Area-Time Efficient Multi-Moduli Adder Design

H. T. Vergos
Department of Computer Engineering and
Informatics, University of Patras,
26500 Greece
Email:vergos@ceid.upatras.gr

D. Bakalis
Department of Physics,
University of Patras,
26500 Greece
Email:bakalis@physics.upatras.gr

***Abstract*—Multi-moduli architectures, that is, architectures that can deal with more than one modulo cases, are very useful for reconfigurable digital processors and fault-tolerant systems that are based on the Residue Number System (RNS). Two novel architectures are proposed for multi-moduli adders that support the most common moduli cases in RNS channels, that is, modulo $2^n-1, 2^n$ and 2^n+1. The proposed architectures use parallel prefix carry computation units composed of $\log_2 n$ levels. The experimental results show that the resulting adders are significantly faster and / or area efficient than the earlier proposals.**

***Index Terms*—Multi-moduli architectures, residue number system, modulo $2^n \pm 1$ arithmetic, parallel prefix carry computation**

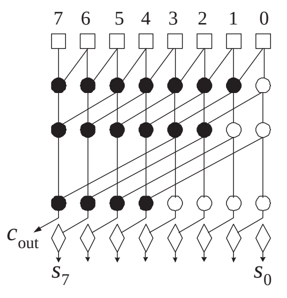
I. Introduction

The Residue Number System (RNS) [1], [2] is a number system commonly adopted for speeding up computations in digital signal processing [3], [4], [5], [6], cryptography [7] and telecommunication applications [8], [9] and for fault-tolerant computing [10], [11], [12]. A non-positional RNS is defined by a set of L moduli, suppose $\{m_1, \ldots, m_L\}$ that are pair-wise relatively prime. Let $|A|_M$ denote the modulo M residue of an integer A, that is, the least non-negative remainder of the division of A by M. A has a unique representation in the RNS, given by the set $\{a_1, \ldots, a_L\}$ of residues, where $a_i = |A|_{m_i}$. An operation $\diamond$ over an RNS is defined as $(z_1, \ldots, z_L) = (a_1, \ldots, a_L) \diamond (b_1, \ldots, b_L)$, where $z_i = |a_i \diamond b_i|_{m_i}$. The computation of z_i only depends on a_i, b_i, and m_i, implying that all z_is can be computed in parallel, each in a separate arithmetic unit often called a channel. Since all channels operate in parallel and each deals with narrow residues instead of wide numbers, significant speedup over the binary may be achieved.

RNS channels of the $2^n, 2^n-1$ or 2^n+1 moduli forms have received significant attention. This is because the required circuits for modulo 2^n can be straightforwardly derived from the corresponding integer circuits by limiting the result to n bits, while arithmetic circuits have been proposed for the $2^n \pm 1$ moduli that can operate as fast as the integer ones for the same operand widths. It is therefore no surprise that the three-moduli set $\{2^n-1, 2^n, 2^n+1\}$ is the most commonly used, given apart from the fast implementations of the channel circuits, fast converters between the residue and the binary representation. In this set of moduli, the 2^n+1 channel has to deal with operands one bit wider than the other two, leading to a performance bottleneck. To avoid this, and given that in the case of a zero operand the result can be derived straightforwardly, [13] introduced the diminished-1 representation. In the diminished-1 representation each number is represented decreased by one compared to its normal representation and all arithmetic operations are inhibited for a zero operand. The diminished-1 representation has the advantage that it requires only n bits, allowing to better equalize the delay of the three channels.

Reconfigurable computing for RNS-based systems has recently gained a significant interest. Multi-moduli architectures [14], that is, architectures for arithmetic circuits that support more than one modulo cases are very useful building blocks since they can be exploited for hardware reuse and offer substantial area savings. Multi-moduli architectures can be applied in reconfigurable RNS-based digital signal processors for providing flexibility and for easing the customization of the desired dynamic range and in fault-tolerant RNS-based systems for reducing the hardware costs of the redundant RNS channels. To this end, several reconfigurable multi-moduli architectures for the $\{2^n-1, 2^n, 2^n+1\}$ moduli set have been presented during the last few years. Reconfigurable multi-operand modulo $2^n \pm 1$ adders have been presented in [15]. Furthermore, [16] has presented combined multiplication/sum-of-squares modulo $2^n \pm 1$ units. Finally, [17], [18] have presented reconfigurable multi-moduli multipliers and squarers for the $\{2^n-1, 2^n, 2^n+1\})$ moduli set, respectively.

