NEW ARCHITECTURES FOR MODULO $2^N - 1$ ADDERS

G. Dimitrakopoulos, D. G. Nikolos, H. T. Vergos, D. Nikolos

Technology and Computer Architecture Lab
Computer Engineering and Informatics Dept.
University of Patras, 26500, Patras, Greece.

C. Efstathiou

Informatics Department
ATEI of Athens,
12210 Egaleo, Athens, Greece

ABSTRACT

Two architectures for parallel-prefix modulo $2^n - 1$ adders are presented in this paper. For large wordlengths we introduce the sparse modulo $2^n - 1$ adders that achieve significant reduction of the wiring complexity without imposing any delay penalty. Then, the Ling-carry formulation of modulo $2^n - 1$ addition is presented. Ling modulo adders save one logic level of implementation and provide high-speed solutions for smaller adder widths, where wiring complexity is small. The performance of the proposed adders has been validated with static CMOS implementations. In all examined cases, the proposed designs achieve significant savings in both area and delay compared to previously published architectures.

1. INTRODUCTION

Modulo $2^n - 1$, or equivalently one's complement addition, plays an essential role in Residue Number System applications [1], in fault-tolerant computer systems [2], in error detection in computer networks [3], and in floating-point arithmetic [4].

Many solutions have been presented for fast modulo $2^n - 1$ addition. In [5] modulo adders are proposed that use a parallel-prefix carry computation unit along with an extra prefix level that handles the end-around-carry. In [6] it was shown that the recirculation of the end-around-carry can be performed within the existing prefix levels. Therefore, the need of the extra prefix level is cancelled and parallel-prefix modulo $2^n - 1$ adders are derived that can perform carry computation in $\log_2 n$ levels. However, the routing requirements are increased. Finally, in [7] select-prefix modulo $2^n - 1$ adders have been proposed, that aim at reducing the area complexity of the parallel-prefix structures but suffer from significant delay penalties.

In this paper two new architectures are presented. The first one, is based on the architecture of [6], and allows the design of sparse modulo $2^n - 1$ adders with reduced wiring complexity. The proposed sparse adders are equally fast with the adders of [6] and since they do not suffer from increased routing are better suited for large wordlengths. In the second case, the Ling-carry formulation of modulo $2^n - 1$ addition is proposed for the first time in the open literature. Ling adders [8] are based on a simplified form of the carry-lookahead equations and allow faster implementations [9]. The proposed Ling-based modulo adders offer significant delay reductions compared to the adders of [6], since they save one logic level of implementation, without increasing wiring complexity.

The rest of the manuscript is organized as follows. Section 2 revisits the basics of parallel-prefix addition and sparse adder design, and provides a brief review of modulo $2^n - 1$ adder architectures. Section 3 presents the proposed sparse modulo adders. Section 4 describes the design of Ling-based modulo adders, while experimental results that validate the efficiency of the proposed architectures, are given in Section 5. Finally, conclusions are drawn in Section 6.

We thank the European Social Fund (ESF), Operational Program for Educational and Vocational Training II (EPEAEK II), and particularly the Program PYTHAGORAS, for funding the above work.

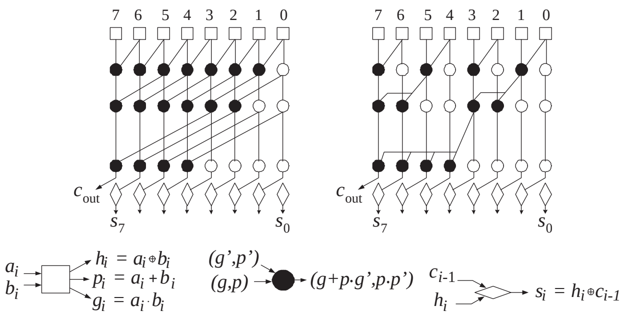


Figure 1. Examples of parallel-prefix structures for integer adders.

