Hard Multiple Generator for Higher Radix Modulo 2^n-1 Multiplication

Ramya Muralidharan and Chip-Hong Chang
Centre for High Performance Embedded Systems
Nanyang Technological University, Singapore

***Abstract*—High-speed modulo multipliers are essential elements in RNS datapath. Booth recoding algorithm can be used to improve the performance of the multiplier by reducing the number of partial products. In radix-8 Booth encoding, the number of partial products is reduced to one-third. However, the inevitable carry propagation adder required to generate the hard multiple, $3X$, where X is the multiplicand, falls on the critical path of the multiplier. This paper presents an efficient modulo 2^n-1 hard multiple generator based on the parallel-prefix addition. The proposed hard multiple generator employs $\lceil \log_2 n \rceil - 1$ prefix levels, making radix-8 Booth encoding a feasible choice for high-speed modulo 2^n-1 multiplier design. The merit of the design is corroborated by synthesis results based on TSMC 0.18µm CMOS standard-cell library.**

***Index Terms*—Modulo multiplication, Residue Number System**

I. INTRODUCTION

Modulo 2^n-1 multiplication is used extensively in Residue Number System (RNS) based Digital Signal Processing (DSP) and cryptography units. High-speed modulo 2^n-1 multipliers with Carry Save Adder (CSA) tree and Booth encoding have been proposed in literature [1], [2]. The radix-4 Booth encoding technique is most prevalent as all required modulo-reduced partial products can be generated by circular-left-shift operation and bit-wise complementation, thereby minimizing the hardware complexity [1], [2]. The reduction in the number of partial products is determined by the radix of the Booth encoding technique employed. Reduction of partial products by more than half is possible with higher radix Booth encoding. Similar to the radix-4 algorithm, the radix-8 Booth encoding algorithm can be considered as a digit set conversion of four consecutive multiplier bits $y_{3i-1}y_{3i}y_{3i+1}y_{3i+2}$, $y_i \in \{0, 1\}$ from Y, to d_i, $d_i \in [-4, 4]$, for $i = 0, 1, \ldots \lfloor N/3 \rfloor$. The digit set conversion is given by

$$d_i = y_{3i-1} + y_{3i} + 2y_{3i+1} - 4y_{3i+2} \tag{1}$$

where y_{-1}, y_n, y_{n+1} and y_{n+2} are zero.

For the radix-8 Booth encoded modulo 2^n-1multiplier, the required modulo-reduced partial products are shown in Table I. From Table I, the necessary modulo-reduced partial products except $\pm 3X$ can be generated by circular-left-shift operation and/or bit-wise complementation of the multiplicand, X. The generation of $\pm 3X$ requires a large word-length adder which increases the critical path delay of the multiplier significantly. This delay penalty outweighs the advantages of using higher-radix Booth encoding scheme.

TABLE I. MODULO-REDUCED PARTIAL PRODUCTS FOR RADIX-8 BOOTH ENCODING

d_i	PP_i
0	00···00
+1	$x_{n-1-3i}x_{n-2-3i}\cdots x_0 x_{n-1}\cdots x_{n-3i}$
+2	$x_{n-2-3i}x_{n-3-3i}\cdots x_0 x_{n-1}\cdots x_{n-1-3i}$
+3	$+3X$
+4	$x_{n-3-3i}x_{n-4-3i}\cdots x_0 x_{n-1}\cdots x_{n-2-3i}$
–4	$\overline{x}_{n-3-3i}\overline{x}_{n-4-3i}\cdots \overline{x}_0 \overline{x}_{n-1}\cdots \overline{x}_{n-2-3i}$
–3	$-3X$
–2	$\overline{x}_{n-2-3i}\overline{x}_{n-3-3i}\cdots \overline{x}_0 \overline{x}_{n-1}\cdots \overline{x}_{n-1-3i}$
–1	$\overline{x}_{n-1-3i}\overline{x}_{n-2-3i}\cdots \overline{x}_0 \overline{x}_{n-1}\cdots \overline{x}_{n-3i}$
–0	11···11

Instead of using a conventional two-operand adder, an efficient modulo 2^n-1 hard multiple generator (HMG) for radix-8 Booth encoding algorithm is proposed in this paper. The carry equations of the adder are reformulated by considering modulo 2^n-1 addition of X and $2X$. The parallel-prefix formulation of the HMG is derived from the simplified carry equations. The proposed design calculates the hard multiple in $\lceil \log_2 n \rceil - 1$ prefix levels where $\lceil k \rceil$ denotes the smallest integer greater than or equal to k. We show that the proposed HMG results in significant area and delay savings as compared to the two-operand modulo 2^n-1 adders.