In this paper we introduce two novel architectures for the design of multi-modulo adders, that support the $2^n-1, 2^n$ and 2^n+1 modulo cases, assuming the diminished-1 representation for the latter. The multi-modulus adder is a critical subdesign in multi-moduli arithmetic components since it is used as the last stage of multipliers and squarers. The remaining of the paper is organized as follows. Section II presents some background issues on parallel prefix addition, on sparse parallel prefix carry computation units and on modulo $2^n \pm 1$ addition. It also briefly reviews the multi-modulus adder used in [17], [18], which forms the basis for our comparisons. The proposed multi-moduli adder architectures are then introduced in Section III and evaluated by VLSI implementations in Section IV. Our conclusions are presented in the last Section.



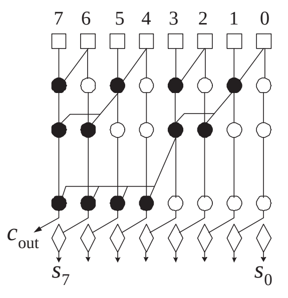


Fig. 1. Examples of 8-bit parallel prefix structures for integer adders.

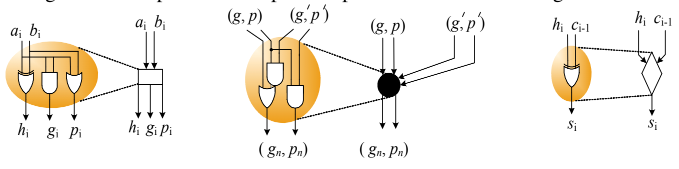
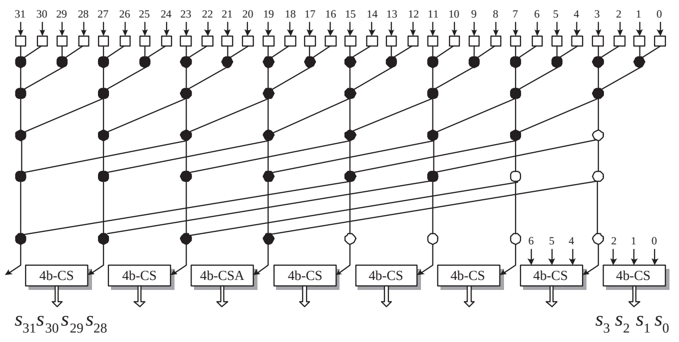


Fig. 2. Logic-level implementation of the basic cells.



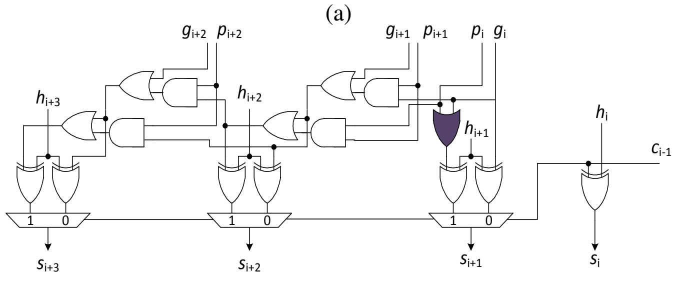


Fig. 3. (a) Sparse-4 parallel prefix structure for a 32-bit integer adder and (b) logic level implementation of the CSB.

II. Background

A. Parallel prefix carry computation

Every adder can be considered as a three-stage circuit. Assuming 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 addition operands, the preprocessing stage computes the carry-generate bits g_i and the half-sum bits h_i, for every i, $0 \leq i \leq n-1$, as $g_i = a_i \cdot b_i$ and $h_i = a_i \oplus b_i$. In an inclusive-OR adder, the preprocessing stage also computes the carry-propagate bits p_i, as $p_i = a_i + b_i$, where $\cdot$, $\oplus$ and $+$ in the above, denote logical AND, exclusive-OR and inclusive OR, respectively. In an exclusive-OR adder the half sum bits are also used as carry-propagate bits ($p_i = h_i$). This makes an exclusive-OR adder somewhat smaller and slower than an inclusive-OR one. 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. Finally, the third stage computes the sum bits according to $s_i = h_i \oplus c_{i-1}$.

