The CSI Journal on Computer Science & Engineering

The CSI Journal on
Computer Science and Engineering
Vol. 7, No. 2 & 4 (b), 2009
Pages 1-7
Regular Paper

A Modulo 2^n+1 Multiplier with Faithful Representation of Residues

Ghassem Jaberipur Hanieh Alavi Saeed Nejati

Department of Electrical and Computer Engineering, Shahid Beheshti University, Tehran, Iran

Abstract

Modulo 2^n+1 adders and/or multipliers are used in digital filters, cryptographic systems, and digital signal processors based on residue number systems (RNS). The module set $\{2^n–1, 2^n, 2^n+1\}$ is popular in RNS applications, where the design of modulo 2^n+1 multipliers is more challenging than the case of other two module. One reason is that the natural representation of residues in the range $[0, 2^n]$ requires n+ 1 bit. However, a number of modulo 2^n+1 addition or multiplication schemes have used n-bit diminished-1 representation of residues, where zero operands are supposed to be treated separately. On the other hand, double-LSB encoding of modulo 2^n+1 residue (i.e., an n-bit code word with a second least significant bit) has been used in the design of an efficient modulo 2^n+1 adder. We are therefore, motivated to study the impact of the double-LSB encoding of residues on the design of modulo 2^n+1 multipliers. We describe the operation of such multipliers in dot-notation representation and show that the corresponding circuitry uses only standard off the shelf arithmetic cells such as full adders, half adders and carry look-ahead logic. Synthesis based comparison with previously reported multipliers shows the advantages of the proposed design.

Keywords: Modular Multiplier, Residue Number System, Double-Lsb Encoding.

1. Introduction

Modulo $2^n + 1$ adders and/or multipliers are used in a number of applications including digital filters [1], cryptography [2-4], and generally in digital signal processing based on residue number systems (RNS) [5]. The moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ is popular in RNS applications, where the same modular arithmetic operations are performed in the three computation channels in parallel. However, the design of modulo $2^n + 1$ *a*rithmetic circuits is more challenging than the case of straightforward modulo 2^n arithmetic and that of modulo $2^n - 1$. One reason is that the natural representation of residues in the range $[0, 2^n]$ *re*quires n+1 bit versus the n-bit representation for other two modules. A number of modulo $2^n + 1$ addition or multiplication schemes have used diminished-1 representation of residues, where zero operands are treated separately [6-13].

Some other works use normal binary (n+1)-bit encoding of residues [4, 14, 15]. Both schemes have special provisions, bearing additional complexity, for handling the end-around carry. Double-LSB encoding [16] of modulo $2^n + 1$ residues has been used in the design of a modulo $2^n + 1$ adder [17], where each operand is represented by a normal n-bit number that is augmented with a second least significant bit (LSB). This encoding provides a faithful representation, where there would be no code word that evaluates to a number out of the valid range $[0, 2^n]$. Therefore, in this paper, we are motivated to study the impact of double-LSB encoding on the design of modulo $2^n + 1$ multipliers.

The rest of the paper is organized as follows. Some recent designs for modulo $2^n + 1$ multipliers are addressed in Section II and the proposed multiplier is presented in Section III. We provide, in Section IV, the results of simulations and synthesis for the best previous modulo $2^n + 1$ multiplier and those of the proposed one. Finally we draw our conclusions in Section V.

2. General Modulo 2^n+1 Multiplication

The first contribution on memory-less modulo $2^n + 1$ multiplication, as we have encountered, is due to [3]. This and some other works are based on diminished-1 representation of residues [6, 8-13]. In this representation, values in $[1, 2^n]$ are represented by *n*-bit code words $^n0$ (i.e., a string of *n* 0s) to $^n1$, respectively. Zero is exclusively represented via an extra bit and a zero result is handled separately. This zero-indicator bit is always 0 except for true zero residue. On the other hand, [14] and [15] have used unsigned (*n*+1)-bit binary numbers to represent values in$[0, 2^n]$, where the most significant bit (MSB) is always 0, except in the representation of2^n. Therefore, the MSB acts like a 2^n-indicator. In some applications with no zero operand, such as International Data Encryption Algorithm (IDEA) [3], *n*-bit encoding is used [4], where 2^n is represented by the code word. In the following we briefly look at three works on modulo-$(2^n + 1)$ multipliers based on the aforementioned encodings of residues; namely [4] for zero less residues, [6] on diminished-1 and the one due to [15] that uses (*n*+1)-bit encoding.

