A 200-MHz RNS Core

H. T. Vergos*

Abstract – **The need for fast computation of digital signal processing algorithms and the development of VLSI techniques of fabrication have motivated the development of efficient hardware implementations of Residue Number System (RNS) arithmetic. In this paper the architecture and implementation details of a core capable to perform both addition and multiplication operations over the moduli set $<2^{32}, 2^{32}-1$ and $2^{32}+1>$ is presented. The core accepts its input operands in either residue or straight binary forms. Heavy pipelining of the multiplication modules is used to achieve a 200 MHz operating frequency in a 0.6 um implementation technology.**

1 Introduction

The use of pre-designed (either in-house or provided as third-party Intellectual Property blocks / cores) blocks, is the only means for handling both the ever shrinking time to market and the increased design effort complexity imposed by the millions of transistors in contemporary integrated circuits. In this paper, we present a core capable of performing addition and multiplication over the Residue Number System (RNS) defined by the moduli $<2^{32}, 2^{32}-1, 2^{32}+1>$.

Since the 1950s there has been an increasing interest in Residue Number System (RNS) arithmetic as a basis for computational hardware [1 - 3]. Because of the complexity of division and comparison, RNS is judged unsuitable for general purpose computers. However, the growing importance of digital signal processing within electrical engineering as well as the newly introduced design techniques for RNS hardware have renewed the interest in RNS arithmetic. RNS is especially suitable for the implementation of DSP related algorithms due to its capabilities of rapid computations in the simple operations of addition, substraction and multiplication.

A set of L moduli suppose $(m_1, m_2, \ldots, m_L)$ that are pair-wise relative prime defines a RNS. Any integer X, with $0 \le X < M$, where $M = m_1 * m_2 * \ldots * m_L$ has a unique representation in the RNS system given by the L-tuple of residues $X= (x_1, x_2, \ldots, x_L)$, where $x_i = X \bmod m_i$. A two operand RNS operation, suppose ♦, is defined as $(Z_1, Z_2, \ldots, Z_L) = (X_1, X_2, \ldots, X_L) \diamond (Y_1, Y_2, \ldots, Y_L)$, where $Z_i = (X_i \diamond Y_i) \bmod m_i$. For most RNS applications ♦ is either addition or multiplication. According to the above, each residue can be computed independently of the others allowing fast data processing in L parallel independent channels.

The latency of an RNS operation depends on the latency of the slowest among the channels. Therefore, for nearly equal delay among the channels, their moduli should be close as possible. Moreover, the choice of the moduli dictates the maximum number M, of distinct numbers that can be represented. Finally, one has to consider the complexity and cost of the transformations between binary to RNS encoding process, since the binary system has still to be used because of its memory storage efficiency. A choice often made is L = 3 with moduli of the form $m_1 = 2^n$, $m_2 = 2^n-1$ and $m_3 = 2^n+1$, because of the existence of efficient combinational designs of residue generators, adders and multipliers for such moduli. In this paper, we further consider n = 32, since this gives us a representation range similar to that of a 96-bit precision binary system. This precision is adequate for most digital signal processing algorithms. For meeting the equal channel delay goal, we further adopt the diminished number representation [4] for the modulo $2^{32}+1$ channel. In the diminished-one system each number X is represented by $X^* = X - 1$. The representation of 0 is treated in a special way. Therefore, the circuits that implement the diminished–one modulo 2^n+1 operations are combinational circuits accepting n bits wide operands.

In a RNS a datum that is stored in binary may have to be converted into RNS and then used in several operations before it needs to be converted back to binary and stored. It was also chosen to include in the presented core a binary to RNS converter (a circuit that in other design philosophies could be thought as a wrapper of a RNS core) as well as to augment the instruction set in order to include convert and process operations. A similar input and output interface is used for the core, such that more than one cores can be cascaded in order to handle different RNS computations.

