OPTIMIZATION OF RNS FIR FILTERS FOR 6-INPUTS LUT BASED FPGAS

G.C. Cardarilli, M. Re, A. Salsano

University of Rome Tor Vergata
Department of Electronic Engineering
Via del Politecnico 1 / 00133 / Rome / ITALY
{marco.re, g.cardarilli}@ieee.org
salsano@ing.uniroma2.it

S. Pontarelli

(ASI) Italian Space Agency
Viale Liegi 26
00198 Rome, ITALY
pontarelli@ing.uniroma2.it

ABSTRACT

In this paper optimized Residue Number System (RNS) arithmetic blocks to better exploit some of the architectural characteristics of the last generation FPGAs are presented. The implementation of modulo m adders, constant and general multipliers, input and output converters is presented. These architectures are based on moduli sets chosen in order to optimally use the six inputs Look-Up Tables (LUTs) available in the Complex Logic Blocks (CLBs) of the new generation FPGAs. Experiments based on the implementation of Finite Impulse Response (FIR) filters characterized by different number of taps and wordlengths shows that the use the RNS together with suitable moduli sets optimally fits the six inputs LUTs of the last generation FPGAs architectures.

1. INTRODUCTION

The silicon integrated circuits trend is characterized by a steady reduction in the feature size combined with a steady rise in density and speed as shown in [1]. In the last twenty years FPGAs evolved rapidly in terms of complexity and architecture starting from the first FPGA, the Xilinx XC2064 chip with its 1,000 gates of complexity [2] to the newest generations. The major evolution was related to the structure of the interconnect, the topology of the basic cell (LE i.e. the Logic Element), and the introduction of full custom processing elements such as multipliers, hardware processor cores, MAC units, and very high speed serial I/O blocks. One of the last innovation in the FPGAs architecture, has been the introduction of 6-inputs LUTs as the main block for the implementation of combinatorial functions [4], [3]. Moreover, changes in the FPGAs architecture require changes in the synthesis algorithms in order to guarantee an optimum mapping on the available resources. In this paper, the use of a RNS representation based on suitable moduli sets is used to optimally implement the basic arithmetic operators by using six-inputs LUTs. In particular it is shown that the use of moduli that are represented by five bits offers the best results in terms of used resources and delay. For this reason the moduli set that has been used used for the synthesis experiments is composed by five bits moduli and the bigger modulo has been choses as a power of two. In this way dynamic ranges of up to 34 bits are obtained. The paper is organized as follows: in Section II a background on the RNS arithmetic is given. In Section III architectures and performance of 6-inputs LUT based implementations of modulo m arithmetic operators such as adders, constant multipliers and general multipliers are discussed. Section IV illustrates the implementation of the input and output converters, while in Section V a set of experiments based on the implementation of FIR filters are shown discussing the obtained area and speed results. Conclusion are drawn in Section VI.

2. BACKGROUND ON RESIDUE NUMBER SYSTEM

A Residue Number System (RNS) is defined by a set of relatively prime integers:

$$\{m_1, m_2, ..., m_P\}$$

The dynamic range of the system is given by the product of the moduli m_i

$$M = \prod_{i=1}^{P} m_i$$

Any integer $X \in [0, M-1]$ has a unique RNS representation given by

$$X \xrightarrow{RNS} (\langle X \rangle_{m_1}, \langle X \rangle_{m_2}, \ldots, \langle X \rangle_{m_P}) \qquad (1)$$

where $\langle X \rangle_{m_i} = X \bmod m_i$

A comprehensive description of the RNS theory and its application to computer systems can be found in [6], [7], and [8]. In the RNS representation, operations, such as addition and multiplication, are executed in parallel on the different moduli

$$Z = X \text{ op } Y \xrightarrow{\text{RNS}} \begin{cases} \langle Z \rangle_{m_1} = \langle X_{m_1} \text{ op } Y_{m_1} \rangle_{m_1} \\ \dots \\ \langle Z \rangle_{m_P} = \langle X_{m_P} \text{ op } Y_{m_P} \rangle_{m_P} \end{cases} \quad (2)$$

where eq. (2) is valid if the final results prior the conversion in the two's complement representation (TCS) belongs to the range $[0, M-1]$. The conversion of Z in TCS is accomplished by the Chinese Remainder Theorem (CRT)

Clearly, conversions the from the binary representation to RNS, and vice-versa, constitute an overhead for systems based on the RNS representation. However, efficient methods to perform those conversions have been presented in [9], [10], and [11].