2.1. Modulo 2^n+1 Multiplier with Zero-Free Residues

Implementation of the IDEA cryptosystem [3] requires modulo-$2^n + 1$ multiplication $|x \times y|_{2^n+1}$ ($|e|_m = e \bmod m$), where zero never occurs as residual values [4]. The all-0 *n*-bit code word is reserved to represent 2^n. Therefore, a special 2^n-handler is needed to detect the case of all-0 multiplier (i.e. $^n0$) and compute $|2^n x|_{2^n+1} = |-x|_{2^n+1} = |2 + 2^n - 1 - x|_{2^n+1} = 2 + \bar{x}$, where *x* is the multiplicand and $\bar{x}$ is it's bitwise inverted code word. The main multiplier is basically designed as the standard *n*×*n* binary multipliers. However, every bit $p_{n+k,j} = x_i y_j (0 \le i, j, k \le n-1,\ k = i + j - n)$ of the j^{th} partial product, is evaluated as $\left|2^{n+k} p_{n+k,j}\right|_{2^n+1} = 2^k(-p_{n+k,j})$. Given that $-p_{n+k,j} = \overline{p_{n+k,j}} - 1$, $p_{n+k,j}$ is removed from position *n*+*k* and $\overline{p_{n+k,j}}$ is inserted in position *k*. Moreover, a corrective value -2^k shall be compiled and evaluated with other similar values.

Figure 1 illustrates the latter operation, where the compiled corrective values per displaced bits amount to $-(2^j - 1)$ for the j^{th} partial product. The sum of these corrective values, is computed as in Eq. 1. Therefore, the corrective –1 value need not be actually inserted in the partial product matrix.

$$\left|-\sum_{j=1}^{n-1}(2^j - 1)\right|_{2^n+1} = -|2^n - n - 1|_{2^n+1} = n + 2 \qquad (1)$$

The *n*×*n* partial product matrix can normally be reduced to a 2×*n* matrix via (*n*–2) carry-save adders (CSA), where the carry-out *c* of the leftmost full adder of each CSA is returned to position 0 as $-1 + \bar{c}$. These new corrective –1 values amount to – (*n*–2). Adding the latter to the sum of earlier corrective values (i.e., *n*+2) leads to 4 as the overall corrective value, which can be represented as a 1 in position 2. To handle this extra 1 another level of carry-save reduction is necessary, which calls for another corrective –1. Therefore, the speculative corrective value is actually 3. A mathematical derivation of this amount is given in [14].

	x_{n-1}	x_{n-2}	⋯	x_2	x_1	x_0
×	y_{n-1}	y_{n-2}	⋯	y_2	y_1	y_0
	$p_{n-1,0}$	$p_{n-2,0}$	⋯	$p_{2,0}$	$p_{1,0}$	$p_{0,0}$
	$p_{n-2,1}$	$p_{n-3,1}$	⋯	$p_{1,1}$	$p_{0,1}$	$\overline{p_{n-1,1}}$ -1
	$p_{n-3,2}$	$p_{n-4,2}$	⋯	$p_{0,2}$	$\overline{p_{n-1,2}}$ -1	$\overline{p_{n-2,2}}$ -1
			⋮			
	$p_{0,n-1}$	$\overline{p_{n-1,n-1}}$ -1	⋯	$\overline{p_{3,n-1}}$ -1	$\overline{p_{2,n-1}}$ -1	$\overline{p_{1,n-1}}$ -1