2. PARALLEL-PREFIX ADDITION

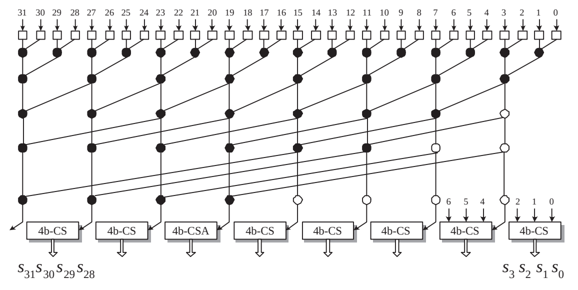
Suppose that $A = a_{n-1}a_{n-2}\ldots a_0$ and $B = b_{n-1}b_{n-2}\ldots b_0$ represent the two numbers to be added and $S = s_{n-1}s_{n-2}\ldots s_0$ denotes their sum. An adder can be considered as a three-stage circuit. The preprocessing stage computes the carry-generate bits g_i, the carry-propagate bits p_i, and the half-sum bits h_i, for every i, $0 \leq i \leq n-1$, according to: $g_i = a_i \cdot b_i$, $p_i = a_i + b_i$, $h_i = a_i \oplus b_i$, where $\cdot$, $+$, and $\oplus$ denote the logical AND, OR and exclusive OR operations respectively. The second stage of the adder, hereafter called the carry-computation unit, computes the carry signals c_i, for $0 \leq i \leq n-1$ using the carry generate and carry propagate bits g_i and p_i. The third stage computes the sum bits according to $s_i = h_i \oplus c_{i-1}$.

Carry computation is transformed into a parallel prefix problem using the $\circ$ operator, which associates pairs of generate and propagate signals and was defined in [10] as, $(g, p) \circ (g', p') = (g + p \cdot g', p \cdot p')$. In a series of associations of consecutive generate/propagate pairs (g, p) the notation $(G_{k:j}, P_{k:j})$, is used to denote the group generate/propagate term produced out of bits $k, k-1, \ldots, j$, that is,

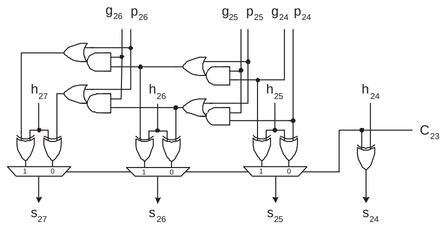
$$(G_{k:j}, P_{k:j}) = (g_k, p_k) \circ (g_{k-1}, p_{k-1}) \circ \ldots \circ (g_j, p_j). \quad (1)$$

The Kogge-Stone [11] and Ladner-Fisher [12] parallel-prefix structures, for an 8-bit adder, are shown in Fig. 1.

For large wordlengths ($n > 32$) the design of sparse parallel-prefix adders is preferred, since the wiring and area of the design are significantly reduced without sacrificing delay [1]. The design of sparse adders relies on the use of a sparse parallel-prefix carry computation unit and carry-select (CS) blocks. Only the



(a)



(b)

Figure 2. (a) Sparse-4 parallel-prefix structure for a 32-bit integer adder and (b) the logic level implementation of the CS block.

carries at the boundaries of the carry-select blocks are computed, saving considerable amount of area in the carry-computation unit. A 32-bit adder with sparseness of 4 bits is shown in Fig. 2(a). The carry select block computes two sets of sum bits corresponding to the two possible values of the incoming carry. A logic-level implementation of a 4-bit carry-select block is shown in Fig. 2(b). Since the two candidate sum bits are computed earlier than the selecting carry, no extra delay is imposed by the use of the carry-select blocks.