II. PRELIMINARY ON PARALLEL-PREFIX ADDERS

Carry propagation in binary addition can be considered as a prefix problem. A parallel-prefix structure with n inputs $a_1, a_2, \ldots a_n$ computes the n outputs $b_1, b_2, \ldots b_n$ in parallel, using an associative operator, $\odot$, as follows [1]:

$$\begin{aligned} b_1 &= a_1 \\ b_2 &= a_2 \odot a_1 \\ &\vdots \\ b_n &= a_n \odot a_{n-1} \odot \cdots \odot a_1 \end{aligned} \tag{2}$$

Let $A = \sum_{i=0}^{n-1} a_i \cdot 2^i$ and $B = \sum_{i=0}^{n-1} b_i \cdot 2^i$ be the addends. The sum and carry from bit position i, denoted as s_i and c_i, respectively are given by

$$c_i = a_i \cdot b_i + (a_i + b_i) \cdot c_{i-1}$$
$$s_i = a_i \oplus b_i \oplus c_{i-1} \tag{3}$$

The input carry, c_{-1} is assumed to be zero for simplicity. The s_i and c_i can be generated in three stages.

Preprocessing:

$$g_i = a_i \cdot b_i, p_i = a_i + b_i, h_i = a_i \oplus b_i \tag{4}$$

where g_i, p_i and h_i are the generate, propagate and half-sum signals, respectively at bit position i.

Prefix computation:

For $i = 0$

$$(G_{0:0}, P_{0:0}) = (g_0, p_0) \tag{5}$$

For $i = 1$ to $n-1$

$$\begin{aligned}(G_{i:0}, P_{i:0}) &= (g_i, p_i) \bullet (G_{i-1:0}, P_{i-1:0}) \\ &= (g_i + p_i \cdot G_{i-1:0}, p_i \cdot P_{i-1:0}) \\ &= (g_i, p_i) \bullet (g_{i-1}, p_{i-1}) \bullet \cdots \bullet (g_0, p_0)\end{aligned} \tag{6}$$

Postprocessing:

$$c_i = G_{i:0} \text{ and } s_i = h_i \oplus c_{i-1} \tag{7}$$

Numerous prefix structures that provide tradeoffs between the number of logic levels, maximum fanout and wiring tracks exist in literature [3]. Fig. 1 shows a parallel-prefix adder that employs Kogge-Stone structure with minimum logic depth. In Fig. 1, the nodes □ and ◊ represent the preprocessing and postprocessing units, respectively. The solid node (●) performs the prefix computation while the hollow node (○) acts as a buffer [3].

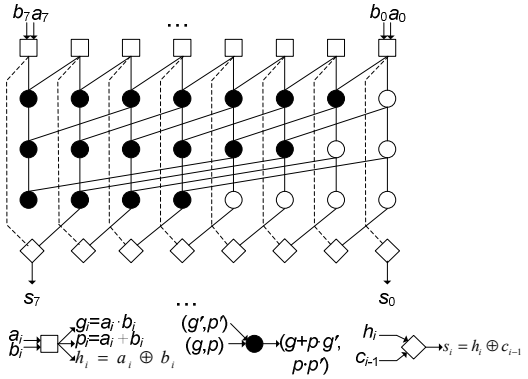

Figure 1. Parallel-prefix binary adder

III. MODULO 2^N–1 ADDERS

Modulo 2^n–1 arithmetic possesses numerous number theoretic properties that can be exploited to simplify arithmetic circuits. Let $CLS(X,j)$ denote the circular-left-shift of $X = \sum_{i=0}^{n-1} x_i \cdot 2^i$ by j bit positions. *Property 1* shows that the power-of-two multiplication of X modulo 2^n–1 is equivalent to a *CLS* operation [4].