Figure 1. Zero-free modulo 2^n+1 Partial product generation

2.2. Modulo 2^n+1 Multiplier Based on Diminished-1 Representation of Residues

Diminished-1 (D1) representation of a binary number *a* is encoded as (a_z, a'), where $a_z = 1$ (0) if $a = 0$ ($\neq 0$) and $a' = a - 1$. Therefore, modulo $2^n + 1$ residues in $[1,2^n]$ are mapped to *n*-bit numbers in $[0, 2^n - 1]$. Let $p = x \times y$, where $p \equiv (p_z, p')$, $x \equiv (x_z, x')$ and $y \equiv (y_z, y')$. In case of either or both operands being zero the zero indicator for the product is computed as $p_z = x_z \vee y_z$. The D1 product can be expressed as $p' = x \times y - 1 = (x' + 1) \times (y' + 1) - 1 = x' \times y' + x' + y'$. To compute p', the partial product matrix for $x' \times y'$ (obtainable as in Section A above) is augmented by x' and y'. Therefore, two more corrective –1 values are generated that reduce the overall corrective value from 3 to 1. After the operation of the final CSA, which is a simplified one due to representation of the third operand as 00…01, an *n*-bit end-around carry modulo $2^n + 1$ adder is required to compute the final product. The most efficient architecture for this adder [10], as reproduced in Figure 2, is a totally parallel prefix (TPP) adder with $\frac{3n}{2}(logn - 1) + 2$ parallel prefix computing nodes recognized as black circles. The TPP is considerably more complex than regular parallel prefix tree. The latter, as depicted by Figure 3, has $n.\log n - n + 1$ black nodes (all single) and no backward connections, where the white circles represent simple buffers and white squares (in both Figure 2 and 3) produce the conventional propagate, generate and half-sum signals. A compromise design uses another level of black nodes, instead of reverse connections to accommodate the end around carry [10].

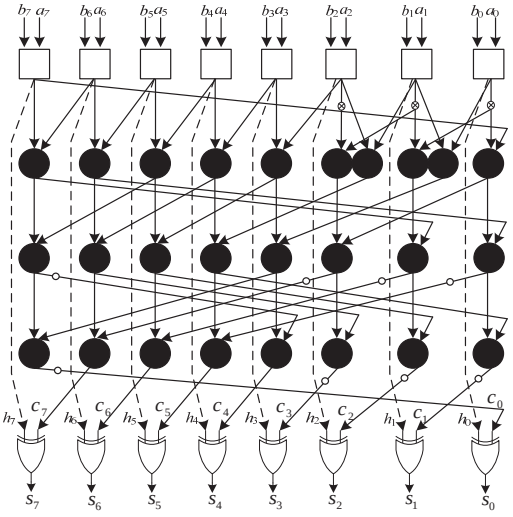


Figure 2. The end-around carry modulo-2^n+1 TPP adder [10]

The aforementioned D1 scheme has been implemented in [6]. However, the zero-output case $x \times y = 2^n + 1 = |0|_{2^{n+1}} \Longrightarrow p_z = 1$, that can occur for nonprime moduli (e.g., 9, 33, 65, 129, for n =3, 5, 6, 7), is not duly addressed.

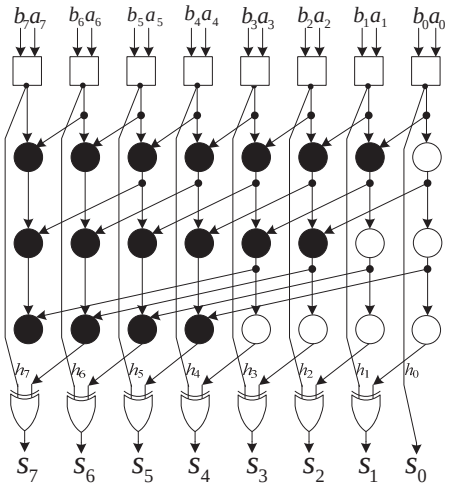


Figure 3. Regular parallel prefix adder

2.3. Modulo 2^n+1 Multiplier Based on (n+1)-Bit Representation of Residues

The range of modulo-$(2^n + 1)$ residues is $[0,2^n]$. The straight forward representation for these values uses an (n+1)-bit code word, where the most significant bit is 0 but for 2^n. Therefore, there are $2^n - 1$ code words that represent invalid values in $[2^n + 1,2^{n+1} - 1]$. Vergos et al. [15] have used such representation in the design of modulo-$(2^n + 1)$ multipliers with a special partial product generation scheme. This method leads to a provisional, diamond shape, $n \times n$ partial product bit matrix, which is achieved at the cost of four logic levels delay, while the corresponding delays in the previous two Subsections equal to that of one AND gate. Similar analysis as was explained in Section A can be used to rearrange the partial products as a straight rectangle (see Figure1). Then standard carry-save adders are used to reduce the depth of partial product matrix to two. The corrective value before the final addition is 3. Therefore, the final CSA is again a simplified one. This leads to representation of the product as two equally weighted n-bit numbers, which should be added modulo-$(2^n + 1)$.