The input conversion is obtained by the reduction modulo m_i of the input samples $x(n)$, providing the residue digits x_{m_i}. The mod. m_i RNS filters compute the residues y_{m_i} defined in eq. (4), while the output conversion based on CRT computes back $y(n)$.

3. MODULO OPERATIONS BASED ON SIX INPUTS LUTS

In FPGAs, the LEs are based on LUTs and, in particular, the last generation FPGAs are characterized by LEs containing six inputs LUTs (useful to implement six input one output combinatorial functions) that can be also configured as double 5 inputs LUTs (useful to implement five inputs double output combinatorial functions) ([4], [5]). Consequently, in the paper, the moduli set is chosen such that the moduli range belongs to the interval $[17, 64]$, moreover they must be coprime and usually it is convenient to use a power of two (such as 2^n with $n = 5$ or $n = 6$) as the bigger modulus. In the rest of the paper the following arithmetic blocks are analyzed

1. Modulo m adders
2. Constant multipliers (constant coefficients FIR filters)
3. General multipliers (variable coefficients FIR filters)

3.1. Modulo m adders

If a modulo m is chosen such that $2^{n-1} < m \leq 2^n$, the range of the results generated by operations mod. m are in the range $[0, 2^n - 1]$ and therefore n bits are used to represent the results. A fast architecture can be used to implement the addition modulus m as shown in Fig. 1

It is composed by a two operands adder, a three operands adder and a multiplexer. The two operands n-bits adder computes $S1 = X + Y$, the three inputs n-bits adder computes $S2 = X + Y - m$ and the 2×1 multiplexer selects $S1$ or $S2$ depending on the the carry out of $S2$.

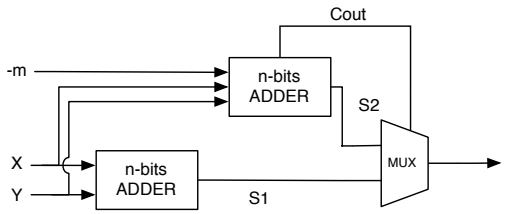

Fig. 1. The parallel modulo m adder.

In this paper a different architecture to compute $\langle X + Y \rangle_m$ is presented obtaining a delay comparable to that of the parallel architecture by using less resources. The architecture is shown in Fig. 2 where X and Y are added obtaining S. This value is used to address a ROM (based on six inputs LUTs) containing $\langle S \rangle_m$.

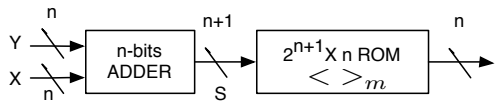

Fig. 2. The ROM based modulo m adder

For a 5 bits modulo, the size of the ROM is $2^6 \cdot 5$, corresponding to 5 six inputs LUTs, while for a 6 bits modulo the size of the ROM is $2^7 \cdot 6$, corresponding to 12 six inputs LUTs. The growth of the ROM size is exponential, but for m up to 64 this structure is slightly convenient with respect to the parallel implementation as shown in Table I. This table shows the synthesis results in terms of number of LUTs and delay for different values of five and six bits moduli in comparison with the parallel implementation.

m	Parallel mod. Adder		ROM based mod. Adder	
	delay(ns)	#LUT	delay(ns)	#LUT
19	1,59	15	1,53	10
31	1,62	15	1,55	10
35	1,77	18	1,77	18
63	1,71	16	1,62	15

Table 1. Area and delay of parallel and ROM based modulo adders implemented on a Xilinx Virtex V FPGA

In the case of five bits moduli this implementation gives 33% of resource savings maintaining the same delay, while for six bits moduli the results in term of used resources and delay are similar and there are no advantages.

3.2. Modulo m Multipliers: variable coefficients, constant coefficients

In this section, constant coefficients and general multipliers are analyzed.

1. **Modulo m constant multipliers.** They are used to implement RNS FIR filters with constant coefficients. If n is the number of bits to represent m, $\langle X \cdot K \rangle_m$ requires n output bits. If $n = 6$, it can be implemented by using a $2^6 \times 6$ ROM that, in the case a Xilinx Virtex V FPGA is implemented by using 6 six inputs LUTs with a critical path of about 0.8 ns.

