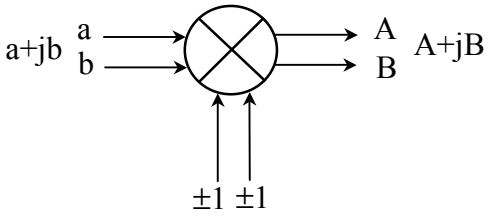


Fig. 2: Schematic symbol of a complex ±1 multiplier

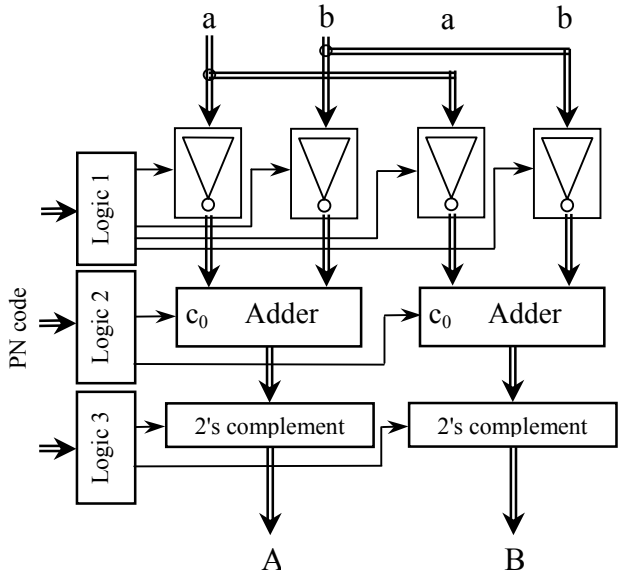
four possible values (as characterized in Table 1). The input signal is described by the complex number a + jb and the PN code by PN_{re} + jPN_{im} where PN_{re} and PN_{im} are in the binary set of {−1,+1}. The output complex signal is A + jB = (a + jb)·(PN_{re} + jPN_{im}). All of the numbers, a, b, A, and B, are

Table 1: Input / output relations for a ±1 complex multiplier

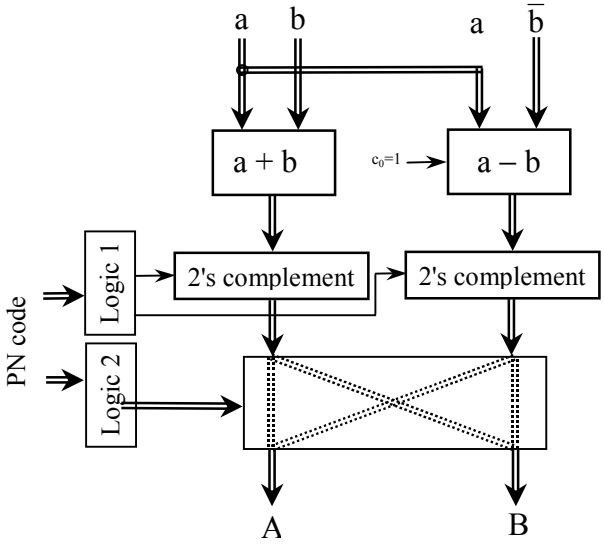
PN code		Outputs	
PN_{re}	PN_{im}	A	B
1	1	a – b	a + b
1	–1	a + b	– (a – b)
–1	1	– (a + b)	a – b
–1	–1	– (a – b)	– (a + b)

represented in two's-complement (TC) format with N-bit precision for the inputs and N+1-bit precision for the outputs.

The structure of a complex ±1 multiplier circuit is therefore different from that of a general purpose complex multiplier. Rather than considering two complex input operands, there is only one complex input and a set of two binary control signals, PN_{re} and PN_{im}. Two attractive architectural solutions to achieve this function are shown in Fig. 3. In the Type I architecture, the two branches are completely independent and both may produce any of the four functions, whereas in the Type II architecture each branch is dedicated to providing either the ±(a+b) or the ±(a–b) functions and a final switch is required to map these results to the correct complex output. Area and power improvements may be achieved by exploiting common features between the a+b and a–b operations. Circuit speed may be increased by reducing the overhead of the two's-complement circuits and the final switch in the type II architecture. The conditional switches controlled by the PN logic produce additional delay along the critical path and, therefore, the number of these gates should be minimized. Note that the critical path delay of both branches must be equal, so that valid results appear almost simultaneously at the outputs.