3. Modulo 2^n+1 Multiplier Based on Double-LSB Representation of Residues

Double-LSB (DLSB) encoding of modulo $2^n + 1$ residues is basically an n-bit unsigned binary number that is augmented with an extra least significant bit (LSB). The encoding faithfully (i.e., without any invalid code word) represents residues in $[0, 2^n]$. In particular, the code word representing the singular value 2^n contains only 1-valued bits. Figure 4 depicts the partial products generated as the first step of multiplying two double-LSB residues $x = x_{n-1} \ldots x_0 x'_0$ and $y = y_{n-1} \ldots y_0 y'_0$, where primed variables denote second LSBs.

Let $p_{k+n} = x_i y_j$ denote an arbitrary bit, in Figure 4, that is on and beyond position n, where $i + j = k + n$ for$k \geq 0$. The following Equation shows that the modulo-$2^n + 1$ arithmetic worth of any such bit p_{k+n} is equal to that of a negabit, in position k, with the same logical value.

$$\left|2^{k+n} p_{k+n}\right|_{2^n+1} = \left|2^k (2^n + 1 - 1) p_{k+n}\right|_{2^n+1} = -2^k p_{k+n}$$

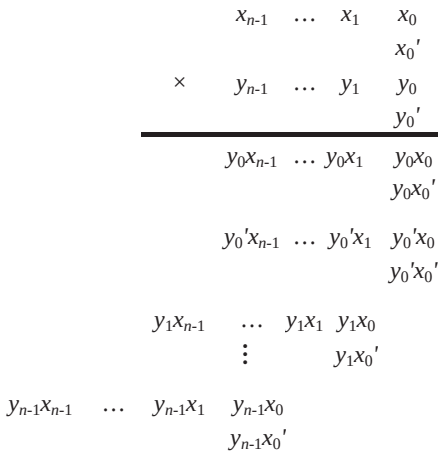


Figure 4. Partial products for multiplying two DLSB numbers

Figure 5 depicts a mixed symbolic dot notation representation after the above transformations, where white (black) circles denote negabits (posibits). Note that the megabit expressions are inverted for reasons to be explained below. Let N_i and P_i denote the number of negabits and posibits in position $i(0 \leq i \leq n-1)$, respectively, which can be computed via Eq. set 2, except for $P_0 = 4$. Therefore, the number of partial products amounts to $N_i + P_i = n + 2$.

$$N_i = n - (i + 1), \quad P_i = 2 + (i + 1) \tag{2}$$

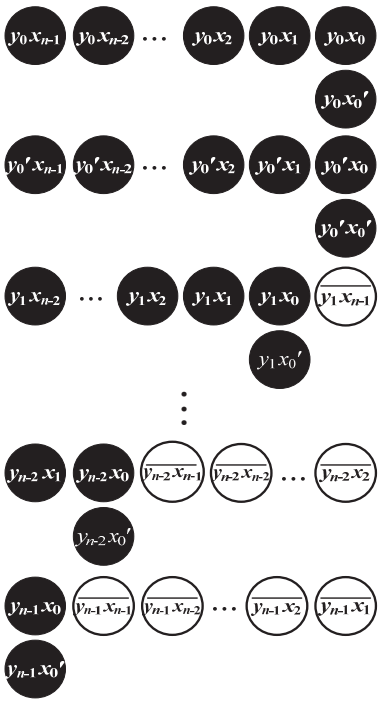


Figure 5. Partial products of DLSB modulo- $(2^n + 1)$ multiplication

It has been shown elsewhere [18, 19] that standard full (half) adders may accept any 3- (2-) collection of posibits and inversely encoded negabits and produce correct sum and carry bits. Inverted encoding of negabits uses logical states 0 and 1 for arithmetic values –1 and 0, respectively. Figure 6 depicts all the different posibit/megabit arrangements that can be correctly handled by standard full adders and half adders, where ● (○) denotes a posibit (inversely encoded megabit).