2.1. Modulo $2^N - 1$ adders

The computation of modulo $2^n - 1$ addition is in fact a conditional operation defined as,

$$(A+B) \bmod (2^n - 1) = \begin{cases} (A+B) \bmod 2^n, & A+B < 2^n \\ (A+B) \bmod 2^n + 1, & A+B \geq 2^n \end{cases}$$

or equivalently,

$$(A+B) \bmod (2^n - 1) = (A+B) \bmod 2^n + c_{n-1}.$$

According to [5], the addition of the carry output c_{n-1} of the integer adder to the sum $(A+B)$ mod2^n, can be performed using an additional carry increment stage as shown in Fig. 3(a). In this case, one extra level of • cells is required that are driven by the carry output of the adder. This approach suffers from large delay since both one extra prefix level is required, and the re-entrant carry has a fanout proportional to the adder's wordlength.

In [6] an alternative approach has been presented. It was shown that the ith carry c_i in the case of modulo $2^n - 1$ addition can be expressed as

$$c_i = G_{i:0} + P_{i:0}\, G_{n-1:i+1}\,, \tag{2}$$

which is computed using the $\circ$ operator as,

$$(g_i, p_i) \circ \cdots \circ (g_0, p_0) \circ (g_{n-1}, p_{n-1}) \circ \cdots \circ (g_{i+1}, p_{i+1}).$$

Using this formulation, the carries can be computed by recirculating the intermediate generate and propagate terms in the existing prefix levels of a parallel-prefix tree. The resulting adder is shown in Fig. 3(b). The adders of [6] are faster than those proposed in [5], but suffer from excessive wiring that limits their use to small wordlengths. This problem is alleviated by the new sparse adder topologies presented in the next section.

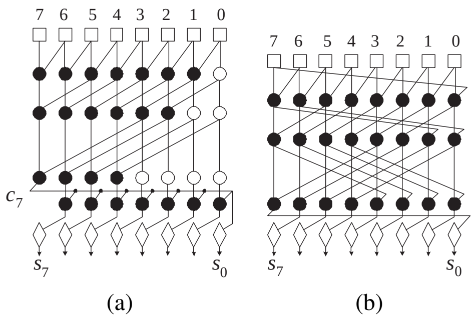


Figure 3. Modulo $2^8 - 1$ adder designs.

3. SPARSE MODULO $2^N - 1$ ADDERS

The proposed methodology for the design of sparse parallel-prefix modulo $2^n - 1$ adders is presented via an example. Assume that we need to design a sparse-4 parallel-prefix modulo $2^{32}-1$ adder. In this case, only one every four carries is generated. Therefore, in order to design the carry-select block, we need to associate the rest of the carries with the available ones.

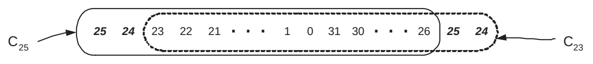


Figure 4. Circular overlapping between the terms of c_{23} and c_{25}.

For example the carry $c_{25} = G_{25:0} + P_{25:0}\, G_{31:26}$ (see (2)), needs to be derived based on $c_{23} = G_{23:0} + P_{23:0}\, G_{31:24}$ in order to generate s_{26}. We will show that although c_{23} contains the group generate term $G_{25:24}$ which partially overlaps with the group generate term $G_{25:0}$ of c_{25}, in a circular manner, as shown in Fig. 4, it is still possible to produce c_{25} from c_{23}.

Since, according to (1), $G_{25:0} = G_{25:24} + P_{25:24}\, G_{23:0}$ and $P_{25:0} = P_{25:24}\, P_{23:0}$, carry c_{25} can be written as:

$$c_{25} = G_{25:24} + P_{25:24}\, G_{23:0} + P_{25:24}\, P_{23:0}\, G_{31:26}. \tag{3}$$