Type I: Independent-branches architecture



Type II: Switched-outputs architecture

Fig. 3: Conventional architectural solutions for a complex ±1 multiplier

B. The Signed-Binary Approach

Generally, numbers in VLSI-based digital circuits are represented in two's-complement format to facilitate the implementation of arithmetic operations. During the past decade, signed-binary number representation (SBNR) has received increasing interest due to the attractive features for carry free addition [4-7]. This capability leads to significant benefits in the implementation of wide adders and more complex arithmetic functions. However, the advantages of this approach decrease in small addition operations due to the significant overhead required for converting to the two's-complement system. The addition of two numbers in two's-complement format is essentially equivalent to the conversion of an SBNR number into the two's-complement counterpart, as shown by Blair [4]. The attractiveness of the signed-binary (SB) approach lies in the parallel block implementation of an adder [5,6] or in the utilization of this format in sequential arithmetic operations without the overhead of the final back conversion to TC [7].

The SBNR in sign-magnitude form is selected as an internal representation of the proposed multiplier architecture. In signed-binary format, the numbers b and –b differ only in the sign bits. This feature may be exploited to identify common stages of the a+b and a–b operations. Inverting a number in sign-magnitude format is accomplished by inverting all of the sign bits. This operation is more efficient than two's-complement, which has a complexity on the order of an adder. The benefits of the signed-binary representation may potentially increase if several arithmetic stages are incorporated in an SBNR tree before the final conversion to two's-complement [7]. Considering the significant complexity of CDMA multiuser detection algorithms [1]-[3], a number of demanding operations may be implemented in this intermediate format without inefficiently transforming all results into two's-complement format. This strategy leads to considerable improvements in power, area, and delay as compared to conventional TC arithmetic.

Having chosen a number representation, the tradeoffs that exist between the two proposed architectures are analyzed in more detail. The final switch of the Type II solution shown in Fig. 3 directs the results to the real and imaginary outputs. This switch may be eliminated if both the addition and subtraction operations over any operands (±a,±b) are produced in both branches as is the case in the Type I architecture. The switch may be implemented as 2(N+1) multiplexers, while in the Type I circuit, 4N gates are required. This switch, however, becomes less efficient if the SBNR results are supplied to the following stage, doubling the number of output lines to 4N.

The most expensive component in a signed-binary architecture is the SB→TC conversion. This operation is essentially equivalent to a two's-complement addition. Therefore, an existing efficient adder structure may be applied in this conversion process [4]. The implementation of an 8-bit circuit is possible through a carry-select architecture with 4-bit carry-generation blocks [5,6] or via a standard 8-bit carry-lookahead adder (CLA).

C. Proposed SBNR Architecture

In order to benefit from the advantages of redundant arithmetic, the two's-complement value of the SB number must be transformed into the required function. This transformation is relatively easy to achieve for a single function but becomes nontrivial when four functions are required for the real and imaginary outputs as specified in Table 1. One approach is to replace the addition and the formation of the CLA generate-propagate (G-P) signals with a preprocessing stage as reported in [4]. This concept is further developed to achieve an efficient implementation of all four functions of interest: (a+b), –(a+b), (a–b), and –(a–b) [8]. The objective is to ensure that the two branches have close to equal delay times with a minimum number of conditional switches controlled by the PN code logic.

The sum of any two bits is a digit in the *initial sum* set $S_y=\{0,1,2\}$, while the *signed-binary* set is $S_x=\{1,0,\bar{1}\}$. The computation of any of the four functions is accomplished in four steps:

1. Map the input bits directly to the signed-binary representation (number x) of the initial sum;
2. Manipulate this SB number according to the formulae characterizing the particular function;
3. Convert the SB number x to the equivalent two's-complement value of $T_x(x)$ (using a regular CLA adder and inverting output bits 0:N-1);
4. Set the N^{th} sign bit to produce the (N+1)-bit two's-complement function result.

The realization of each of these operations is briefly described below. Additional details are provided in [8]. The essence of the direct mapping from the input bits to signed-binary is in the transformation $tr_{xy}(z_i) = (1 - z_i)$ for $z_i \in \{S_x \cup S_y\}$, where $tr_{xy}(z_i)$ can operate on both sets and is self-inverting such that $tr_{xy}[tr_{xy}(z_i)] = z_i$ for $z_i \in \{S_x \cup S_y\}$ [4,8]. These transformations are listed in Table 2, where the first transformation performs the sum and the second transformation is represented by $tr_{xy}(y_i)$, where the initial sum digit y_i is mapped to the signed-binary digit x_i.