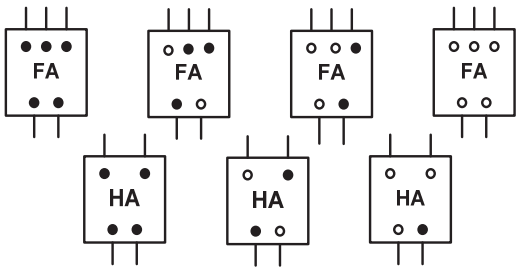


Figure 6. FA/HA with mixed polarity inputs and outputs

Figure 7 illustrates the partial product reduction, for n = 4, where the number of negabits and posibits in each column are derived via Eq. set 2. Standard carry-save adders (CSA) are used to reduce the depth of partial products to two. The HCSA stage consists of half adders and is used to appropriately change the bit polarities such that the output of the final carry propagate adder (CPA) is represented by posibits only. This stage is not always necessary. For example, in the case of $n = 5$ (Figure 8), the result after CSA IV is readily in the desired form.

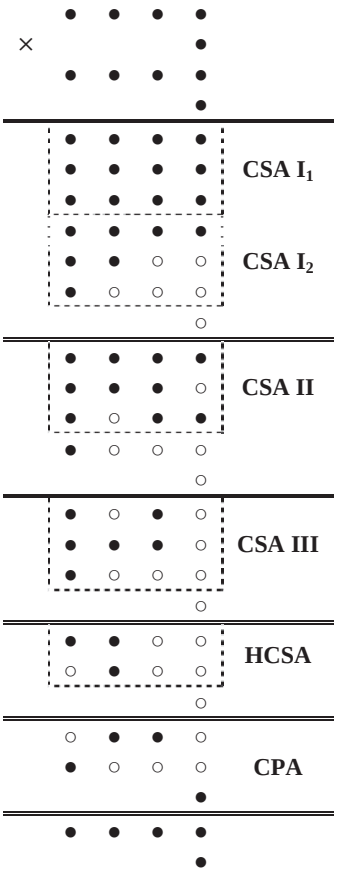


Figure 7. Modulo 17 multiplier

Figure 9 shows an actual high level circuit implementation of the proposed multiplier for n = 4. Note that the final n-bit adder is generic and can be replaced by any desired architecture. For example, in case of n = 5, the parallel prefix adder (PPA) of Figure 3 works fine, where the inputs for positions 5 to 7 are obviously zero. However, for n = 4, we can leave the posibit in position 0 intact, as the final second LSB, and use the end around carry architecture of Figure 2 for summing the two n-bit numbers modulo $2^n + 1$.

4. Comparison with the Best Previous Work

The best previous modulo $2^n + 1$ multiplier, as we have investigated, is due to [15], which was briefly described in Section 2.3. There are three improving design decisions that are undertaken in the proposed double-LSB scheme:

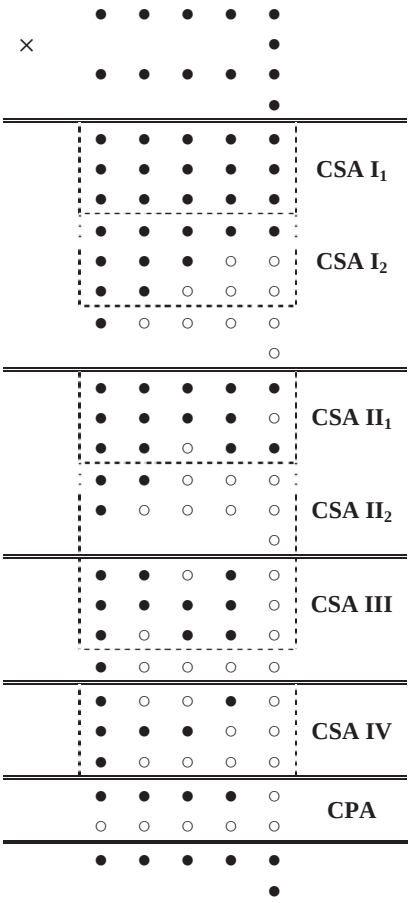


Figure 8. Modulo 33 multiplier

- **Partial product generation (PPG)**: The PPG matrix, in the proposed design, is composed of $n(n+1)/2 + 2n + 1$ AND and $n(n-1)/2$ energy-efficient NAND gates (see figure 7 and 8). However, the latency and gate complexity of PPG in [15] is determined by a collection of an XOR, an AND, and a 3-input OR gate that is used for some nodes in the PPG matrix.
- **Partial product reduction (PPR)**: The effective depth (i.e., except for rightmost column) of PPR tree, in the proposed scheme, is n+2 and in that of [15] is n+1. Since both schemes use CSA reduction cells, this minimal difference translates to equal number of