The rest of the paper is organized as follows : the interface, architecture and instruction set of the core is presented in the next section. The specific implementation of the modules used are presented in Section 3, along with area and delay results. The conclusions are drawn in the last Section.

* Department of Computer Engineering & Informatics, University of Patras, 26 500 Greece &
Computer Technology Institute, 3 Kolokotroni Str, 262 21 Patras, Greece.
E-mail : vergos@cti.gr
Tel : +30 – 61 – 960 312, Fax : +30 – 61 – 991 909.

2 RNS Core Architecture

Figure 1 indicates the RNS core interface.

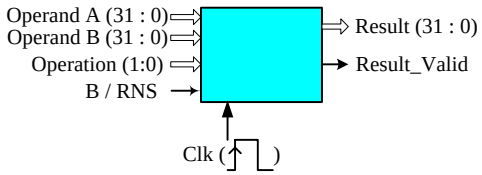

Figure 1. Input / Output Interface of the RNS core.

On the inputs side there is a 32 bit bus for each operand. Each operand is introduced to the core during three consecutive cycles. In case of RNS operands, the modulo 2^{32}, the modulo 2^{32}-1 and the modulo 2^{32}+1 of the operand should appear at the busses of operands A and B during respectively the first, the second and the third consecutive cycle. The buses are sampled with the rising edge of the clock. In case of 96-bit binary operands the operands should appear from the least significant 32-bit quantity to the most significant one during the three consecutive clock cycles.

The desired operation of the core is designated by the values present during the first cycle of a new operation at the Operation bus and the B(inary)/RNS line. Table 1 lists the possible operations. Note that when 00 is the desired operation and B/RNS is at 0 only the Operand A bus is sampled for input.

Op	B/RNS	Description
00	0	Binary to RNS conversion
00	1	*Invalid*
01	0	Addition of binary operands
01	1	Addition of RNS operands
10	0	Multiplication of binary operands
10	1	Multiplication of RNS operands
11	X	*Reserved for Testing purposes*

Table 1: RNS core operations.

The result of the RNS core is present some cycles later on the Result Bus in a way similar to that of the input. However, the result is always in a RNS form. Therefore during the first cycle the modulo 2^{32} of the result appears, while in the two subsequent cycles the modulo 2^{32}-1 and the modulo 2^{32}+1 in diminished-one form of the result appear on the same bus. To handle correct sampling of the results by subsequent modules the Result_Valid output is asserted during the three output cycles. It is obvious that several RNS cores can be cascaded providing a pipelined at the instruction level RNS application.

Figure 2 presents a more detailed look at the architecture of the core and depicts the main design blocks. The two residue generators are used only when the operands are in binary format and respectively produce the modulo 2^{32}-1 and the modulo 2^{32}+1 of the input operands. Note that this outline description hides a lot of the actual resource sharing that is performed on chip. For example each residue 2^{32}-1 generator can be built using two modulo 2^{32}-1 adders. Moreover, two more such adders are needed in the modulo 2^{32}-1 channel; one for the addition and one as the last stage of the multiplier. Therefore, one may suppose that six copies of a modulo 2^{32}-1 adder are required in the core. However, efficient resource sharing and the way that the input operands arrive permits to perform these operations using only three modulo 2^{32}-1 adders, without any conflicts.

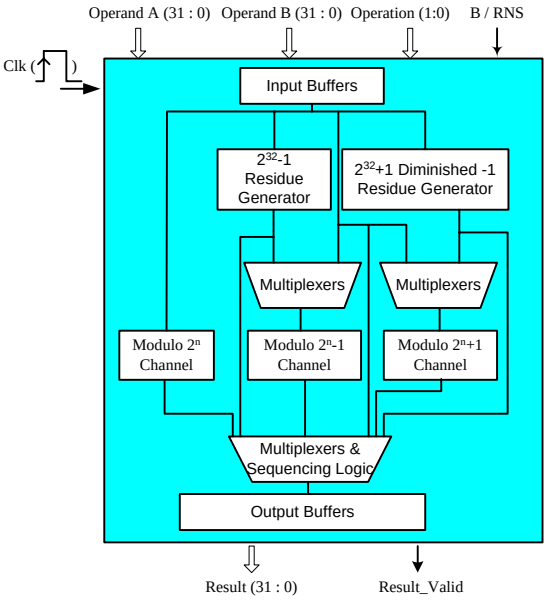

Figure 2. Architecture of the RNS Core.