Carry computation was transformed into a parallel prefix problem by the introduction of the $\circ$ operator [19], which associates pairs of generate and propagate signals according to $(g,p)\circ(g',p') = (g+p\cdot g', p\cdot p')$. In a series of associations of consecutive generate/propagate pairs (g,p) the notation $(g_{k:j}, p_{k:j})$, with $k > j$, is used for the group generate / propagate term produced out of bits $k, k-1, \ldots, j$, that is:

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

Since every carry c_i in an integer adder is equal to $g_{i:0}$, a number of distinct algorithms have been introduced for computing all the carries using only $\circ$ operators. These algorithms are most often represented by acyclic directed graphs in which the required prefix operators constitute the nodes. Figures 1(a) and (b) present the most well-known algorithms proposed by Kogge-Stone [20] and Ladner-Fischer [21], respectively, when used for the design of an 8-bit adder, while figure 2 depicts the logic-level implementation of the basic cells used in an inclusive-OR parallel prefix adder.

For large wordlengths, sparse parallel prefix adders are preferred, since they offer significantly reduced wiring and area without sacrificing delay. Their design relies on the use of a sparse parallel prefix carry computation unit and carry-select blocks (CSBs). Only the carries at the boundaries of the carry-select blocks are computed. A 32-bit adder with 4-bit sparseness is shown in figure 3(a). The carry select block computes a set of sum bits for each of the two possible values of the incoming carry. A logic-level implementation of a 4-bit carry-select block is shown in figure 3(b). The shaded OR is not required in inclusive-OR adders in which the input of the XOR gate can be driven by p_i. 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 block.

B. Modulo $2^n \pm 1$ addition basics

It is very well known that a modulo $2^n - 1$ adder can be implemented using an integer adder and by incrementing the sum when the carry output is 1, or equivalently, if we connect the carry output back to the carry input. Such a direct connection however may lead to oscillations and therefore to a poor design. [22] attacked this problem by using an extra level in the parallel prefix carry computation unit of an integer adder which is used to account for the carry output of the integer addition. On the other hand, [23] proposed to account for the carry output within the existing prefix levels by performing recirculation of the reentering carry. Figures 4(a) and (b) present these two alternatives assuming a Ladner-Fischer parallel prefix carry computation unit for [22]. The

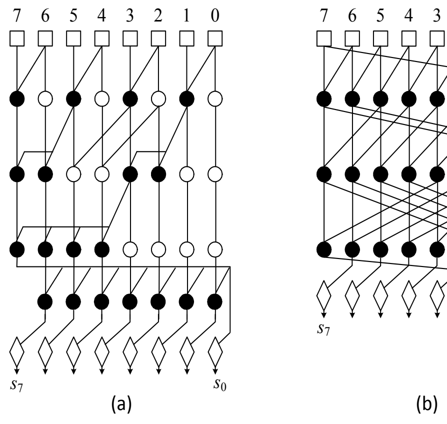


Fig. 4. The proposals of [22] (a) and [23] (b) for a modulo $2^8 - 1$ adder.

proposal of [23] has been shown to lead to superior adders in terms of delay and area than that of [22].

Similarly, a diminished-1 modulo $2^n + 1$ adder can be implemented using an integer adder and by incrementing the sum when the carry output is zero, or equivalently, if we connect the inverted carry output back to the carry input [22], [24]. Therefore, one needs to add just an inverter at the architecture of figure 4(a) to attain a diminished-1 adder. In [24] a new technique was proposed for recirculating the inverted carry output within the existing prefix levels of the carry computation unit and thus for computing the carries of the diminished-1 addition within $\log_2 n$ prefix levels.

Moreover, [23], [24] have shown that when written in parallel prefix form, the equations that define the carries of the modulo $2^n \pm 1$ carries computation, have a cyclic form and in contrast to integer addition, the number of generate and propagate pairs that have to be associated for each carry is equal to n. This means that a parallel prefix carry computation unit of a modulo $2^n \pm 1$ adder has significantly increased area complexity than that of a corresponding integer adder and therefore, a sparse design methodology may be preferred even for narrow operands. Sparse parallel prefix modulo $2^n - 1$ and modulo $2^n + 1$ adders have been presented in [25] and [26], respectively, in which recirculation of the carry output (normal and inverted respectively) is performed within the existing prefix levels and the same carry select block with that of the integer adders is used.