reduction levels (RL) in most cases. For example, for $4 \le n \le 7$, RL is equal in both schemes, except for $n = 5$ and for $64 \le n \le 1024$, there are only six cases (i.e., $n \in \{93,140,210,315,473,710\}$) that the proposed scheme requires one more RL. Both schemes can make use of (4; 2) compressors and deeper ones for faster performance and/or more VLSI regularity. The presence of negabits in the proposed PPR tree does not jeopardize this improvement for the proposed scheme. The reason is that conventional compressors can handle any mix of posibits and inversely encoded negabits [20].

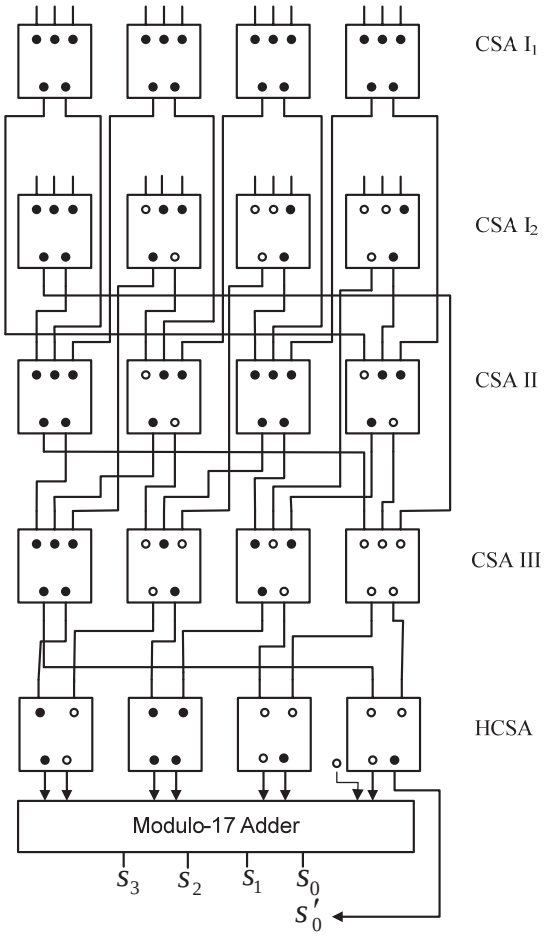


Figure 9. A high level circuit implementation of Figure 7

- **Final carry-propagating addition**: We use a conventional *n*-bit adder, as in Figure 8, that can be replaced by any carry-accelerating adder (e.g., Figure 3). However, [15] uses a specialized parallel prefix adder (Figure 2) that is more complex than a regular one in terms of wiring, regularity, and the number of computing nodes [17].

The above differences hint at improvements in delay, area and power. To confirm the latter claim we have synthesized the proposed multiplier and that of [15] for *n* = 8 and 16 under TSMC 0.13 μm CMOS technology with conservative wire model using Synopsys Design Compiler.

Power results are calculated using with 50% static probability on all inputs. The results are shown in Table 1 where the ratio of DLSB measures to that of [15] show the advantages of our design, especially in 9% more speed.

Table 1. Performance comparison

Design	Delay(*ns*)		Area (μm)		Power (mW)	
	8	16	8	16	8	16
DLSB	2.16	2.55	3825	5733	3.63	5.91
[15]	2.37	2.80	4091	5910	3.72	6.02
Ratio	0.91	0.91	0.93	0.97	0.97	0.98

5. Conversion Between DLSB and Natural Encoding

Let $a_n a_{n-1} \dots a_0$ denote the natural (n+1)-bit representation of integer $A \in [0, 2^n]$ and $A_{n-1} \dots A_0 A'_0$ represents its DLSB encoding. The singular case $[a_n = 1]$ implies both $[a_{n-1} = \dots = a_0 = 0]$ and $[A_{n-1} = \dots = A_0 = A'_0 = 1]$. Also $[a_n = 0]$ leads to $A_i = a_i$, for $0 \le i \le n-1$ and $A'_0 = a_n$. This concludes an easy conversion to DLSB via the following equations:

$$A_i = a_i \oplus a_n, \text{ for } 0 \le i \le n-1, A'_0 = a_n$$

The reverse conversion can be done via the simple increment $A_{n-1} \dots A_0 + A'_0$. However, the CPA stage of Figure 7 can use an inverted end around carry adder to directly generate the natural (n+1)-bit representation of A. Here is how it works: We use a full adder in the rightmost position of the HCSA stage of Figure 7, which leads to the 2-deep representation of Figure 9, where an inverted end around carry adder, like the one in Figure 2, can take over to produce the desired result. Note that the most significant bit of the result comes from the *G* output of the bottom leftmost black node of Figure 2.

