Configurable Booth-encoded Modulo 2^n±1 Multipliers

Evangelos Vassalos, Dimitris Bakalis
Electronics Laboratory, Department of Physics
University of Patras
Patras, Greece
vassalos@upatras.gr, bakalis@physics.upatras.gr

Haridimos T. Vergos
Department of Computer Engineering and Informatics
University of Patras
Patras, Greece
vergos@ceid.upatras.gr

Abstract—**Multi-moduli architectures are very useful for reconfigurable digital processors and fault-tolerant systems that utilize the Residue Number System (RNS). In this paper we propose a novel architecture for configurable modulo 2^n±1 multipliers. It uses the modified Booth encoding of the input operand for deriving the required partial products and an adder tree followed by a sparse parallel-prefix final adder for their addition. Experimental results show that the proposed multipliers offer significant savings in area and delay compared to those previously reported in the literature.**

Keywords-modulo 2^n±1 arithmetic; modulo multipliers; Booth-encoding; configurable circuits; residue number system

I. INTRODUCTION

The Residue Number System (RNS) is commonly adopted for speeding up computations in digital signal processing (DSP), cryptography and telecommunication applications and for fault-tolerant computing [1] [2]. 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. Then A is represented by the L-tuple $\{a_1, \ldots, a_L\}$, where $a_i = |A|_{m_i}$.

Moduli of the 2^n±1 form are most commonly used in RNS, given apart from fast implementations of the arithmetic operations for them, fast converters from/to residue to/from binary. The modulo 2^n+1 channel has to deal with operands that are one bit wider than those of the modulo 2^n-1 channel, forming a performance bottleneck. The diminished-one representation was introduced to face this problem [3]. In the diminished-one representation each modulo 2^n+1 operand is represented decreased by one compared to its normal representation. Zero values in input operands or results are indicated by special bits (zero indication bits) and are treated as special cases. The diminished-one representation requires only n-bit wide computation channels and is therefore more suitable for configurable modulo 2^n±1 architectures.

Configurable computing for RNS-based systems has recently gained a significant interest. Configurable modulo architectures [4], 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 offering substantial area savings. They can be applied in RNS-based digital signal processors for providing flexibility and for easing the customization of the desired dynamic range. A reconfigurable RNS datapath has been presented in [5]. For attaining low power consumption, some of its modulo channels can be turned on or switched off depending on the dynamic range required by an application, while each can be programmed with the required modulus value or the required arithmetic operation. Configurable architectures are also extremely useful in fault-tolerant RNS-based systems for reducing the hardware costs of the redundant RNS channels [6] [7]. A redundant RNS is formed by adding extra channels that are used for error detection and correction either by replication and comparison or by replacing the faulty ones. In case a modulus channel fails, then one of the extra ones can be used instead. The use of configurable channels instead of spare units for every distinct modulus value results in much lower area [8]. Finally, configurable architectures can be also utilized in cryptographic processors for executing multiple security protocols.

To this end, several configurable multi-moduli architectures have been presented during the last years. Configurable two-operand and multi-operand modulo adders have been presented in [8-10]. Furthermore, [11-14] have presented configurable multipliers and squarers for modulo 2^n±1 or for the moduli set $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$.

In this paper we propose a novel architecture for designing area-time efficient configurable modulo 2^n±1 multipliers. For the 2^n+1 case, the diminished-one representation is adopted while zero values are also taken into consideration. The proposed architecture uses radix-4 Booth encoding to reduce the number of generated partial products. An adder tree and a sparse-parallel prefix configurable final adder are utilized for partial product addition. Area and delay estimates, based on the unit-gate model as well as on specific VLSI implementations indicate that the proposed multipliers are significantly smaller than those previously reported in [12] while also being faster for medium and large values of n.

The remaining of the paper is organized as follows. Section II presents some basics on modulo 2^n±1 multiplication. The proposed architecture is introduced in Section III while Section IV presents experimental results and comparisons against a previous solution. Section V concludes the paper.

II. BACKGROUND

In the following, we assume an even value for n, which is the commonly examined case for Booth-encoded multipliers. However, at the end of the next Section we extend the proposed architecture in order to cover odd values of n as well. Hereafter, $\bar{x}$ is used to denote the complement of x.