The term $G_{25:24}$ can be expanded to

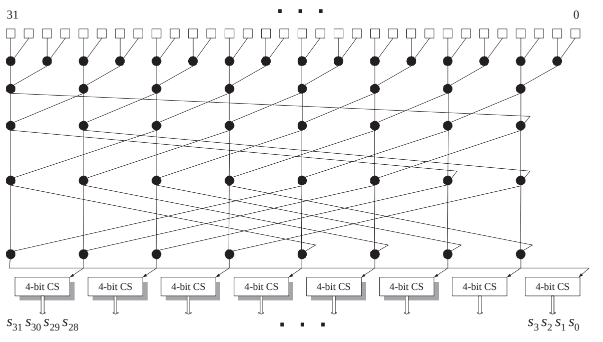
$$G_{25:24} + G_{25:24}\,(P_{25:24}\, P_{23:0}\, P_{31:26}). \tag{4}$$

Replacing (4) in (3) and factoring the common terms of the resulting relation it is derived that

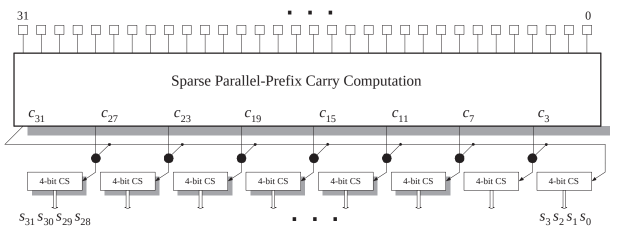
$$\begin{aligned} c_{25} &= G_{25:24} + P_{25:24}\,(G_{23:0} + P_{23:0}\,(G_{31:26} + P_{31:26}\, G_{25:24})) \\ &= G_{25:24} + P_{25:24}\,(G_{23:0} + P_{23:0}\, G_{31:24}) \\ &= G_{25:24} + P_{25:24}\, c_{23}. \end{aligned} \tag{5}$$

From (5) it is derived that although carries c_{25} and c_{23} refer to modulo $2^n - 1$ addition, the relation that associates them is the same as in the case of integer addition. Therefore, the carry select block used for integer adders can be used without modifications for the design of sparse modulo $2^n - 1$ adders. The resulting design is shown in Fig. 5(a). The same principle can be applied to the design of sparse modulo adders with a carry increment stage (see Fig. 5(b)). In this case, both the area and the large fanout of the re-entrant carry are reduced compared to the structures proposed in [5].

Equation (5) is the inverse application of the circular idempotency property, which holds for modulo $2^n - 1$ addition, and has been used in [13] for designing modulo adders for every n.



(a)



(b)

Figure 5. The proposed 32-bit sparse modulo adders using the architectures of [6] and [5], respectively.

4. LING MODULO $2^N - 1$ ADDERS

Ling's definition of carry-lookahead equations [8], simplifies carry computation and leads to faster adder architectures. In this section, we present the Ling-based formulation of modulo $2^n - 1$ addition, while the corresponding parallel-prefix structures are obtained using the architecture presented in [9]. In order to describe the proposed approach, a modulo $2^8 - 1$ adder will be used as an example.

According to (2) the carry c_2 of a modulo $2^8 - 1$ adder is equal to:

$$\begin{aligned} c_2 =& g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 g_7 + \dots \\ & \dots + p_2 p_1 p_0 p_7 p_6 p_5 p_4 g_3 \end{aligned} \quad (6)$$

Since $g_i = p_i \cdot g_i$ relation (6) is expressed as

$$\begin{aligned} c_2 = & p_2(g_2 + g_1 + p_1 g_0 + p_1 p_0 g_7 + \dots \\ & \dots + p_1 p_0 p_7 p_6 p_5 p_4 g_3) = p_2 H_2. \end{aligned} \quad (7)$$

The term H_2 can be considered as the Ling-carry equivalent for the case of modulo $2^n - 1$ addition. The terms of H_2 can be further grouped as follows

$$\begin{aligned} H_2 =& (g_2 + g_1) + (p_1 p_0)(g_0 + g_7) + (p_1 p_0)(p_7 p_6)(g_6 + g_5) + \\ & + (p_1 p_0)(p_7 p_6)(p_5 p_4)(g_4 + g_3). \end{aligned} \quad (8)$$