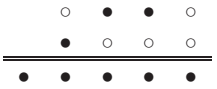


Figure 10. Generation of (n+1)-bit product

6. Conclusion

We have used double–LSB encoding for faithful representation of modulo $2^n + 1$ residues. Given that the relevant modulo $2^n + 1$ adder has been previously designed, we presented the design details of the corresponding modulo 2^n+1 multiplier. The design procedure is quite simpler than that of similar works that have appeared in the literature, where the main characteristics and benefits are:

- The mixed use of posibits and negabits in the partial product matrix obviates the need for complex, though off-line, computation of the collective value of corrective −1 constants.
- The practice of inverted encoding of negabits has allowed for the use of standard compression cells (e.g., full adder) and conventional *n*-bit adders.
- The main benefit of double-LSB encoding is the possibility of storing the final end-around carry as the second LSB of the product, thus no need for another round of carry propagation nor using complex specialized parallel prefix adders to avoid the latter.

The synthesis of the best previous modulo-$2^n + 1$ adder and the proposed one, both for *n* = 8 (16) under 0.13 μm CMOS technology shows 9% (9%), 7% (3%) and 3% (2%)

savings in latency, area consumption and power dissipation, respectively.

Further research can explore the possible benefits of modulo $2^n + 1$ multiply-accumulator with double-LSB inputs and output.

References

[1] J. Ramirez, and U. Meyer-Baese, "High Performance, Reduced Complexity Programmable RNS–FPL Merged FIR Filters," *Electronic Letters*, Vol. 38, No. 4, pp. 199–200, 2002.

[2] R. Zimmermann, A. Guriger, H. Bonnemberg, H. Kaeslin, N. Felber, and W. Fichtner, "A 177 Mb/s VLSI Implementation of the International Data Encryption Algorithm," *IEEE J. Solid-State Circuits*, Vol. 29, No. 3, pp. 303-307, 1994.

[3] X. Lia and J. L. Messay, "A proposal for a new block encryption standard," *Proceedings of Advances in Cryptography*- EUROCRYPT'90, Berlin, Germany, pp. 389-404. 1990.

[4] Y. J. Chen, D. R. Duh, and Y. S. Han, "A New Modulo 2^n+1 Multiplier for IDEA," *Proceedings of the International Conference on Security and Management* (SAM'04), Las Vegas-Nevada, pp. 318-324. 2004.

[5] E. D. Claudio, et al. "Fast Combinatorial RNS Processors for DSP Applications," *IEEE Trans. On Computers*, Vol. 44, No. 5, pp. 624–633, 1995.

[6] C. Efstathiou, H. T. Vergos, G. Dimitrakopoulos, and D. Nikolos, "Efficient diminished-1 Modulo 2^n+1 multipliers," *IEEE Trans. On Computers*, Vol. 54, No. 4, pp. 491-496. 2005.

[7] L. M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform," *IEEE Trans. Acoustics, Speech, and Signal Processing*, Vol. 24, pp. 356-359. 1976.

[8] M. Benaissa, A. Bouridane, S. S. Dlay, and A. G. J. Holt, "Diminished-1 Multiplier for a Fast Convolver and Correlator Using the Fermat Number Transform," *IEE Proceeding on Electronic Circuits and Systems*, Vol. 135, No. 5, pp. 187-193, 1988.

[9] S. Sunder, F. El-Guibaly, and A. Antoniou "Area-Efficient Diminished-1 Multiplier for Fermat Number-Theoretic Transform," *IEE Proceedings on Circuits, Devices and System*, Vol. 140, No. 3, pp. 211-215, 1993.

[10] H. T. Vergos, C. Efstathiou, and D. Nikolos, "Diminished-One Modulo 2^n+1 Adder Design," *IEEE Trans. on Computers*, Vol. 51, pp. 1389-1399, 2002.