If the operation was just a conversion from binary to RNS the outputs of the residue generators are sent to the outputs. In all other cases the residue representation of the operands (that is, either the outcome of the generators or the input buffers contents in case of operands already in RNS representation) are sent to the three channels. Note that the three channels perform their operation in parallel and independently and that the modulo 2^n channel starts its operation earlier than the rest two (since both its operands are input earlier and no residue generation is required). Both generators require the same (two) clock cycles to produce their results and given the ordering in the arrival of the inputs the modulo 2^n-1 result is also available always a cycle earlier than those of the modulo 2^n+1 channel and at most two cycles later than the result of the modulo 2^n channel.

Table 2 shows the operations performed in each cycle and channel for converting and multiplying two numbers that are introduced to the core in 96-bit binary representation. This is the operation that

requires the most cycles for its completion.

Cycle #	Operations Performed	
1	2^{32}:	Operand Sampling
	2^{32}-1:	-
	2^{32}+1:	-
2	2^{32}:	Multiplication Cycle 1
	2^{32}-1:	Residue Generation Cycle 1
	2^{32}+1:	-
3	2^{32}:	Multiplication Cycle 2
	2^{32}-1:	Residue Generation Cycle 2
	2^{32}+1:	Residue Generation Cycle 1
4	2^{32}:	Multiplication Cycle 3
	2^{32}-1:	Multiplication Cycle 1
	2^{32}+1:	Residue Generation Cycle 2
5	2^{32}:	Multiplication Cycle 4
	2^{32}-1:	Multiplication Cycle 2
	2^{32}+1:	Multiplication Cycle 1
6	2^{32}:	Multiplication Cycle 5
	2^{32}-1:	Multiplication Cycle 3
	2^{32}+1:	Multiplication Cycle 2
7	2^{32}:	Multiplication Cycle 6
	2^{32}-1:	Multiplication Cycle 4
	2^{32}+1:	Multiplication Cycle 3
8	2^{32}:	-
	2^{32}-1:	Multiplication Cycle 5
	2^{32}+1:	Multiplication Cycle 4
9	2^{32}:	Output Result
	2^{32}-1:	Multiplication Cycle 6
	2^{32}+1:	Multiplication Cycle 5
10	2^{32}:	-
	2^{32}-1:	Output Result
	2^{32}+1:	Multiplication Cycle 6
11	2^{32}:	-
	2^{32}-1:	-
	2^{32}+1:	Output Result

Table 2. The operations performed in each cycle for a binary multiplication.

A new operation can either begin after the fourth cycle of the previous operation in the case that the later involved residue generation or at every new cycle when the inputs of the last operation were in residue format, if an extended input interface is used.

3 Implementation Analysis

In this section we focus on the architectures chosen for the design of the individual blocks of Figure 2. The presented implementation results were gathered by describing each design module in HDL and mapping to the AMS CUB implementation technology (0.6 μm, 2-metal layer, 5.0 V) using the Design Analyzer® tool of Synopsys Inc. Each mapped design was recursively optimized until the tool was unable to provide a faster design. Then the tool was instructed to recover as much area as possible. All delay results assume worst case process parameters and are expressed in ns, whereas all area results are expressed in mils^2. Table 3 summarizes the implementation results and also lists the cycles of a 200 MHz clock required by each module.