Assuming that

$$G_i^* = g_i + g_{i-1} \text{ and } P_i^* = p_i \cdot p_{i-1}, \quad (9)$$

where $g_{-1} = g_7$ and $p_{-1} = p_7$ equation (8) can be written as,

$$H_2 = G_2^* + P_1^* G_0^* + P_1^* P_7^* G_6^* + P_1^* P_7^* P_5^* G_4^* \quad (10)$$

which can be equivalently expressed using the $\circ$ operator as

$$H_2 \leftrightarrow (G_2^*, P_1^*) \circ (G_0^*, P_7^*) \circ (G_6^*, P_5^*) \circ (G_4^*, P_3^*) \quad (11)$$

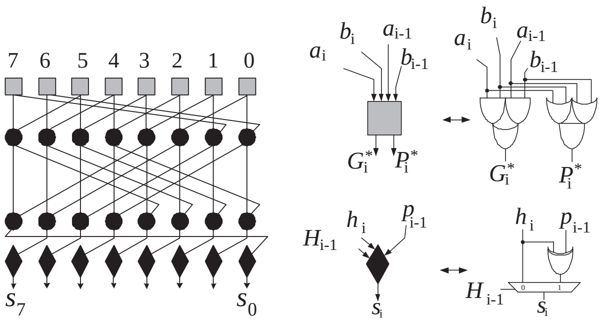


Figure 6. The proposed Ling-based modulo $2^8 - 1$ adder and the definition of the new preprocessing and sum bits computation stages.

The corresponding expressions for the rest Ling modulo carries H_i can be derived in a similar manner.

$$\begin{aligned} H_0 &\leftrightarrow (G_0^*, P_7^*) \circ (G_6^*, P_5^*) \circ (G_4^*, P_3^*) \circ (G_2^*, P_1^*) \\ H_2 &\leftrightarrow (G_2^*, P_1^*) \circ (G_0^*, P_7^*) \circ (G_6^*, P_5^*) \circ (G_4^*, P_3^*) \\ H_4 &\leftrightarrow (G_4^*, P_3^*) \circ (G_2^*, P_1^*) \circ (G_0^*, P_7^*) \circ (G_6^*, P_5^*) \\ H_6 &\leftrightarrow (G_6^*, P_5^*) \circ (G_4^*, P_3^*) \circ (G_2^*, P_1^*) \circ (G_0^*, P_7^*) \end{aligned}$$

$$\begin{aligned} H_1 &\leftrightarrow (G_1^*, P_0^*) \circ (G_7^*, P_6^*) \circ (G_5^*, P_4^*) \circ (G_3^*, P_2^*) \\ H_3 &\leftrightarrow (G_3^*, P_2^*) \circ (G_1^*, P_0^*) \circ (G_7^*, P_6^*) \circ (G_5^*, P_4^*) \\ H_5 &\leftrightarrow (G_5^*, P_4^*) \circ (G_3^*, P_2^*) \circ (G_1^*, P_0^*) \circ (G_7^*, P_6^*) \\ H_7 &\leftrightarrow (G_7^*, P_6^*) \circ (G_5^*, P_4^*) \circ (G_3^*, P_2^*) \circ (G_1^*, P_0^*) \end{aligned}$$

It is evident that using the pairs (G_i^*, P_{i-1}^*), where $G_{-1}^* = G_7^*$ and $P_{-1}^* = P_7^*$, the Ling modulo carries of the even and the odd-indexed bit positions can be computed independently. The parallel-prefix architecture for the Ling-based modulo $2^8 - 1$ adder is shown in Fig. 6. In general, the Ling-carries in the case of modulo $2^n - 1$ addition can be defined as follows:

$$H_i \leftrightarrow (G_i^*, P_{i-1}^*) \circ (G_{i-2}^*, P_{i-3}^*) \circ \dots \circ (G_{i+2}^*, P_{i+1}^*)$$