A. Booth-Encoded Modulo 2^n-1 Multipliers

Suppose that $A=a_{n-1}a_{n-2}\ldots a_1a_0$ and $B=b_{n-1}b_{n-2}\ldots b_1b_0$ are two n-bit operands in modulo 2^n-1 representation, where A is the multiplicand and B is the multiplier. In order to half the number of the partial products of the multiplication and thus reduce the implementation area, the multiplier B can be encoded according to the modified Booth algorithm as $B=\left|\sum_{i=0}^{n/2-1}B_i2^{2i}\right|_{2^n-1}$, where $B_i=-2b_{2i+1}+b_{2i}+b_{2i-1}$ are the Booth-encoded digits and $b_{-1}=b_{n-1}$. The value of the product P of A and B taken modulo 2^n-1 can then be expressed as [15]:

$$P=\left|AB\right|_{2^n-1}=\left|\sum_{i=0}^{n/2-1}PP_i\right|_{2^n-1} \quad (1)$$

where $PP_i=\left|AB_i2^{2i}\right|_{2^n-1}$. Each n-bit wide partial product can be derived according to Table I, while the $n/2$ partial products must then be added modulo 2^n-1 in order to derive the result of the multiplication.

B. Booth-Encoded Diminished-one Modulo 2^n+1 Multipliers

Suppose now that A and B are two non-zero modulo 2^n+1 operands, that is $A,B\in(0,2^n]$. Let their diminished-one representations be denoted as $A_{-1}=A-1=a_{n-1}a_{n-2}\ldots a_1a_0$ and $B_{-1}=B-1=b_{n-1}b_{n-2}\ldots b_1b_0$, respectively. Among the various architectures that appear in the literature for Booth-encoded diminished-one modulo 2^n+1 multiplication, the architecture that is more suitable for the design of a configurable modulo $2^n\pm1$ multiplier is the one proposed in [16]. According to [16], the value of the diminished-one representation of the product $A\times B$ taken modulo 2^n+1 can be expressed as:

$$P_{-1}=\left|AB-1\right|_{2^n+1}=\left|\sum_{i=0}^{n/2-1}PP_i+\overline{CT}+C_0+n/2+1\right|_{2^n+1} \quad (2)$$

The $n/2$ n-bit wide partial products PP_i can be derived according to Table II, where $B_i=-2b_{2i+1}+b_{2i}+b_{2i-1}$ denotes the corresponding Booth-encoded digit of B_{-1} ($b_{-1}=\bar{b}_{n-1}$). CT is the n-bit wide vector $0z_{n/2-1}\cdots0z_10z_0$ with z_i denoting whether B_i has a zero value or not. C_0 is constant and equal to 0. The $n/2$ partial products PP_i along with $\overline{CT}$ and C_0 must then be added modulo 2^n+1 so as to form the diminished-one result of the multiplication. Note that although C_0 is equal to 0, it must be added modulo 2^n+1 with the rest partial products, for creating a constant correction term ($-n/2$) in order to compensate the term $n/2$ of (2) (more details are given in [16]).

TABLE I. MODULO 2^n-1 PARTIAL PRODUCT FORMATION

B_i	b_{2i+1}	b_{2i}	b_{2i-1}	PP_i						
				$PP_{i,[n-1]}$	…	$PP_{i,[2i+1]}$	$PP_{i,[2i]}$	$PP_{i,[2i-1]}$	…	$PP_{i,[0]}$
0	0	0	0	0 or 1	…	0 or 1	0 or 1	0 or 1	…	0 or 1
+1	0	0	1	a_{n-2i-1}	⋯	a_1	a_0	a_{n-1}	…	a_{n-2i}
+1	0	1	0	a_{n-2i-1}	⋯	a_1	a_0	a_{n-1}	…	a_{n-2i}
+2	0	1	1	a_{n-2i-2}	⋯	a_0	a_{n-1}	a_{n-2}	…	a_{n-2i-1}
-2	1	0	0	$\bar{a}_{n-2i-2}$	⋯	$\bar{a}_0$	$\bar{a}_{n-1}$	$\bar{a}_{n-2}$	…	$\bar{a}_{n-2i-1}$
-1	1	0	1	$\bar{a}_{n-2i-1}$	⋯	$\bar{a}_1$	$\bar{a}_0$	$\bar{a}_{n-1}$	…	$\bar{a}_{n-2i}$
-1	1	1	0	$\bar{a}_{n-2i-1}$	⋯	$\bar{a}_1$	$\bar{a}_0$	$\bar{a}_{n-1}$	…	$\bar{a}_{n-2i}$
0	1	1	1	0 or 1	…	0 or 1	0 or 1	0 or 1	…	0 or 1