Table 2: Redundant arithmetic transformations at the prelogic stage

Input bits		+	Initial sum digit $y_i \in S_y$			tr_{xy}	Signed-binary digit $x_i \in S_x$		
a_i	b_i		y_i	c_{i+1}	S_i		x_i	$sign_i$	$magn_i$
0	0	→	0	0	0	→	1	0	1
01 , 10			1	0	1		0	0	0
1	1		2	1	0		$\bar{1}$	1	1

This process reveals the redundancy of the intermediate representation in S_y and the direct relations for the sign-magnitude format,

$$sign_i = a_i b_i = G_i \quad (1)$$

Table 3: Summary of transform relations for the functions in a complex ±1 multiplier

Arithmetic function	Bitwise processing	Relation to $T_Y(y)$	Relation to $T_x(x)$	Implementation description
a+b	a_i, b_i	$T_Y(y)$	$2^N-(1+T_x(x))$	1. Obtain all x_i from a_i,b_i, with +1 and invert sign bits 2. Produce the 2's complement $-(T_x(x)+1)$, set the N^{th} bit
–(a+b)	a_i, b_i	$-T_Y(y)$	$-2^N+1+T_x(x)$	1. Obtain all x_i from a_i,b_i with +1 2. Produce the 2's complement $T_x(x)+1$, set the N^{th} bit
(a–b)	a_i, $\overline{b_i}$	$T_Y(y)+1$	$2^N-T_x(x)$	1. Obtain all x_i from a_i and $\overline{b_i}$ 2. Invert all sign bits of x 3. Produce the 2's complement $-T_x(x)$, set the N^{th} bit
–(a–b)	a_i, $\overline{b_i}$	$-[T_Y(y)+1]$	$-2^N+T_x(x)$	1. Obtain all x_i from a_i and $\overline{b_i}$ 2. Produce the 2's complement $T_x(x)$, set the N^{th} bit

$$magn_i = \overline{a_i \oplus b_i} = \overline{P_i} \quad . \tag{2}$$

These functions are implemented in *prelogic* stages in both branches. Note that these intermediate signals are the same as the carry-generate G_i and the inverse of the carry-propagate P_i in the carry-lookahead adder [4,5,9,10]. It is conceptually convenient to distinguish between these signals such that a number is referred to as sign-magnitude when discussing the SBNR form and these signals are applied (with the magnitude inverted) to the corresponding *G-P* inputs of a CLA. The two's-complement values of the signed-binary representation and the initial sum are related according to

$$T_y(y)=T_y[tr_{xy}(x)]=\sum_{i=0}^{N-1} tr_{xy}(x_i)2^i = 2^N-1-T_x(x)=2^N+\overline{T_x(x)}, \tag{3}$$

where $\overline{T_x(x)}$ is the binary number $T_x(x)$ with all bits inverted. Similar expressions for all four functions in the complex multiplier are listed in Table 3 [8]. As described by (3), the addition may be accomplished by inverting the bits of $T_x(x)$. Unfortunately, such a realization is not possible for the –(a+b) function. In order to minimize the circuit differences between the (a+b) and –(a+b) functions, as realized in the left branch, an alternative realization of a+b through the $-[T_x(x)+1]$ operation is used. The left branch prelogic maps the input bits directly to $T_x(x)+1$ while the rest of the circuit remains the same as the right branch. The "+1" operation is simple to perform in SB. The following (N+1)-digit signed-binary result corresponds to $T_x(x)+1$ [8],

$$d_0'' \begin{cases} S_0'' = \overline{M_0'} = a_0 \oplus b_0 \\ M_0'' = \overline{M_0'} = a_0 \oplus b_0 \end{cases} \tag{4}$$

$$d_1'' \begin{cases} S_1'' = M_1' \cdot S_0' = \overline{a_1 \oplus b_1} \cdot a_0 b_0 \\ M_1'' = \overline{M_1' \oplus S_0'} = \overline{a_1 \oplus b_1} \oplus a_0 b_0 \end{cases} \tag{5}$$

$$d_i'' \begin{cases} S_i'' = M_i' \cdot \overline{\overline{S_{i-1}'} M_{i-1}'} = \overline{a_i \oplus b_i} \cdot (a_{i-1} + b_{i-1}) \\ M_i'' = M_i' \oplus \left(\overline{S_{i-1}'} M_{i-1}'\right) = \overline{a_i \oplus b_i} \cdot \overline{a_{i-1} + b_{i-1}} \end{cases} \quad 2 \le i \le N-1 \tag{6}$$

$$d_N'' \begin{cases} S_N'' = 0 \\ M_N'' = \overline{S_{N-1}'} M_{N-1}' = \overline{a_{N-1} + b_{N-1}} \end{cases} , \tag{7}$$

where each digit at the output d_i'' is expressed in sign-magnitude form and is a function of the signed-binary input or the input two's-complement operands, *a* and *b*. With these expressions, the SBNR of $T_x(x)+1$ is achieved in two gate delays. This number is converted to two's-complement or alternatively may be inverted while in signed-binary form to alternate between the (a+b) and –(a+b) functions.