It should be noted that the parallel prefix computation of the modulo $2^n - 1$ Ling carries H_i requires one less prefix level, compared to the adders shown in Fig. 3(b). The pairs (G_i^*, P_{i-1}^*) are directly computed in a single logic level from the input bits (a_i, b_i) and (a_{i-1}, b_{i-1}) using AND-OR gates according to (9). The sum bits s_i can be computed using a multiplexer that selects either h_i or $(h_i \oplus p_{i-1})$ according to the value of H_{i-1} [9], since

$$s_i = \overline{H_{i-1}} \cdot h_i + H_{i-1} \cdot (h_i \oplus p_{i-1}), \quad (12)$$

where $\overline{x}$ denotes the complement of x. Taking into account that, in general, an XOR gate has almost the same delay as a multiplexer and that both h_i and $(h_i \oplus p_{i-1})$ are computed in fewer logic levels than H_{i-1}, it is evident that no extra delay is imposed for the calculation of the output bits from the Ling carries.

5. EXPERIMENTAL RESULTS

In this section we present a complete experimental investigation of the performance of the proposed adders and the ones presented in [5] and [6]. All adders were described in Verilog HDL and mapped on a 0.18μm technology library [14] assuming typical conditions (1.8V, 25°C), and using the Synopsys Design Compiler. Each design was iteratively optimized, until no further delay optimizations were possible. The same design constraints,

Table 1. The area (μm^2) and delay (ns) results for the proposed Ling-based modulo adders and the adders of [6] and [5].

bits		Prop. Ling	[6]	Red (%)	[5]	Red (%)
8	Delay	0.51	0.60	15	0.73	30
	Area	2353	2402	2	2504	6
16	Delay	0.63	0.71	11	0.89	29
	Area	5696	5898	3	5834	2

Table 2. The area (μm^2) and delay (ns) results for the adders of [5] and [6] and their proposed sparse counterparts.

bits		Prop. Sparse [6]	[6]	Red (%)
32	Delay	0.84	0.84	0
	Area	10815	13363	19
64	Delay	0.95	0.95	0
	Area	21991	31002	29

(a)

bits		Prop. Sparse [5]	[5]	Red (%)
32	Delay	0.99	1.02	2
	Area	10583	13347	20
64	Delay	1.11	1.18	5
	Area	21508	28783	25

(b)

bits		Prop. Sparse [6]	Prop. Sparse [5]	Red (%)
32	Delay	0.84	0.99	15
	Area	10815	10583	-2
64	Delay	0.95	1.11	14
	Area	21991	21508	-2

(c)

such as maximum fanout, output capacitance, and available input drive strength, were used for all designs.

The results for the 8 and 16-bits adders are shown in Table 1. The proposed Ling-based modulo $2^n - 1$ adders outperform all other solutions both in terms of delay and area by an average of 21.41% and 3.47% respectively. For the adders proposed in [5] the Kogge-Stone parallel prefix structure has been used, since it offers the minimum delay.

The sparse structures presented in Section 3 are better suited for large adder wordlengths. Therefore we implemented 32 and 64-bits sparse adders utilizing 4-bit carry-select blocks. In Tables 2(a) and 2(b), the adders proposed in [5] and [6] are compared with their sparse counterparts presented in Section 3. The new derived structures are more efficient both in terms of delay and area. Among them the proposed sparse variants of [6], as the one shown in Fig. 5(a), are the most efficient in the area-time sense. They offer an average of 24% area reduction compared to the adders of [6] without any delay penalty (see Table 2(a)). Also, as shown in Table 2(c), they require almost the same implementation area with the proposed sparse modulo adder that utilizes an extra carry-increment stage (Fig. 5(b)), while being faster by 14% on average.