Property 1: For $j < n$

$$\left| 2^j \cdot X \right|_{2^n-1} = CLS(X, j) \tag{8}$$

Corollary 1:

$$2 \cdot X = CLS(X, 1) = \sum_{i=0}^{n-2} x_i \cdot 2^{i+1} + x_{n-1} \cdot 2^0 \tag{9}$$

Assuming that zero has a dual representation, *i.e.*, $\underbrace{00\cdots00}_{n}$ and $\underbrace{11\cdots11}_{n}$, *Property 2* shows that modulo 2^n–1 addition is equivalent to an n-bit end-around-carry addition [1].

Property 2:

$$\left| A + B \right|_{2^n-1} = \left| A + B + c_{out} \right|_{2^n} \tag{10}$$

In [1], a parallel-prefix modulo 2^n–1 adder that employs an additional level of ● nodes for addition of c_{out}, as shown in Fig. 2, was proposed. The delay of this adder is undermined by the high fanout of the reentrant carry. A modulo 2^n–1 adder with only $\lceil \log_2 n \rceil$ prefix levels and carry recirculation at each level, as illustrated in Fig. 3, was proposed in [5].

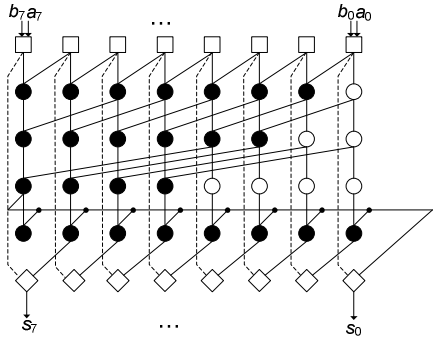

Figure 2. Parallel-prefix modulo 2^n–1 adder [1]

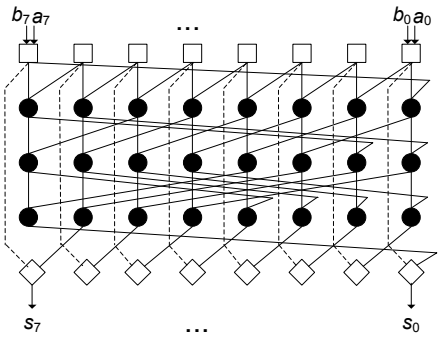

Figure 3. Parallel-prefix modulo 2^n–1 adder [5]

In [5], the carry equation for modulo 2^n–1 addition was shown to be:

$$\begin{aligned}c_i &= (G_{i:0}, P_{i:0}) \bullet (G_{n-1:i+1}, P_{n-1:i+1}) \\ &= (g_i, p_i) \bullet \cdots \bullet (g_0, p_0) \bullet (g_{n-1}, p_{n-1}) \bullet \cdots \bullet (g_{i+1}, p_{i+1})\end{aligned} \tag{11}$$

The Ling carry based modulo 2^n–1 adder proposed in [6] is illustrated in Fig. 4. The parallel-prefix computation of modulo

2^n–1 Ling carries requires fewer prefix levels. However, the preprocessing stage uses AND-OR logic, while the postprocessing stage uses 2:1 multiplexers (MUX) and XOR gates to generate the sum bits from the Ling carries.

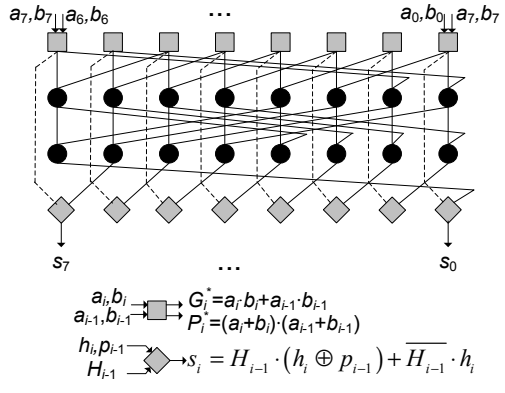

Figure 4. Parallel-prefix modulo 2^n–1 adder using Ling carries

IV. Parallel-prefix addition for modulo 2^N–1 hard multiple generation

A high-speed adder design that computes the hard multiple of the radix-8 Booth encoded modulo 2^n–1 multiplier is proposed. The architecture of the HMG is derived for n = 8 for illustration but similar derivation is applicable to any n. Let the binary representation of the multiplicand X be denoted by ($x_7x_6x_5x_4x_3x_2x_1x_0$). By *Corollary 1*, $2X$ is represented as ($x_6x_5x_4x_3x_2x_1x_0x_7$). To compute the hard multiple $+3X$, the generate and propagate signals for modulo 2^n–1 addition of X and $2X$ are given by

$$(g_0, p_0) = (x_0 \cdot x_7, x_0 + x_7), \ldots, (g_7, p_7) = (x_7 \cdot x_6, x_7 + x_6) \tag{12}$$