More interestingly though, [26] has shown that the carries of a diminished-1 adder, suppose c_i^+, are closely related to those of a modulo $2^n - 1$ adder suppose c_i^-, with $-1 \leq i < n - 1$. Specifically, it holds that $c_i^+ = c_i^- \oplus D_{i:0}$, where $D_{i:0} = h_i \cdot h_{i-1} \cdot \ldots h_0$ and $D_{-1:0} = 1$. Therefore, a diminished-1 adder can be designed following any architecture already proposed for a modulo $2^n - 1$ adder provided that the $D_{i:0}$ are computed in parallel and are used in the final stage of the adder for deriving the required diminished-1 carries.

C. Previous proposal

The only available multi-modulus adder in the open literature for the $\{2^n, 2^n - 1, 2^n + 1\}$ moduli set is the one used in [17], [18] for the design of multi-modulus multipliers and squarers, respectively, which is shown in figure 5 for 16-bit operands. It is based on the proposals of [22] and therefore uses $\log_2 n + 1$ prefix levels for computing the carries. A 3 to 1 multiplexer is further used to select the carry inserted at the last level. Zero is selected for modulo 2^n addition, the carry output for modulo $2^n - 1$ and the inverted carry output for diminished-1 modulo $2^n + 1$ addition. This architecture will be used as a basis for our comparisons and will be denoted by EAC-KS and EAC-LS in the following, when a Kogge-Stone or a Ladner-Fischer prefix structure is used, respectively.

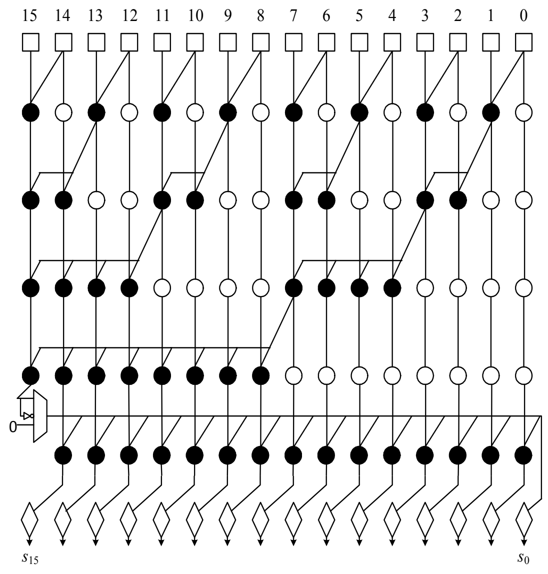


Fig. 5. Previous proposal for a multi-modulus 16-bit adder.

III. Novel Multi-Modulus Adder Architectures

Two novel multi-modulus adder architectures are introduced in this section. They are based on the proposals of [23] and [25] on the design of efficient modulo $2^n - 1$ adders, respectively. The main idea behind them is to modify a modulo $2^n - 1$ adder in order to also provide the carries of a modulo 2^n addition and to use the theory recently developed in [26] for conditionally attaining the diminished-1 carries from the modulo $2^n - 1$ ones before the last stage of the adder.

Both proposals use two function control signals, namely, n/m and dim. When n/m and dim are zero a modulo 2^n sum is attained, whereas for $n/m = 1, dim = 0$ and $n/m = 1, dim = 1$ a modulo $2^n - 1$ and a diminished-1 modulo $2^n + 1$ sum is produced. Irrespectively of the carry computation prefix structure used, a Ladner-Fischer parallel prefix structure is used in parallel for the $D_{i:0}$ signals computation in both architectures. Each node of the later is a simple two-input AND gate. However, since both proposals use the exclusive-OR carry propagate signal definition a significant amount of AND gates between the carry generation logic and the logic for the computation of the $D_{i:0}$ signals, is shared.

Finally, since the $D_{i:0}$ signals are expected to be derived earlier than the carry signals, in both architectures the