As a final step, we compared the proposed Ling modulo adders with the proposed sparse variants of [6], for 32 and 64-bit adder widths. The results are shown in Table 3(a). The Ling-based adders are faster but require significantly more implementation area, more than 18%. In order to better compare the two proposed solutions in the area-time space, the Ling-based adders were re-optimized targeting the delay of the sparse variants of [6]. The more relaxed delay constraint for the proposed Ling adders resulted in more area efficient designs. However, as shown in Table 3(b), the proposed sparse variants of [6] still require at least 14% less area for their implementation, thus providing an attractive solution for large adder widths. Ling-based adders remain the most efficient solution for smaller wordlengths.

Table 3. The area (μm^2) and delay (ns) results for the proposed sparse variants of [6] and the new Ling-based modulo adders.

bits		Prop. Sparse [6]	Prop. Ling	Red (%)
32	Delay	0.84	0.76	-9
	Area	10815	13339	18
64	Delay	0.95	0.87	-9
	Area	21991	31044	29

(a)

bits		Prop. Sparse [6]	Prop. Ling	Red (%)
32	Delay	0.84	0.84	0
	Area	10815	12606	14
64	Delay	0.95	0.95	0
	Area	21991	29165	24

(b)

6. CONCLUSIONS

In this paper we have presented two new architectures for the case of modulo $2^n - 1$ addition that outperform previously published designs in both area and delay. The efficiency of the proposed designs has been validated using static CMOS implementations. For large wordlengths area-time efficient sparse modulo $2^n - 1$ adders have been proposed. Also for smaller adder widths, where wiring complexity is small, Ling-based modulo adders have been introduced, and offer the fastest implementations.

7. REFERENCES

[1] I. Koren, *Computer Arithmetic Algorithms*, Prentice-Hall, 1993.

[2] T. R. N. Rao and E. Fujiwara, *Error Control Coding for Computer Systems*, Prentice-Hall, 1989.

[3] F. Halsall, *Data Communications, Computer Networks and Open Systems*, Addison Wesley, 1996.

[4] R. V. K. Pillai et al., "A Low Power Approach to Floating Point Adder Design," in *Proc. of the IEEE International Conference on Computer Design*, Oct. 1997, pp. 178–185.

[5] R. Zimmerman, "Efficient VLSI Implementation of Modulo $(2^n \pm 1)$ Addition and Multiplication," in *Proc. of 14th Symp. Computer Arithmetic*, April 1999, pp. 158–167.

[6] L. Kalampoukas et al., "High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders," *IEEE Trans. on Computers*, vol. 49, no. 7, pp. 673–680, Jul. 2000.

[7] C. Efstathiou et al., "Modulo $2^n \pm 1$ Adder Design Using Select Prefix Blocks," *IEEE Trans. on Computers*, vol. 52, no. 11, pp. 1399–1406, 2003.

[8] H. Ling, "High-Speed Binary Adder," *IBM Journal of R & D*, vol. 25, pp. 156–166, May 1981.

[9] G. Dimitrakopoulos and D. Nikolos, "High-Speed Parallel-Prefix VLSI Ling Adders," *IEEE Trans. on Computers*, , no. 2, pp. 22–22, Feb. 2005.

[10] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. on Computers*, vol. 31, no. 3, pp. 260–264, Mar. 1982.

[11] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. on Comp.*, pp. 786–792, Aug. 1973.

[12] R. E. Ladner and M. J. Fisher, "Parallel Prefix Computation," *JACM*, vol. 27, no. 4, pp. 831–838, Oct. 1980.

[13] G. Dimitrakopoulos et al., "A Family of Parallel-Prefix Modulo $2^n - 1$ Adders," in *Proc. of IEEE ASAP*, 2003, pp. 326–336.

[14] UMC-18, "eSi-Route/11 0.8 Standard Cell Library," *Virtual Silicon Technology*, Jan. 2001.