2. **General multipliers.** In this case, being m a prime number (6 bit) the isomorphism technique [6] can be used to perform the multiplication. This technique is based on the algebraic properties of the structure composed by the modulo m addition and multiplication and the numbers in the interval $[0, m-1]$. In fact the ring is a finite field and therefore

   (a) each element different from zero has a multiplicative inverse

   (b) it exists an element of the field, called α, such as $\forall x \in [1, m-1] \; \exists \; i \mid \alpha^i = x$ and $\alpha^m = \alpha$

   The modulo m multiplication of two numbers becomes $\langle x \times y \rangle_m = \left\langle \alpha^i \times \alpha^j \right\rangle_m = \left\langle \alpha^{i+j} \right\rangle_m$. Because $\alpha^m = \alpha$ the addition of $i+j$ is performed modulo $m-1$. The architecture of the isomorphic multiplier is shown in Fig. 3.

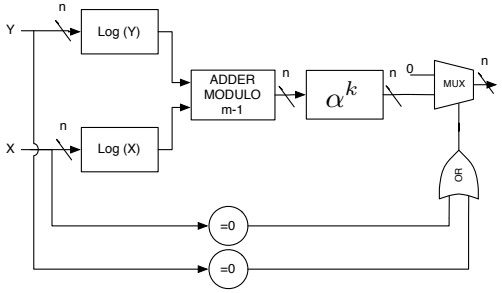

Fig. 3. The modulo m multiplier based on the isomorphism technique

The blocks named *Log* (based on LUTs) performs the association between the value X and Y and the corresponding indexes i and j, while the block α^k performs the inverse association between the result of $\langle i + j \rangle_{m-1}$ and the value α^k. Some additional logic allows resolving the case in which one or both the operands are zero. The modulo $m-1$ adder can be implemented by either the parallel and the ROM based modulo adder. If the ROM based modulo adder is used, two ROMS, the first performing the operation $\langle \, \rangle_{m-1}$, the second performing the inverse isomorphism, are used as shown in Fig. 4.

The two ROMs can be combined in a single ROM performing both the operations. This implementation requires

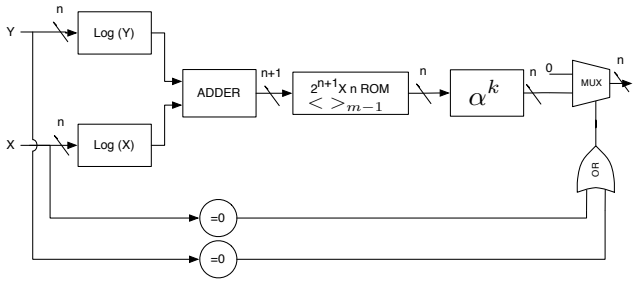

Fig. 4. The isomorphic based modulo m multiplier with ROM based Modulo addition

the use of about 30 LUTs, with a maximum delay of 3.04ns. Instead, the parallel implementation requires 36 LUTs with a maximum delay of 3.78ns. Therefore, by embedding the two ROMs in a single ROM the architecture is about 20% faster and shows a 15% of resource savings.

4. FIR FILTER IMPLEMENTATION

A N taps FIR filter is described by

$$y(n) = \sum_{k=0}^{N-1} h_k \cdot x(n-k) \tag{3}$$

Its fixed point implementation, in transposed or direct form, is obtained by using multipliers adders and registers. In particular, in parallel implementations, the reduction of the used resources is usually accomplished by truncating the multipliers outputs. The number of truncated bits is the result of a fixed point optimization phase that is based on a trade of between resource savings and signal to noise ratio worsening. The implementation of RNS FIR filters is a direct consequence of eq. (2) and eq. (3) becomes

$$\begin{aligned} \langle y(n) \rangle_{m_1} &= \left\langle \sum_{k=0}^{N-1} \left\langle \langle h_k \rangle_{m_1} \cdot \langle x(n-k) \rangle_{m_1} \right\rangle_{m_1} \right\rangle_{m_1} \\ &\dots \\ \langle y(n) \rangle_{m_P} &= \left\langle \sum_{k=0}^{N-1} \left\langle \langle h_k \rangle_{m_P} \cdot \langle x(n-k) \rangle_{m_P} \right\rangle_{m_P} \right\rangle_{m_P} \end{aligned} \tag{4}$$

The filter is implemented in RNS by decomposing it into P FIR filters working in parallel, as sketched in Fig. 5 (P=3).