TABLE II. MODULO 2^n+1 PARTIAL PRODUCT FORMATION

B_i	b_{2i+1}	b_{2i}	b_{2i-1}	PP_i						
				$PP_{i,[n-1]}$	…	$PP_{i,[2i+1]}$	$PP_{i,[2i]}$	$PP_{i,[2i-1]}$	…	$PP_{i,[0]}$
0	0	0	0	0	…	0	0	1	…	1
+1	0	0	1	a_{n-2i-1}	⋯	a_1	a_0	$\bar{a}_{n-1}$	…	$\bar{a}_{n-2i}$
+1	0	1	0	a_{n-2i-1}	⋯	a_1	a_0	$\bar{a}_{n-1}$	…	$\bar{a}_{n-2i}$
+2	0	1	1	a_{n-2i-2}	⋯	a_0	$\bar{a}_{n-1}$	$\bar{a}_{n-2}$	…	$\bar{a}_{n-2i-1}$
-2	1	0	0	$\bar{a}_{n-2i-2}$	⋯	$\bar{a}_0$	a_{n-1}	a_{n-2}	…	a_{n-2i-1}
-1	1	0	1	$\bar{a}_{n-2i-1}$	⋯	$\bar{a}_1$	$\bar{a}_0$	a_{n-1}	…	a_{n-2i}
-1	1	1	0	$\bar{a}_{n-2i-1}$	⋯	$\bar{a}_1$	$\bar{a}_0$	a_{n-1}	…	a_{n-2i}
0	1	1	1	0	…	0	0	1	…	1

III. PROPOSED CONFIGURABLE MODULO $2^n\pm1$ MULTIPLIERS

The proposed architecture for designing configurable modulo $2^n\pm1$ multipliers is based on the architectures described in the previous section and, depending on the value of a select signal, *sel*, can perform either modulo 2^n-1 (*sel*=0) or modulo 2^n+1 multiplication (*sel*=1). The resulting multipliers consist of three main units: (a) the partial product generation (*PPG*), (b) the partial product reduction (*PPR*), and (c) the final parallel addition (*FPA*) units. Each one is described in the following.

A. Partial Product Generation (PPG)

Both modulo 2^n-1 and 2^n+1 Booth-encoded multiplication architectures that have been described in the previous section are based on the generation of $n/2$ n-bit wide partial products. Fig. 1 presents logic implementations for the Booth Encoder (BE) (ignore the shaded logic gates for now) and the Booth Selector (BS) blocks that have to be used in order to derive the partial product bits. Suppose that we encode a zero partial product in the modulo 2^n-1 case only as 0…0 and not as 1…1. Then, by comparing Tables I and II we can see that for all PP_is, the partial product bits with weights 2^{2i+1} up to 2^{n-1} are the same in both modulo cases and thus modulo independent. On the other hand, the partial product bits with weights 2^0 up to 2^{2i-1} have complementary values in the two modulo cases. We can thus utilize 2-input XOR gates at the outputs of some BS blocks to selectively invert the values of the latter partial

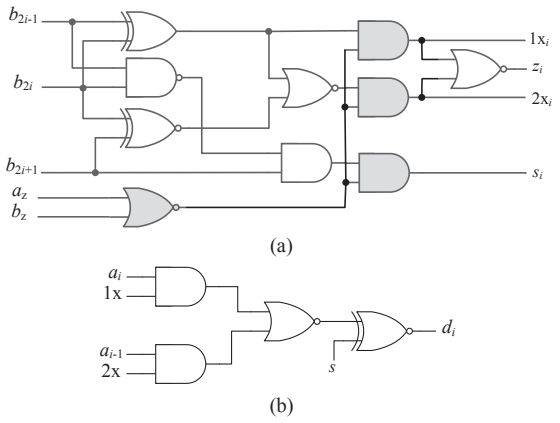

Figure 1. (a) Booth Encoder (BE) and (b) Booth Selector (BS) blocks

product bits depending on the value of *sel*. The partial product bit with weight 2^{2i} has to be inverted only when the corresponding Booth-encoded digit B_i has a value equal to ±2. Hence one 2-input XOR gate can also be used at the output of the corresponding BS block in each PP_i with one input driven by $sel \wedge 2x_i$, where $\wedge$ denotes the logical AND and $2x_i$ denotes the output of the BE block that designates that $B_i = \pm 2$.