Hence,

$$p_i \cdot g_{|i-1|_n} = g_{|i-1|_n} \tag{13}$$

From (11), c_0 can be calculated by

$$\begin{aligned} c_0 &= (g_0, p_0) \bullet (g_7, p_7) \bullet (g_6, p_6) \bullet (g_5, p_5) \\ &\quad \bullet (g_4, p_4) \bullet (g_3, p_3) \bullet (g_2, p_2) \bullet (g_1, p_1) \\ &= g_0 + p_0 \cdot g_7 + p_0 \cdot p_7 \cdot g_6 + p_0 \cdot p_7 \cdot p_6 \cdot g_5 \\ &\quad + p_0 \cdot p_7 \cdot p_6 \cdot p_5 \cdot g_4 + p_0 \cdot p_7 \cdot p_6 \cdot p_5 \cdot p_4 \cdot g_3 \\ &\quad + p_0 \cdot p_7 \cdot p_6 \cdot p_5 \cdot p_4 \cdot p_3 \cdot g_2 \\ &\quad + p_0 \cdot p_7 \cdot p_6 \cdot p_5 \cdot p_4 \cdot p_3 \cdot p_2 \cdot g_1 \end{aligned} \tag{14}$$

From (13), (14) can be reformulated to:

$$\begin{aligned} c_0 &= (g_0 + g_7) + (p_0 \cdot p_7 \cdot g_6 + p_0 \cdot p_7 \cdot g_5) \\ &\quad + (p_0 \cdot p_7 \cdot p_6 \cdot p_5 \cdot g_4 + p_0 \cdot p_7 \cdot p_6 \cdot p_5 \cdot g_3) + \\ &\quad (p_0 \cdot p_7 \cdot p_6 \cdot p_5 \cdot p_4 \cdot p_3 \cdot g_2 + p_0 \cdot p_7 \cdot p_6 \cdot p_5 \cdot p_4 \cdot p_3 \cdot g_1) \end{aligned} \tag{15}$$

$$c_0 = G_0^* + (P_0^* \cdot G_6^*) + (P_0^* \cdot P_6^* \cdot G_4^*) + (P_0^* \cdot P_6^* \cdot P_4^* \cdot G_2^*) \tag{16}$$

$$c_0 = (G_0^*, P_0^*) \bullet (G_6^*, P_6^*) \bullet (G_4^*, P_4^*) \bullet (G_2^*, P_2^*) \tag{17}$$

where

$$\begin{aligned} G_i^* &= g_i + g_{|i-1|_n} = x_{|i-1|_n} \cdot (x_i + x_{|i-2|_n}) \\ P_i^* &= p_i \cdot p_{|i-1|_n} = x_{|i-1_n|} + x_i \cdot x_{|i-2|_n} \end{aligned} \tag{18}$$

The carry, c_i for modulo 2^8–1 addition of X and $2X$ is generated as follows.

$$c_i = (G_i^*, P_i^*) \bullet (G_{|i-2|_8}^*, P_{|i-2|_8}^*) \bullet (G_{|i-4|_8}^*, P_{|i-4|_8}^*) \bullet (G_{|i-6|_8}^*, P_{|i-6|_8}^*) \tag{19}$$

The proposed HMG consists of three stages. In the Preprocessing stage, the modified generate G_i^* and propagate P_i^* signals, are generated using OR-AND and AND-OR logic, respectively. The half-sum bits h_i are generated using XOR gates. In the Prefix stage, the carry c_i is generated from the modified generate and propagate pairs using prefix operation in $\lceil \log_2 n \rceil - 1$ levels as shown in Fig. 5. Finally, in the Postprocessing stage, the sum bits are generated from h_i and c_{i-1} using XOR gates.