This algorithm is only correct if the input operands are N-bit unsigned numbers or if the output result does not cause overflow in N-bit precision. This problem is addressed in [8] where it is shown that for two's-complement numbers, the N^{th}

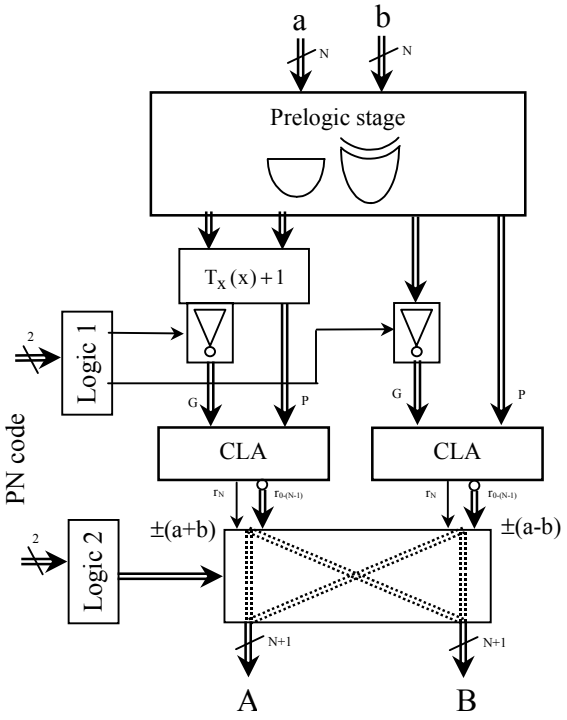


Fig. 4: SBNR architecture of a complex ±1 multiplier

(sign) bit of the general (N+1)-bit result is

$$r_N = a_{N-1}b_{N-1} + (a_{N-1} \oplus b_{N-1})\overline{c_{N-1}} = G_{N-1} + P_{N-1}\overline{c_{N-1}} \quad , \quad (8)$$

where r_i denotes the i[th] bit of the output result and c_i is the input carry of the i[th] full-adder cell. This function may be conveniently implemented by an inversion of the carry bit c_{N-1} passed to the circuit forming the final carry c_N, in order to account for the negative weight of the input sign bits. Since the sign bit is controlled according to (8), the computation of the N[th] digit as in (7) is unnecessary (only r_N and r_{N+1} are controlled by d_N). Therefore, an N-digit SB number is computed.

The realization of the ±(a–b) functions in the right branch is precisely the same as that of the left branch with the exception of the "+1" addition. In the specific logic implementation, this operation produces a negligible delay overhead. Since the left branch prelogic must operate on a and $\overline{b}$ (see Table 3), it is preferable to combine the prelogic stages at either the gate or the layout level, considering the following relations,

$$sign_i(+) = a_i b_i \qquad sign_i(-) = a_i \overline{b_i} \quad (9)$$

$$magn_i(+) = \overline{a_i \oplus b_i} \qquad magn_i(-) = \overline{a_i \oplus \overline{b_i}} = \overline{magn_i(+)} \quad . \quad (10)$$

An architecture that includes these design concepts is proposed in Fig. 4.

3. LOGIC LEVEL DESIGN

Most of the operations along the critical path are implemented in NMOS CPL logic due to the speed, power efficiency [11], and complementary outputs. Complementary outputs are employed to achieve an efficient implementation of the conditional inverters by integrating these circuits with the previous stage. The availability of complementary outputs also supports resource sharing between the (a+b) and (a–b) branches [see (9) and (10)].

The adder performing the conversion to two's-complement is the key logic circuit of the architecture shown in Fig. 4. Adder circuits have been extensively discussed in the literature and are applicable to the conversion circuit [4]. A slow but area- and power-efficient adder is preferable as long as the speed satisfies the target specification.