Furthermore, the least significant digit, B_0, of the Booth-encoded operand has to use as b_{-1} either b_{n-1} or $\overline{b}_{n-1}$. Therefore, one more XOR gate with *sel* has to be utilized at the input of the least significant BE block to invert b_{n-1} when necessary.

Finally, since the $\overline{CT}$ vector should be equal to 0…0 in the modulo 2^n-1 case, we have to use the n-bit vector $sel(\bar{z}_{n/2-1} \wedge sel) \cdots sel(\bar{z}_1 \wedge sel) sel(\bar{z}_0 \wedge sel)$ in its place.

B. Partial Product Reduction (PPR)

In the proposed architecture, the $n/2+2$ vectors that have been generated (i.e. the $n/2$ PP_is with the extra logic gates, the $\overline{CT}$ and the C_0) are reduced in two n-bit summands by a Dadda adder tree of Carry Save Adders (CSAs). Each CSA consists of n full adders (FA), while the one that accepts C_0 can be simplified to n half adders (HAs). The carry output at the most significant bit position of a CSA has to be moved to the least significant bit position either inverted (modulo 2^n+1 case) or non-inverted (modulo 2^n-1 case) [12]. Hence, in the proposed architecture, each carry at the most significant bit position of a CSA is driven to a 2-input XOR gate with the second input driven by *sel*. The XOR gate drives the correct value to the least significant bit position.

C. Final Parallel Addition (FPA)

The two n-bit outputs of the adder tree must be finally added using a two-input configurable modulo 2^n±1 adder. Although a parallel prefix-based modulo adder with an extra prefix level for the end-around-carry, as the one described in [12], can be utilized, a more efficient solution in both area and delay terms can be achieved by simplifying the recently proposed configurable sparse parallel-prefix modulo $\{2^n\text{-}1, 2^n, 2^n\text{+}1\}$ adder [10]. We adopt the latter approach in the proposed architecture to save as much area and delay as possible.

D. Zero Handling in Modulo 2^n+1 Multiplication

So far we have assumed that the inputs of the proposed multiplier in the diminished-one modulo 2^n+1 case correspond to non-zero values. In the following we examine the case where either A or B or both are equal to zero. Let a_z and b_z denote the zero indication bits of inputs A and B, respectively, and let p_z denote the zero indication bit of the result of the multiplication. The product $A \times B$ taken modulo 2^n+1 is equal to 0 when either (a) A or B or both A and B are equal to 0, or (b) $A,B \neq 0$ and $|AB|_{2^n+1} = 0$ (eg. in the $|3 \times 3|_9$ multiplication). Case (a) can be taken into account by a 2-input OR gate driven by a_z and b_z. Case (b) can be easily covered according to [10]. By merging the above two cases, we conclude that the zero indication bit of the result can be given by the logical equation $p_z = a_z \vee b_z \vee D_{n-1:0}$, where $\vee$ denotes the logical OR operation and $D_{n-1:0}$ is a signal indicating that the final adder's input operands are bitwise complementary. This signal is provided without any delay cost by the sparse parallel-prefix adder of [10]. Furthermore, in order to force the n-bit output of the final adder to the value of 0 when A=0 or B=0, we can force the output of each BE block to zero [16]. This can be achieved by augmenting each BE block by the shaded logic gates shown in Fig. 1(a), which take into consideration the values of a_z and b_z.

As an example, Fig. 2 presents the configurable modulo 2^8±1 multiplier, according to the proposed architecture.

For odd values of n, the *PPG* unit produces $(n+1)/2+2$ partial products instead of $n/2+2$. The proposed architecture can be easily extended to consider those values of n as well. The only modifications required are: (i) signal b_{-1} that is used by the least significant BE must be driven directly by *sel*, since it is equal to either 0 (2^n-1) [15] or 1 (2^n+1) [16], (ii) the input b_{2i+1} of the most significant BE must be connected to 0 for both moduli cases and (iii) the n-bit vector $\overline{CT}$ must be replaced by $(\bar{z}_{n-1/2} \wedge sel) sel \cdots (\bar{z}_1 \wedge sel) sel (\bar{z}_0 \wedge sel)$.