Design Module	Area	Delay	Cycles
2^{32}-1 Residue Generator	1532.4	4.97	2
2^{32}+1 Residue Generator	1983.2	7.65	2
2^{32} Adder	876.5	4.29	1
2^{32}-1 Adder	1341.2	4.97	1
2^{32}+1 Adder	1356.4	4.79	1
2^{32} Multiplier	5284.1	17.01	6
2^{32}-1 Multiplier	11156.2	29.18	6
2^{32}+1 Multiplier	12090.5	29.72	6
Whole Core	46483.7		

Table 3. Summary of the implementation results

The design of each 2^{32}-1 residue generator is given in Figure 3 and is composed of a single modulo 2^{32}-1 adder. Supposing that the three 32 bit slices of the 96-bit operand X are symbolized as MS (most significant), MES (medium significant) and LS (least significant) the residue generation is performed in two cycles : a) during the first cycle LS is added with MES and b) during the second cycle the result is added with MS. Several architectures have been proposed for the design of the modulo 2^{32}-1 adder [5 – 7]. Although the architecture of [7] requires the largest implementation area, since we are mainly interested in the attained execution frequency, we have adopted it in this paper.

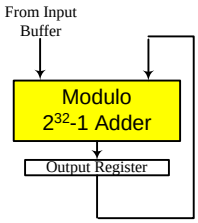

Figure 3. Design of the 2^{32}-1 Residue Generator

The design of each 2^{32}+1 residue generator is based on the generators proposed in [8]. We have modified the last stage of these generators in order to produce the diminished-one residue representation, that is, to also subtract 1 from the original result. Although, these generators can be pipelined at the full-adder level, in the proposed design this was not necessary, since only two pipeline stages were enough to provide the same clock frequency with that of the 2^{32}-1 residue generator.

For the modulo 2^{32}-1 multiplier it is obvious that the ROM-based solutions that have been proposed in the past based on look-up tables [9] are clearly inapplicable. Combinational modulo multipliers that are based on Carry Save Adder (CSA) trees have been proposed in [10]. However in this specific case, such an architecture would lead to the need for the addition of 32 partial products; that is, a time inefficient design. Attempts to reduce the number of partial products by applying the Booth algorithm have appeared in [6, 11]. It has been recently shown [12] that for even values of n, modified Booth modulo 2^n-1 multipliers with the minimal number of n/2 partial products can be

devised. Moreover, Wallace trees partial products reduction have been shown [12] to be more efficient than CSA arrays. In the proposed RNS core we adopt the architecture of [12] along with Wallace trees partial product reduction.

Even with this minimized number of partial products the latency of the multiplication operation is still too long. To spread and "hide" the latency of the multiplication over more cycles a deep pipelining scheme is used. The modulo 2^{32}-1 multiplier has 6 pipeline stages. The first stage accepts 4 partial products and reduces them to two vectors of sum and carry bits. Each of the following 4 stages accepts the later two vectors and three more partial products and performs a similar reduction. The last stage of the multiplier is composed of the modulo 2^{32}-1 parallel adder. The implementation results, indicated that the pipelined version of the modulo 2^{32}-1 multiplier requires approximately 32% more implementation area than the non-pipelined one.

Several architectures have also been proposed for the modulo 2^{32}+1 multiplier, for example, [6, 13]. For the proposed RNS core the modified Booth architecture proposed in [6] was adopted, along with Wallace trees partial products reduction. The multiplier was also in this case heavily pipelined using 6 stages in a similar to the modulo 2^{32}-1 way. The last stage parallel diminished-one modulo 2^{32}+1 adder follows the architecture recently proposed in [14].

The implementation of the core in the target 0.6um technology achieves an operating frequency of 200 MHz. Operations of operands in residue format can be done at this frequency, provided that the input interface is extended. Operands in binary format attain an execution frequency of 50 MHz since a new operation can start every fourth cycle if the previous operation also required a binary to RNS conversion. The presented core offers high execution rate since in the same implementation technology a 96-bit parallel-prefix binary adder has an operating frequency of 34.7 MHz, while a 6-stage pipelined 96-bit operand binary modified Booth multiplier achieves an operating frequency of 19.4 MHz.