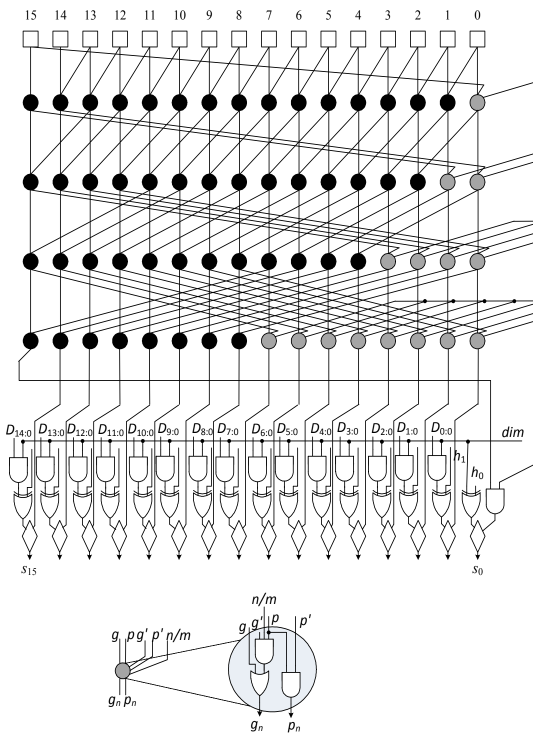


Fig. 6. Proposed FPP multi-modulus adder.

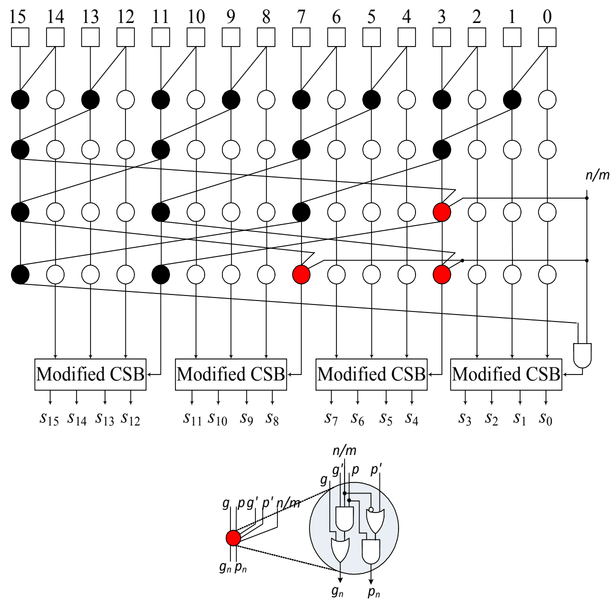


Fig. 7. Proposed SPP multi-modulus adder.

$D_{i:0}$ signals are directly associated with the half-sum (or equivalently with the carry propagate bits since we adopt an exclusive-OR definition) bits and the carry is then logically XOR-ed with the result. That is, instead of computing $s_i = h_i \oplus (c_{i-1} \oplus (dim \cdot D_{i-1:0}))$ both architectures compute $s_i = (h_i \oplus (dim \cdot D_{i-1:0})) \oplus c_{i-1}$, so that the late arriving carries are moved as close to the output of the adder as possible.

A. Proposed Parallel Prefix Multi-Modulus adders

Figure 6 presents the first proposed architecture for a 16-bit multi-modulus adder, hereafter called FPP architecture. It follows the parallel prefix structure proposed in [23]. All prefix operators that accept a feedback carry generate signal (indicated by the light grey shading in figure 6) are modified in order to take into account the n/m signal. If this is asserted, then the feedback carry is taken into account. Otherwise, it is ignored, leading to a modulo 2^n carry generation by the prefix node. The implementation of the modified prefix operator is also shown in figure 6. Compared to the normal prefix operator, this requires a 3-input AND gate instead of a 2-input one before the OR gate that produces the group carry signal. In custom VLSI technologies the group carry signals of both operators are implemented by AOI compound gates, indicating that the area and time overhead imposed by the modified operators, would be very small.

The following should be noted about the proposed FPP architecture : (a) since the group propagate signals of the modified operators are not needed in the following levels of the adder in the case of modulo 2^n addition, they do not need to be modified according to the value of the n/m signal and (b) the AND gate at the least significant bit position ensures a zero carry input for the modulo 2^n addition case.

B. Proposed Sparse Parallel Prefix Multi-Modulus adders

Figure 7 presents the second proposed architecture for a 16-bit multi-modulus adder, hereafter called SPP architecture. It follows the sparse parallel prefix structure proposed in [25]. The prefix operators that accept a feedback carry generate signal (indicated by the different shading in figure 7) are modified in order to take into account the n/m signal. In contrast to the FPP architecture both the group generate and group propagate outputs need to be modified, since the later will be used as inputs to the modified CSB at the final stage of the adder. The required implementation of the modified prefix operator is also shown in figure 7. Compared to the normal prefix operator, this requires the same modifications as those introduced above for the group generate term plus an extra OR gate for the group carry propagate term. Although the above modifications increase the area of the modified prefix operator they do not increase further the logic levels; the group propagate logic can be implemented in VLSI by an OAI compound gate.