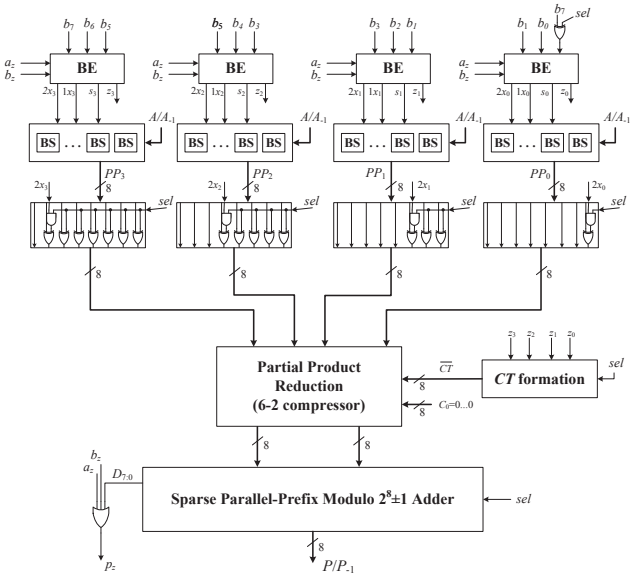

Figure 2. Proposed configurable modulo 2^8±1 multiplier

IV. Evaluation and Comparison

According to the proposed architecture, a configurable modulo $2^n\pm1$ multiplier consists of: (a) $\lfloor (n+1)/2 \rfloor$ BE blocks, (b) $\lfloor (n+1)/2 \rfloor \times n$ BS blocks, (c) $\lfloor (n+1)/2 \rfloor$ CSAs, each one composed of n FAs or HAs and one 2-input XOR gate, and (d) a sparse parallel-prefix configurable modulo $2^n\pm1$ adder [10]. It also uses: (a) $\lfloor (n+1)/2 \rfloor^2$ XOR gates and $\lfloor (n+1)/2 \rfloor$ AND gates for deriving the modulo dependent partial product bits, (b) $\lfloor (n+1)/2 \rfloor$ AND gates for forming the $\overline{CT}$ vector, (c) one 3-input OR gate for the zero indication bit of the result, and (d) one XOR gate at the input of the least significant BE block (required only for even values of n).

We compare the proposed multipliers against those presented in [12] which are the only available in the literature. Table III presents the area and delay requirements of both solutions for even values of n, in terms of gate equivalents according to the unit-gate model [17]. Similar results can be easily derived for odd values of n but are omitted due to lack of space. $\vartheta(k)$ denotes the minimum number of levels of an adder tree that processes k input operands. We observe that the proposed multipliers are more area efficient than those of [12] while savings in delay may also be achieved for medium and large values of n.

We also described in HDL configurable multipliers for several values of n (8, 16, 32 and 64). After validating the correct operation of the HDL descriptions via simulation, we synthesized them in a standard cell 90nm CMOS technology, using a standard delay optimization script, and derived estimates for area and delay. The attained results for both the proposed multipliers and those of [12] are given in Table IV. The proposed multipliers are significantly smaller than those of [12] in every examined case. They are also faster for values of n greater than 8. The offered savings in area are up to 26.7% while those in delay are up to 23.7% in the examined wordlengths. The savings are attributed to the Booth encoding of the B input operand which reduces the area and the delay of the *PPR* unit and to the use of the sparse parallel-prefix final adder.