Several relations are deduced from the full adder truth table [9], (1)-(8), and the Karnaugh maps associated with the corresponding signals,

$$G_i = \overline{P_i}G_i \quad , \quad P_i = P_i\overline{G_i} \quad , \quad (11)$$

$$c_{i+1} = G_i + P_i c_i = \overline{P_i}G_i + P_i c_i \quad , \quad 0 \le i \le (N-1) \quad (12)$$

$$\overline{c_{i+1}} = \overline{P_i}\,\overline{G_i} + P_i\overline{c_i} \quad , \quad 0 \le i \le (N-1) \quad (13)$$

$$r_i = P_i \oplus c_i = \overline{P_i} \oplus \overline{c_i} = \overline{P_i \oplus c_i} \quad , \quad 0 \le i \le (N-1) \quad (14)$$

$$r_N = G_{N-1} + P_{N-1} \cdot \overline{c_{N-1}} = \overline{P_{N-1}}G_{N-1} + P_{N-1} \cdot \overline{c_{N-1}} \quad , \quad (15)$$

$$\overline{r_N} = \overline{P_{N-1}} \cdot \overline{G_{N-1}} + P_{N-1} \cdot c_{N-1} \quad . \quad (16)$$

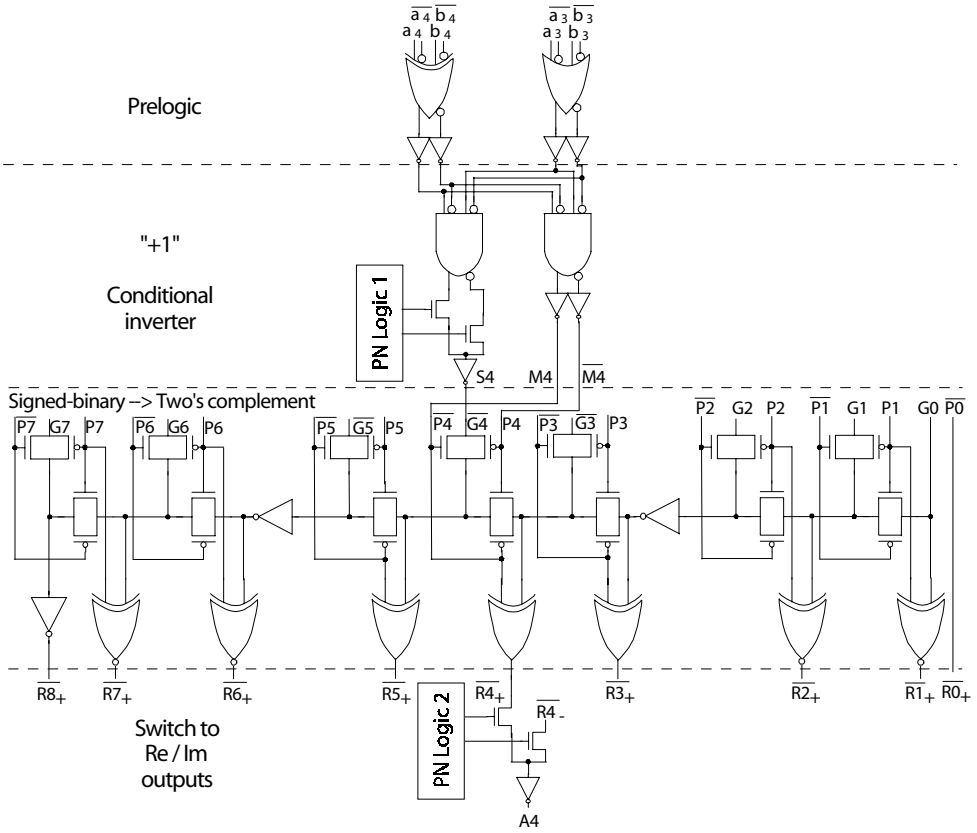


Fig. 5: An 8-bit implementation of the left branch of the SBNR architecture as shown in Fig. 4

Table 4: Area and delay of an 8-bit complex ±1 multiplier
Shaded areas correspond to functional units, which are either
not applicable to the specific architecture or are not along the critical path.