[11] Y. Ma, "A Simplified Architecture for Modulo 2^n+1 Multiplication," *IEEE Trans. on Computers*, Vol. 47, No. 3, pp. 333-337, 1998.

[12] Z. Wang, G. A. Jullien, and W. C. Miller, "An Algorithm for Multiplication Modulo 2^n+1," *Proceedings of the 29th Asilomar Conference on Signals, Systems and Computers, CA., USA*, Vol. 2, pp. 956-960, 1995.

[13] H. T. Vergos, and C. Efstathiou, "A Unifying Approach for Weighted and Diminished-1 Modulo 2^n+1 Addition," *IEEE Trans. on Circuits and Systems II: Express Briefs*, Vol. 55, No. 10, pp. 1041-1045, Oct. 2008.

[14] R. Zimmermann, "Efficient VLSI Implementation of Modulo 2^n±1 Addition and Multiplication," *Proceedings of the 14th IEEE Symp. Computer Arithmetic*, Zurich, Switzerland, pp. 158-167, 1999.

[15] H. T. Vergos, and C. Efstathiou, "Design of Efficient Modulo 2^n+1 Multipliers," *IET Computer & Digital Techniques*, Vol. 1, No. 1, pp. 49-57, January 2007.

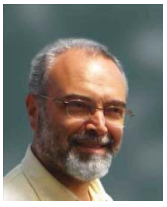
[16] B. Parhami, "Double-Least-Significant-Bits 2's-Complement Number Representation Scheme with Bitwise Complementation and Symmetric Range," *IET Circuits, Devices & Systems*, Vol. 2, No. 2, pp. 179-186, 2008.

[17] G. Jaberipur, "A One-step Modulo 2^n+1 Adder Based on Double-lsb Representation of Residues," *The CSI Journal on Computer Science and Engineering*, Vol. 4, No. 2&4, pp. 10-16, 2006.

[18] G. Jaberipur, and B. Parhami, "Weighted Two-Valued Digit-Set Encodings: Unifying efficient Hardware Representation Schemes for Redundant Number Systems," *IEEE Trans. On Circuits and Systems*, Vol. 52, No. 7, pp. 1348-1357, 2005.

[19] G. Jaberipur, and B. Parhami, "Posibits, Negabits, and their mixed use in efficient realization of arithmetic algorithms," *Proceedings of The 15th International CSI Symposium on Computer Architecture & Digital Systems*, pp. 3-9, 2010.

[20] G. Jaberipur, and B. Parhami, "Constant-Time Addition with Hybrid-Redundant Numbers: Theory and Implementations," *Integration, the VLSI Journal*, Vol. 41, No. 1, pp. 49-64, 2008.



Ghassem Jaberipur received B.S. in Electrical Engineering and Ph.D. in Computer Engineering from Sharif University of Technology in 1974 and 2004, respectively, M.S. in Engineering (majoring in computer hardware) from University of California, Los Angeles, in 1976, and M.S. in Computer Science from University of Wisconsin, Madison, in 1979. Since 1979, he has been with the Department of Electrical and Computer Engineering, Shahid Beheshti University, in Tehran, Iran, teaching courses mainly in computer arithmetic and implementation of programming languages. Dr. Jaberipur is also affiliated with the School of Computer Sciences, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.

E-mail: jaberipur@sbu.ac.ir

Saeed Nejati received his B.Sc. in Computer Engineering and M.Sc. in Computer Architecture from Shahid Beheshti University in 2007 and 2010, respectively. He is currently a Research Assistant in the School of Computer Sciences, Institute for Research in Fundamental Sciences (IPM), and also in the Department of Electrical and Computer Engineering, Shahid Beheshti University, Tehran, Iran.
E-mail: s.nejati@mail.sbu.ac.ir

Hanieh Alavi received her B.S in Computer Engineering from Azad University of Tehran Markaz in 2004, and M.S in Computer Architecture from Shahid Beheshti University in 2008. Since 2004, she's been working at IT Department of IMPASCO.
E-mail: hanie_alavi@mail.sbu.ac.ir

Paper Handling Data:

Submitted: 26.05.2010
Received in revised form: 25.06.2011
Accepted: 07.07.2011
Corresponding author: Dr. Ghassem Jaberipur,
Department of Electrical and Computer Engineering, Shahid Beheshti University, Tehran, Iran.