As mentioned earlier, the $D_{i:0}$ signals are expected to be computed earlier than the carry signals. Therefore, a good design practice is to use them within the CSB logic before the carries. Figure 8 presents the logic level implementation of a modified CSB that achieves this. A 2-input AND gate controlled by the dim and the D signals followed by an XOR gate is added per sum bit for conditionally complementing the half sum bits. The delay that these modifications add does not reside on the critical path of the adder for sufficiently large operand lengths and is hidden in the delay of the carry computation logic.

TABLE I
EXPERIMENTAL RESULTS FOR DELAY OPTIMIZED ADDERS

	EAC-KS		EAC-LF		FPP		SPP	
n	Delay	Area	Delay	Area	Delay	Area	Delay	Area
4	277	732	275	703	186	856	248	572
8	359	1613	360	1207	268	2016	325	1183
16	436	3964	433	2862	336	4872	401	2619
32	519	9044	516	6554	410	11123	476	4679
64	603	20886	601	13495	501	23853	592	12264

TABLE II
EXPERIMENTAL RESULTS FOR AREA OPTIMIZED ADDERS

		EAC-KS		EAC-LF		FPP		SPP	
n	Target Delay	Delay	Area	Delay	Area	Delay	Area	Delay	Area
4	277	277	732	275	534	276	456	276	368
8	360	359	1613	360	1207	358	1242	354	815
16	436	436	3964	436	2620	436	3285	436	1829
32	519	519	9044	519	5830	519	8373	519	3244
64	603	603	20886	603	12791	603	20086	602	8358

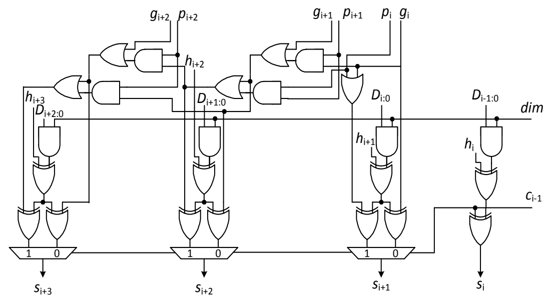


Fig. 8. Logic level implementation of the modified carry select block.

IV. COMPARISONS

In this section we compare the introduced FPP and SPP multi-moduli adder architectures against the EAC-KS and EAC-LF architectures used in [17], [18]. To this end, we described in HDL the multi-moduli adders derived by each architecture for $n = 4, 8, 16, 32$ and 64. Having validated the correct operation of the HDL descriptions by simulation, we synthesized each of them in a standard cell 90nm CMOS technology.

Each netlist was firstly optimized targeting the maximum operation frequency that can be achieved. The attained results are listed in Table I. Delay results are expressed in ps, while the area results in μm^2. The attained results indicate that EAC-KS and EAC-LF multi-moduli adders offer a similar execution speed throughout the examined adder range. Although the Kogge-Stone prefix structure offers a fan-out equal to 2 in the upper $\log_2 n$ prefix levels of the EAC-KS compared to $\frac{n}{2}$ offered by the Ladner-Fischer prefix structure in the EAC-LF case, these gains are limited or offset by the increase in the required prefix operators and their associated routing. Moreover, the multiplexer output that in both cases has a fan-out equal to n and therefore buffers need to be inserted at the larger wordlengths. Both proposed architectures offer significantly faster adder designs. The results of Table I indicate that the FPP adders offer delay savings that range from 16.6% up to 32.9%, while the SPP ones offer savings that range from 1.5% up to 10.5%. These savings can be explained by comparing figures 5, 6 and 7. It is obvious that both proposed architectures remove a parallel prefix level from the carry computation unit of the previous proposal and the required 3 to 1 multiplexer. Both these subdesigns reside on the critical path of the adder and therefore their removal leads to large delay savings at narrow wordlengths, that decrease as we move to wider ones. The FPP architecture leads to faster designs than the SPP because the later one also requires modifications of the group propagate signal. Stated otherwise, the FPP adders offer the delay of the currently fastest reported modulo $2^n - 1$ adders [23] increased by the time difference between an XOR gate and an OR gate and $\log_2 n$ times that between an AOI gate with a 3-input AND over that with a 2-input AND. It should be noted that the required buffering at the control signals n/m and dim in the proposed FPP and SPP adder does not reside on the critical path and therefore the maximum fan-out of the proposed FPP and SPP adders is limited to 2 and to the length of the CSB, respectively.