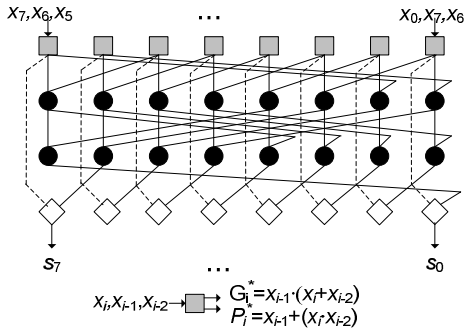

Figure 5. Proposed parallel-prefix modulo 2^n-1 HMG

The proposed HMG and the adder based on Ling carries use $\lceil \log_2 n \rceil - 1$ prefix levels to calculate the n carry bits. However, the proposed design employs traditional carry to eliminate the hardware overhead in the preprocessing and postprocessing stages of [6].

V. Performance comparison

In this section we evaluate and compare the performance of the proposed modulo 2^n–1 HMG with the parallel-prefix modulo 2^n–1 adders of [1], [5] and [6].

We first compare the area and delay analytically using the unit-gate model [7]. In this model, the area and delay of a two-input logic gate are considered as one unit, except XOR and XNOR, the area and delay of which are counted as two units. The area and delay of a prefix operator (•) is 3 and 2 units, respectively.

The generation of the signals G_i^* and P_i^* of the proposed design in the preprocessing stage requires $2n$ AND and $2n$ OR gates. The generation of h_i signals require n XOR gates. The

prefix stage employs $\left(\lceil \log_2 n \rceil - 1\right) n$ prefix operators. The postprocessing stage employs n XOR gates. Therefore, the area and delay of the proposed HMG can be modeled as

$$\begin{aligned} A_{\text{Proposed}} &= \left(2n + 2n + 2n\right) + \left(\lceil \log_2 n \rceil - 1\right) 3n + 2n \\ &= 3n\left(\lceil \log_2 n \rceil\right) + 5n \end{aligned} \tag{20}$$

$$D_{\text{Proposed}} = 2 + \left(\lceil \log_2 n \rceil - 1\right) 2 + 2 = 2\left(\lceil \log_2 n \rceil\right) + 2 \tag{21}$$

From Figs. 2 and 3, the area and delay of [1] and [5] can be modeled as follows.

$$\begin{aligned} A_{[1]} &= \left(n + n + 2n\right) + 3\left[n\left(\lceil \log_2 n \rceil\right) - (n-1)\right] + 3n + 2n \\ &= 3n\left(\lceil \log_2 n \rceil\right) + 6n - 3 \end{aligned} \tag{22}$$

$$D_{[1]} = 2 + 2\left(\lceil \log_2 n \rceil + 1\right) + 2 = 2\left(\lceil \log_2 n \rceil\right) + 6 \tag{23}$$

$$\begin{aligned} A_{[5]} &= \left(n + n + 2n\right) + 3n\left(\lceil \log_2 n \rceil\right) + 2n \\ &= 3n\left(\lceil \log_2 n \rceil\right) + 6n \end{aligned} \tag{24}$$

$$D_{[5]} = 2 + 2\lceil \log_2 n \rceil + 2 = 2\left(\lceil \log_2 n \rceil\right) + 4 \tag{25}$$

From Fig. 4, the preprocessing stage uses $3n$ AND and $3n$ OR gates, in addition to the n XOR gates that generate the h_i bits. The area and delay of a 2:1 MUX is assumed to be 3 and 2 units, respectively. Hence, from Fig. 4, the area and delay of the postprocessing stage is $5n$ and 4, respectively. The area and delay of [6] are given by

$$\begin{aligned} A_{[6]} &= \left(3n + 3n + 2n\right) + \left(\lceil \log_2 n \rceil - 1\right) 3n + 5n \\ &= 3n\left(\lceil \log_2 n \rceil\right) + 10n \end{aligned} \tag{26}$$

$$D_{[6]} = 2 + \left(\lceil \log_2 n \rceil - 1\right) 2 + 4 = 2\left(\lceil \log_2 n \rceil\right) + 4 \tag{27}$$

The unit-gate model analysis shows that the proposed generator has lower area and delay complexity. The VLSI metrics are also evaluated and compared by describing all designs in VHDL, synthesizing using Synopsys Design Compiler (V-2004.06-SP2) and mapping to TSMC 0.18μm 1.8V standard-cell CMOS library. The designs were optimized for minimum achievable area and delay independently under the same nominal environment of 25ºC and 1.8V. The area-optimized and delay-optimized synthesis results are shown in Tables II and III, respectively. Both area (in μm^2) and delay (in ns) are listed under each type of optimization constraint to ensure that the secondary performance metric is not compromised.