TABLE III. Unit-Gate Area and Delay Estimations

Unit	Area		Delay	
	[12] (gate eqs.)	Proposed (gate eqs.)	[12] (gate eqs.)	Proposed (gate eqs.)
PPG	$2n^2+3n+3$	$3n^2+7n+2$	5	12
PPR	$7n^2-9n+6$	$(7/2)n^2-3n$	$5\vartheta(n+1)$	$5\vartheta(n/2+2)$
FPA	$(3/2)n\log n+7n+3$	$(11/8)n\log n+(123/8)n+2$	$2\log n+8$	$2\log n+3$
Total	$9n^2+(3/2)n\log n+n+12$	$(13/2)n^2+(11/8)n\log n+(155/8)n+4$	$2\log n+13+5\vartheta(n+1)$	$2\log n+15+5\vartheta(n/2+2)$

TABLE IV. CMOS VLSI Experimental Results

n	Area			Delay		
	[12] (μm^2)	Proposed (μm^2)	Savings (%)	[12] (ns)	Proposed (ns)	Savings (%)
8	5389	4415	18.1	1.61	1.60	0.6
16	17958	14480	19.4	2.40	2.07	13.6
32	69389	51886	25.2	3.21	2.45	23.7
64	259007	189872	26.7	4.11	3.30	19.6

V. Conclusion

An architecture for designing configurable modulo $2^n\pm1$ multipliers has been presented. For the modulo 2^n+1 case, the diminished-one representation is considered. The number of the partial products that are generated is reduced by using radix-4 Booth encoding of the multiplier, while their summation is performed by a Dadda adder tree and a sparse parallel-prefix configurable modulo adder. Experimental data verified that the proposed architecture results in more area-time efficient circuits than the previously reported [12].

Acknowledgment

This research was supported by the Caratheodory Programme of the University of Patras (D.178).

References

[1] P. V. Ananda Mohan, Residue Number Systems: Algorithms and Architectures, Netherlands: Kluwer Academic Publishers, 2002.

[2] A. Omondi and B. Premkumar, Residue Number Systems: Theory and Implementation, London: Imperial College Press, 2007.

[3] L. Leibowitz, "A simplified binary arithmetic for the fermat number transform," IEEE Trans. Acoust., Speech, Signal Processing, vol. 24, no. 5, pp. 356-359, 1976.

[4] V. Paliouras and T. Stouraitis, "Multifunction architectures for RNS processors," IEEE Trans. on Circuits and Systems - II, vol. 46, no. 8, pp. 1041-1054, 1999.

[5] G. C. Cardarilli, A. Re, A. Nannarelli and M. Re, "Residue number system reconfigurable datapath," IEEE Int. Symp. Circuits and Systems, pp. II-756-759, 2002.

[6] W. Jenkins, B. Schnaufer and A. Mansen, "Combined system-level redundancy and modular arithmetic for fault tolerant digital signal processing," IEEE Int. Symp. Computer Arithmetic, pp. 28-35, 1993.

[7] A. P. Preethy, D. Radhakrishnan and A. Omondi, "Fault-tolerance scheme for an RNS MAC: Performance and cost analysis," IEEE Int. Symp. Computer Arithmetic, pp. 717-720, 2001.

[8] G. Jaberipur and B. Parhami, "Unified approach to the design of modulo-($2^n\pm1$) adders based on signed-LSB representation of residues," IEEE Int. Symp. Computer Arithmetic, pp. 57-64, 2009.

[9] C. –H. Chang, S. Menon, B. Cao and T. Srikanthan, "A configurable dual-moduli multi-operand modulo adder," IEEE Int. Symp. Circuits and Systems, pp. 1630-1633, 2005.

[10] H. T. Vergos and D. Bakalis, "Area-time efficient multi-modulus adders and their applications," Elsevier Microprocessors and Microsystems, 2012, doi: 10.1016/j.micpro.2012.02.004, in press.

[11] S. Menon and C. –H. Chang, "A reconfigurable multi-modulus modulo multiplier," IEEE Asia Pacific Conf. Circuits and Systems, pp. 1168-1171, 2006.

[12] T.-B. Juang, G.-L. Wu and Y.-C. Tsai, "Reconfigurable modulo $2^n\pm1$ multipliers," 21st VLSI Design/CAD Symp., pp. 139-142, 2010.

[13] R. Muralidharan and C. –H. Chang, "Fixed and variable multi-modulus squarer architectures for triple moduli base of RNS," IEEE Int. Symp. Circuits and Systems, pp. 441-444, 2009.

[14] D. Bakalis and H. T. Vergos, "Area-Efficient Multi-Moduli Squarers for RNS," Conf. Digital System Design, pp. 408-411, 2010.

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

[16] J. W. Chen and R. H. Yao, "Efficient modulo 2^n+1 multipliers for diminished-1 representation," IET Circuits, Devices & Systems, vol. 4, no. 4, pp. 291-300, 2010.

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