For comparing the area efficiency of the different architectures, the netlists were then optimized targeting the minimum implementation that can offer an operating frequency set by the worst delay of Table I in each wordlength case. The attained results are listed in Table II. Considering the implementation area, the EAC-LF adders are preferable over the EAC-KS ones. The proposed SPP adders offer the smallest implementations throughout the examined wordlength range. The savings offered against the previous proposals of EAC-LF and EAC-KS increase as we move to wider wordlengths up to 64.1%. This can be attributed to the removal of a large number of prefix operators apart from those of the reentering carry level, since only the carries at the CSB boundaries need to be computed. The removal of a prefix operator also implies the removal of

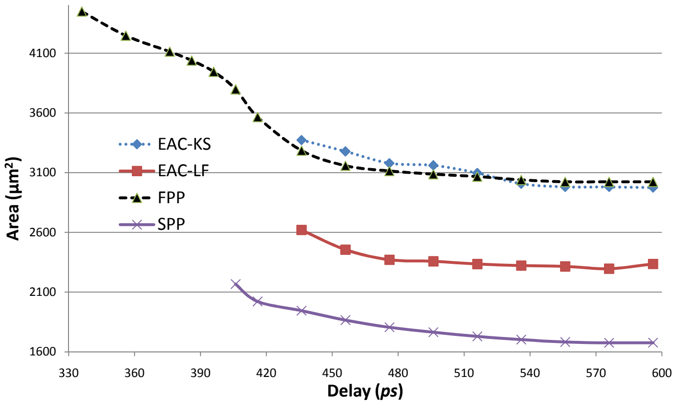


Fig. 9. Design space exploration for 16-bit multi-moduli adders.

at least 3 associated interconnections, leading to a significant reduction in the required wiring tracks. It should be noted that the addition of the Ladner-Fischer prefix structure of AND gates for computing the $D_{i:0}$ signals does not increase the implementation area a lot since its AND gates are shared by the prefix operators for computing the group propagate terms and is further compensated by the removal of n OR gates, since we adopt an exclusive-OR carry propagate definition. Although the proposed FPP adders offer the ultimate speed among the examined architectures, given that their structure relies on that of [23] which augments a Kogge-Stone structure by more operators to achieve reentering carry recirculation within the existing prefix levels, they are only more area efficient than the EAC-KS.

Figure 9 presents area-time curves for the examined multi-moduli adder architectures, for $n = 16$. If a delay smaller than $401ps$ is required then the proposed FPP architecture is the only alternative. In all the rest cases, the proposed SPP architecture should be preferred since it leads to the smallest implementations.

V. Conclusions

Reconfigurable computing in RNS-based systems has gained a significant interest during the last few years. Multi-moduli architectures result in very useful building blocks since they can be exploited for hardware reuse and area saving.

In this paper, two novel architectures for multi-moduli adders have been proposed. Both support the most commonly used moduli set, that is, the $\{2^n - 1, 2^n, 2^n + 1\}$ set. They are based on modifying the parallel-prefix carry computation unit of a modulo $2^n - 1$ adder so as to also produce the modulo 2^n carries. The diminished-1 modulo $2^n + 1$ carries can be derived by the modulo $2^n - 1$ ones using the theory recently developed in [26]. The proposed FPP adders are the fastest proposed and offer delay savings of up to 32.9% over the previous proposals used in [17], [18]. The proposed SPP adders offer less delay savings than the FPP ones, but cut down the implementation area by more than 30% in every examined case. New multi-moduli architectures for other arithmetic components are currently under investigation.