From Table II, for n = 8, the proposed design reduces the hardware by approximately 20% when compared to [1], [5] and [6]. Similarly, for n = 64, there is at least 12% area reduction over [1], [5] and [6].

From Tables II and III, it can be seen that the delay of all designs is a logarithmic function of n. For example, for n in the range of [40, 64], the delay is almost constant for HMG, [5] and [6]. Even though the delay of [1] is logarithmically dependent on n, the high fanout of the reentrant carry results in inconsistent delay values. Furthermore, from Table II, the delays of [5] and [6] are identical. This observation is consistent with the delay modeling using unit-gate model, as shown in (25) and (27).

From Table III, for n = 8, the proposed HMG reduces the critical path delay by 33%, 16% and 12% compared to [1], [5] and [6], respectively. For n = 64, the delay is reduced by 25% over [1] and by 10% over [5] and [6].

TABLE II. AREA-OPTIMIZED SYNTHESIS RESULTS

n	Proposed		[1]		[5]		[6]	
	Area	Delay	Area	Delay	Area	Delay	Area	Delay
8	1144	0.73	1443	1.07	1410	0.72	1437	0.80
16	2820	0.89	3386	1.46	3353	0.88	3406	0.95
24	5029	1.04	5595	1.75	5827	1.03	5907	1.11
32	6706	1.04	7803	2.04	7770	1.03	7876	1.11
40	9713	1.19	10278	2.18	11043	1.18	11176	1.26
48	11655	1.19	12753	2.49	13252	1.18	13412	1.26
56	13598	1.19	15228	2.63	15461	1.18	15647	1.26
64	15540	1.19	17703	2.94	17669	1.18	17882	1.26

TABLE III. DELAY-OPTIMIZED SYNTHESIS RESULTS

n	Proposed		[1]		[5]		[6]	
	Area	Delay	Area	Delay	Area	Delay	Area	Delay
8	2408	0.43	2750	0.64	2737	0.51	3233	0.49
16	5498	0.53	5907	0.75	6533	0.61	6895	0.61
24	9590	0.65	9829	0.83	10907	0.73	12008	0.72
32	12377	0.65	13950	0.92	13960	0.73	15577	0.73
40	17676	0.76	17696	0.99	20384	0.84	21056	0.83
48	20863	0.76	21731	0.97	22928	0.84	25496	0.85
56	25277	0.77	25503	0.98	26814	0.84	29711	0.84
64	29086	0.76	30177	1.02	30739	0.84	33224	0.84

VI. CONCLUSION

In radix-8 Booth encoded modulo 2^n–1 multiplier, the generation of the hard multiple degrades the performance of the multiplier due to the long word-length carry propagation adder. An efficient parallel-prefix modulo 2^n–1 HMG was proposed in this paper. The performance of the proposed HMG was evaluated against existing modulo 2^n–1 adders both analytically and by synthesis results. It was found that the proposed HMG significantly reduces the area and delay complexity for any n.

REFERENCES

[1] R. Zimmermann, "Efficient VLSI implementation of modulo ($2^n \pm 1$) addition and multiplication," *Proc. IEEE Symp. on Computer Arithmetic*, Adelaide, Australia, pp. 158-167, Apr. 1999.

[2] C. Efstathiou, H. T. Vergos and D. Nikolos, "Modified Booth modulo 2^n-1 multipliers," *IEEE Trans. on Computers*, vol. 53, no. 3, pp. 370-374, Mar. 2004.

[3]] D. Harris, "A taxonomy of parallel prefix networks," *Proc. 37th Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, California, pp. 2213-2217, Nov. 2003.

[4] N. S. Szabo and R. I. Tanka, *Residue Arithmetic and its Applications to Computer Technology*, McGraw-Hill, 1967.

[5] L. Kalampoukas et al., "High-speed parallel-prefix modulo (2^n–1) adders," *IEEE Trans. on Computers*, vol. 49, no. 7, pp. 673-680, July 2000.

[6] G. Dimitrakopoulos et al., "New architectures for modulo 2^n-1 adders," *IEEE Intl. Conf. on Electronics, Circuits and Systems*, Gammarth, Tunisia, pp.1-4, Dec. 2005.

[7] A. Tyagi, "A reduced-area scheme for carry-select adders," *IEEE Trans. on Computers*, vol. 42, no. 10, pp. 1163-1170, Oct 1993.