Functional unit	Proposed SBNR architecture			Type I: Independent-branches architecture			Type II: Switched-outputs architecture		
	Area	Delay		Area	Delay		Area	Delay	
	[μm^2]	gates	[ps]	[μm^2]	gates	[ps]	[μm^2]	gates	[ps]
Prelogic	1500	1	200	1000	1	200			
"+1" and inverter	1900	1 (+)	250						
Converter or adder	3000	N	1350	3000	N	1350	3000	N	1350
Two's complement				3000	N	1350	3000	N	1350
Switch	900	1	200				900	1	200
PN logic	300			600			300		
Total	**7600**	**N+3**	**2000**	**7600**	**2N+1**	**2900**	**7200**	**2N+1**	**2900**

As shown in (12)-(14), the inverted carry may also be propagated, permitting a single inverter to be inserted as a repeater to speed up the carry propagation chain. The sign bit is controlled according to (15) and (16). An expression for the sign inverse is also implemented by an inverter to achieve higher output current and enhanced noise margins. Since r_N is a function of c_{N-1}, an even number of inverters is required along the propagation chain.

The critical path delay includes a carry propagation through N – 1 transmission gates and one inverter [6] (note that $c_0 = 0$, so effectively G_0 is propagated). The carry propagation speed is significantly increased through logic level optimization of the transmission gate chain. Inserting two additional inverters in a chain of seven transmission gates decreases the propagation time approximately three times. In this case, either the carry signal or the inverse carry signal is propagated.

4. SIMULATION RESULTS

To demonstrate the performance characteristics of the proposed architecture, an 8-bit circuit is analyzed in a TSMC 0.25 μm CMOS technology with $V_{DD} = 2.5$ volts. Delay and area estimates are presented in Table 4 along with a comparison of the alternative architectures under similar technology and bias conditions. The architectures shown in Fig. 3 have similar area-delay characteristics and, for approximately the same area, the delay of the critical path is 50% higher in these conventional architectures than the proposed SBNR realization. This increased speed is due to reducing the two carry propagation chains to a single chain in the proposed architecture. Adder techniques that trade off area, delay, and/or power may be applied to the three architectures to customize the circuit according to application-specific performance requirements.

The circuit is targeted for a base station receiver where power consumption is not a primary consideration. Since in CPL most of the PMOS transistors along the critical path are eliminated, the node capacitances are significantly reduced, thereby achieving a higher operational speed and lower power consumption. The signal path in CPL is along the source-drains rather than the gates and the gate capacitance is usually much larger than the junction capacitance. Therefore, the delay is further reduced.

5. CONCLUSIONS

An efficient architecture of a complex ±1 multiplier circuit is proposed in this paper. Redundant signed-binary arithmetic is used to achieve a significant reduction in the critical path delay. A comparison of these results with standard architectures is provided. A speed increase of about 50% is observed in the proposed SBNR architecture as compared to conventional architectures.

REFERENCES

[1] 3G TS 25.213 V3.4.0 (2000-12) 3rd Generation Partnership Project; TSG Radio Access Network; Spreading and Modulation (FDD), Release 1999.

[2] R. Cameron, *Fixed-point Implementation of a Multistage Receiver*, Ph.D. Thesis, Virginia Polytechnic Institute and State University, January 1997.

[3] N. Zhang *et al*., "Architectural implementation Issues in a Wideband Receiver Using Multiuser Detection," Proceedings of the Annual Allerton Conference on Communication, Control and Computing, pp. 765-771, September 1998.

[4] G. M. Blair, "The Equivalence of Twos-complement Addition and the Conversion of Redundant-binary to Twos-complement Numbers," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, Vol. 45, No. 6, pp. 669-671, June 1998.

[5] J. Dobson and G. M. Blair, "Fast Two's Complement VLSI Adder Design," *Electronic Letters*, Vol. 31, No. 20, pp. 1721-1722, September 28, 1995

[6] H. Srinivas and K. Parhi, "A Fast VLSI Adder Architecture," *IEEE Journal on Solid-State Circuits*, Vol. 27, No. 5, pp. 761-767, May 1992.

[7] T. Kim and J. Um, "A Practical Approach to the Synthesis of Arithmetic Circuits Using Carry-save Adders," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 19, No. 5, pp. 615-624, May 2000.

[8] B. D. Andreev, E. G. Friedman, and E. L. Titlebaum, "On Some Transformations of Number Representation for Efficient VLSI Arithmetic," in preparation.

[9] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, 2nd Ed., Addison-Wesley, 1993.

[10] R. Brent and H. Kung, "A Regular Layout for Parallel Adders," *IEEE Transactions on Computers*, Vol. C-31, No. 3, pp. 260-264, March 1982.

[11] K. Yano *et al*., "A 3.8 Ns CMOS 16x16-b Multiplier Using Complementary Pass-Transistor Logic," *IEEE Journal on Solid-State Circuits*, Vol. 25, No. 2, pp. 388-395, April 1990