References

[1] P. V. A. Mohan, *Residue Number Systems : Algorithms and Architectures*. Springer-Verlag, 2002.
[2] A. Omondi and B. Premkumar, *Residue Number Systems : Theory and Implementations*. Imperial College Press, 2007.
[3] R. Chaves and L. Sousa, "RDSP: A RISC DSP based on Residue Number System," in 6^{th} *Euromicro Symp. on Digital System Design*, 2003, pp. 128–135.
[4] P. G. Fernandez and A. Lloris, "RNS-based Implementation of 8×8 point 2D-DCT over Field-Programmable Devices," *Electronics Letters*, vol. 39, no. 1, pp. 21–23, January 2003.
[5] Y. Liu and E. M.-K. Lai, "Moduli Set Selection and Cost Estimation for RNS-Based FIR Filter and Filter Bank Design," *Design Automation for Embedded Systems*, vol. 9, no. 2, pp. 123–139, June 2004.
[6] G. Cardarilli *et al.*, "Residue Number System for Low-Power DSP Applications," in *Asilomar Conf. on Signals, Systems and Computers*, 2007, pp. 1412–1416.
[7] J.-C. Bajard and L. Impert, "A Full RNS Implementation of RSA," *IEEE Trans. Comput.*, vol. 53, no. 6, pp. 769–774, June 2004.
[8] U. Meyer-Bäse, A. Garcia, and F. Taylor, "Implementation of a Communications Channelizer using FPGAs and RNS Arithmetic," *Journal of VLSI Signal Processing*, vol. 28, no. 1-2, pp. 115–128, May-June 2001.
[9] A. S. Madhukumar and F. Chin, "Enhanced Architecture for Residue Number System-based CDMA for High-Rate Data Transmission," *IEEE Trans. Wireless Commun.*, vol. 3, no. 5, pp. 1363–1368, May 2004.
[10] L. Impert *et al.*, "Fault-Tolerant Computations over Replicated Finite Rings," *IEEE Trans. Circuits Syst. I*, vol. 50, no. 7, pp. 858–864, 2003.
[11] I. Steiner *et al.*, "A Fault-Tolerant Modulus Replication Complex FIR Filter," in 16^{th} *Symp. on Application-Specific Systems, Architectures and Processors*, 2005, pp. 387–392.
[12] A. O'Donnel and C. Bleakley, "Area Efficient Fault Tolerant Convolution using PRNS with NTTs and WSCA," *Electronics Letters*, vol. 44, no. 10, pp. 648–649, 2008.
[13] L. M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 24, no. 5, pp. 356–359, October 1976.
[14] V. Paliouras and T. Stouraitis, "Multifunction Architectures for RNS Processors," *IEEE Trans. Circuits Syst. II*, vol. 46, no. 8, pp. 1041–1054, 1999.
[15] C.-H. Chang *et al.*, "A Configurable Dual Moduli Multi-Operand Modulo Adder," in *IEEE Int. Symp. Circuits and Systems*, 2005, pp. 1630–1633.
[16] D. Adamidis and H. T. Vergos, "RNS Multiplication / sum-of-squares," *IET Proceedings - Computers and Digital Techniques*, vol. 1, no. 1, pp. 38–48, January 2007.
[17] S. Menon and C. H. Chang, "A Reconfigurable Multi-Modulus Modulo Multiplier," in *IEEE Asia Pasific Conf. on Circuits and Systems*, December 2006, pp. 1168–1171.
[18] R. Muralidharan and C. H. Chang, "Fixed and Variable Multi-Modulus Squarer Architectures for Triple Moduli Base of RNS," in *Proc. of the IEEE Int. Symposium on Circuits and Systems*, May 2009, pp. 441–444.
[19] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. Comput.*, vol. 31, no. 3, pp. 260–264, 1982.
[20] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Trans. Comput.*, vol. 22, no. 8, pp. 786–792, August 1973.
[21] R. E. Ladner and M. J. Fischer, "Parallel Prefix Computation," *Journal of The ACM*, vol. 27, no. 4, pp. 831–838, 1980.
[22] R. Zimmerman, "Efficient VLSI Implementation of Modulo $(2^n \pm 1)$ Addition and Multiplication," in *Proc. of the* 14^{th} *IEEE Symposium on Computer Arithmetic*, April 1999, pp. 158–167.
[23] L. Kalampoukas *et al.*, "High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 673–680, 2000.
[24] H. T. Vergos, C. Efstathiou, and D. Nikolos, "Diminished-One Modulo $2^n + 1$ Adder Design," *IEEE Trans. Comput.*, vol. 51, no. 12, pp. 1389–1399, December 2002.
[25] G. Dimitrakopoulos *et al.*, "New Architectures for Modulo $2^n - 1$ Adders," in *Proc. of the* 12^{th} *IEEE International Conference on Electronics, Circuits & Systems*, December 2005.
[26] H. T. Vergos and G. Dimitrakopoulos, "On Modulo $2^n + 1$ Adder Design," *submitted to IEEE Trans. Comput.*, February 2010.