4.1. Modulo m_i filters

The architecture of the mod m_i filters (based on eq. (4)) is depicted in Fig. 6. where, the shaded area, is filter basic building block (the mod. m_i tap).

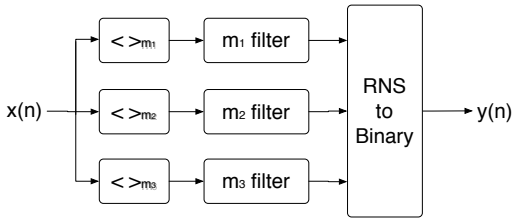

Fig. 5. RNS implementation of a FIR filter

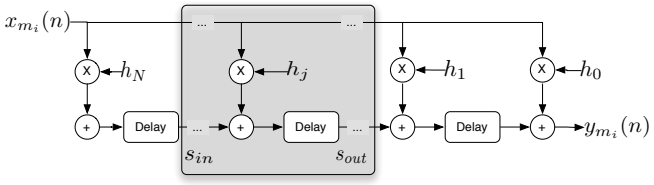

Fig. 6. Architecture of a modulo m_i FIR filter

The filter tap computes the following equation

$$s_{out}(j) = x_{m_i} \cdot h_j + s_{in} \tag{5}$$

where $s_{in} = s_{out}(j-1)$ and $m_i \neq 2^n$. Also in this case, the filter tap has been optimized by using a method similar to that used for the general multiplier presented in the previous section. Moreover, in the following, the analysis is restricted to moduli being prime numbers in order to make it possible the use of the isomorphism technique. For constant coefficients filters, the filter tap (Fig.6) requires a ROM and a modular adder that can be either a parallel or a ROM based adder. Equation 5 can be rewritten as

$$s_{out}(j) = h_j \cdot (x_{m_i} + h_j^{-1} \cdot s_{in}) = h_j \cdot \tilde{s}_{out} \tag{6}$$

where $\tilde{s}_{out} = x_{m_i} + h_j^{-1} \cdot s_{in}$ and h_j^{-1} is the multiplicative inverse of h_j mod. m_i.

For consecutive slices the filter coefficients h_{j+1}^{-1} and h_j can be combined as

$$\begin{aligned} s_{out}(j+1) &= h_{j+1} \cdot \left(x_{m_i} + h_{j+1}^{-1} \cdot s_{out}(j)\right) = \\ & h_{j+1} \cdot \left(x_{m_i} + h_{j+1}^{-1} \cdot h_j \cdot \left(x_{m_i} + h_j^{-1} \cdot s_{out}(j-1)\right)\right) \\ &= h_{j+1} \cdot \left(x_{m_i} + \tilde{h}_j \cdot \tilde{s}_{out}(j)\right) \end{aligned} \tag{7}$$

where $\tilde{h}_j = h_{j+1}^{-1} \cdot h_j$. In this way for the intermediate slices the tap can be implemented as depicted in Fig. 7.

The operation $\left\langle \tilde{h}_j \cdot (x_{m_i} + \tilde{s}_j) \right\rangle_{m_i}$, where $\tilde{h}_j$ is a constant factor is implemented by using a ROM based modular adder. In the ROM precomputed values of $\left\langle \tilde{h}_j \cdot (x_{m_i} + \tilde{s}_j) \right\rangle_{m_i}$

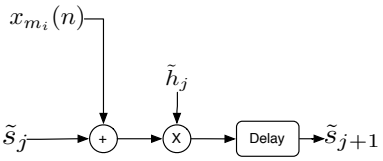

Fig. 7. Optimized slice of a modulo m_i FIR filter with constant coefficients

are memorized. For a 5 bits modulo the resource usage is 10 LUTs and the delay is about 1,5 ns, while for a 6 bits modulo the resource usage is around 16 LUTs and the delay is about 1,7 ns. In Fig. 8 the architecture of a tap in case of variable coefficients filter is shown.