4 Conclusions & Future Work

The use of efficient cores becomes a necessity in contemporary IC design in order to meet strict time to market requirements and to manage increasing design complexity. In this paper we have presented the architecture and implementation details of a core for RNS systems. Due to its capabilities of rapid addition, and multiplication RNS is attractive for the implementation of DSP related algorithms.

The presented core is capable of performing addition and multiplication in operands already in residue representation as well as to of 96-bit wide binary operands. Several cores can also be connected in a cascaded manner to handle larger computations. The implementation of the presented core in a 0.6 um technology can achieve an execution frequency of 200 MHz on RNS operands, and 50 MHz on operands that need first to be converted from binary to RNS.

We are currently developing higher level generators that will produce parameterized HDL descriptions of the design modules required for any size RNS core in order to transform the presented core in a totally soft one.

References

[1] M. A. Sonderstrand et. al., Residue Number System Arithmetic : Modern Applications in Digital Signal Processing, IEEE Press, New York, 1986.

[2] Koren, Computer Arithmetic Algorithms, Prentice–Hall, 1993.

[3] K. M. Elleithy & M. A. Bayoumi, "Fast and Flexible Architectures for RNS arithmetic decoding", IEEE Trans. Circuits and Systems–II, vol. CAS-39, pp. 226 – 235, April 1992.

[4] L. M. Leibowitz, "A simplified binary arithmetic for the Fermat number transform", IEEE Trans. Acoustics, Speech, Signal Processing, Vol. ASSP-24, pp. 356-359, 1976.

[5] C. Efstathiou, et. al., "Area–Time Efficient Modulo 2^n-1 Adder Design", IEEE Trans Circuits and Systems–II, Vol. 41, No 7, pp. 463–467, July 1994.

[6] R. Zimmermann, "Efficient VLSI Implementation of Modulo ($2^n \pm 1$) Addition and Multiplication", in Proc. of 14th IEEE Symp. on Comp. Arithmetic, pp. 158–167, April 1999.

[7] L. Kalampoukas, et. al., "High-Speed Parallel-Prefix Modulo 2^n-1 Adders", IEEE Trans Computers, Vol. 49, No 7, pp. 673-680, July 2000.

[8] S. Piestrak, "Design of Residue Generators and Multioperand Modular Adders Using Carry – Save Adders", IEEE Trans Computers, Vol. 423, No 1, January 1994.

[9] A. Skavantzos and P. B. Rao, "New multipliers modulo 2^n-1", IEEE Trans. on Computers, Vol. 41, No. 8, pp. 957-961, August 1992.

[10] Z. Wang, G. A. Jullien, W. C. Miller, "An algorithm for multiplication modulo 2^N-1", Proc. of the 39th Midwest Sym. on Circuits and Systems, pp. 1301-4, vol. 3, 1997.

[11] C. Efstathiou & H. T. Vergos, " Modified Booth 1's Complement and Modulo 2^n-1 Multipliers",7th IEEE International Conference on Electronics, Circuits & Systems, (ICECS '2K), Volume II, pp. 637-640.

[12] C. Efstathiou, et al., "On Modified Booth Modulo 2^n-1 and 2^n Multipliers", Journal of VLSI Signal Processing Systems, *under review*.

[13] Y. Ma, "A Simplified Architecture for Modulo (2^n+1) Multiplication", IEEE Trans. Computers, Vol. 47, No. 3, pp. 333 – 337, March 1998.

[14] H. T. Vergos, et al., "High Speed Parallel-Prefix Modulo 2^n+1 Adders for Diminished-One Operands", *to be presented at the 15th IEEE Symposium on Computer Arithmetic, Vail Colorado, 11-13 June 2001.*