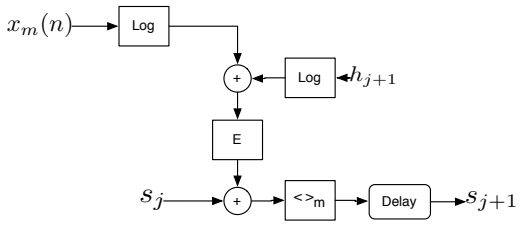

Fig. 8. Optimized architecture of a slice for a modulo m_i variable coefficients FIR filter

The *Log* operators are implemented by $2^n \times n$ ROMs, the E operator is a $2^{n+1} \times n$ ROM performing modulo reduction and exponentiation, the $\langle \, \rangle_{m_i}$ operator is ROM based, the adders are n-bits adders, while the critical path is composed by two adders and three ROMs. The first optimization consists in sharing the Log operator that is the same for all the slices composing the modulo m_i filter. The second optimization is obtained by balancing the paths of the slices moving the ROM implementing the $\langle \, \rangle_{m_i}$ operator after the delay element. In this way the critical path is reduced to two ROMs and two adders.

5. FIR FILTERS EXPERIMENTS

In this section a set of experiments for the characterization of FIR filters are described.

Two cases have been selected: 8 bits and 12 bits both for the coefficients and input samples while, the number of filter taps vary from 16 to 256. The analysis has been restricted to constant coefficient filters, but it can be easily extended to variable coefficient filters. The set of moduli is composed by a power of two modulo (2^n, n up to 9) and the remaining moduli are prime numbers that can be represented by 5 bits. In table II the set of synthesized filters are shown.

For dynamic ranges up to 23 bits 4 moduli have been used while for the biggest dynamic range (32 bits) 7 moduli

FIR	Input/Coeff (bits)	N. taps	M(Bits)	Moduli set
FIR1	8	16	20	64,31,29,23
FIR2	8	32	21	128,31,29,23
FIR3	8	64	22	256,31,29,23
FIR4	8	128	23	512,31,29,23
FIR5	8	256	24	64,31,29,23,19
FIR6	12	16	28	64,31,29,23,19,17
FIR7	12	32	29	128,31,29,23,19,17
FIR8	12	64	30	256,31,29,23,19,17
FIR9	12	128	31	512,31,29,23,19,17
FIR10	12	256	32	64,31,29,23,19,17,13

Table 2. Description of the set of FIR filters synthesis experiments

are required. In Table III the results in terms of resources and speed performances for the set of synthesized filters are listed. The maximum frequency for the 8 bits filters (from FIR1 to FIR5) is bounded by the maximum operating frequency of the filter tap (about 435 MHz), while for the 12 bits filters (from FIR6 to FIR10) the maximum working frequency of the filter is limited by the input converter speed (300 MHz).

FIR	Max. freq. (MHz)	Taps (#LUTs)	In converter (#LUTs)	Out converter (#LUTs)	Total resources (#LUTs)
FIR1	400	592	30	182	804
FIR2	400	1248	30	182	1460
FIR3	400	2688	30	182	2900
FIR4	400	6144	30	182	6356
FIR5	400	12032	40	224	12296
FIR6	303	912	70	270	1252
FIR7	303	1888	70	270	2228
FIR8	303	3968	70	270	4308
FIR9	303	8704	70	270	9044
FIR10	303	17152	84	309	17545

Table 3. Resource usage and speed for the experiments

In this table, the resources for the implementation of the input and output converters have been evaluated showing that it become less than 10% for $N > 64$ (see FIR3 and FIR8).

Finally, the results of the synthesis of the RNS filters have been compared to a TCS implementation (no truncation). As indicated in section II usually truncation is used in order to limit the resources in TCS filters but it has been shown in the literature [15] that truncation do not offset the advantages of a RNS implementation. Moreover, the RNS representation is often used to design filters with error detection and correction capabilities [13], [14]). If truncation is used, error detection techniques cannot be used. The results are presented in table IV.

The resource savings obtained by using RNS are always greater than 30% when the dynamic range of the input data is 12 bits, while in case of 8 bits the advantage depends on the number of taps. For the FIR1 there are no savings but a small increment in the resources usage due to the overhead of the conversion blocks but savings up to 20% are obtained for FIR5 and FIR 3 experiments. The experimental results shows that the presented techniques offer interesting advantages for FIR filters characterized by high dynamic range and high number of taps especially when full custom multipliers are not available in the target FPGA architecture or when they must to be used for different purposes.

Exp. Name	canonical (#LUTs)	RNS (#LUTs)	saving (%)
FIR1	788	804	-2
FIR2	1800	1460	18
FIR3	3632	2900	20
FIR4	6966	6356	8
FIR5	15203	12296	19
FIR6	1899	1252	34
FIR7	3338	2228	33
FIR8	6555	4308	34
FIR9	14043	9044	35
FIR10	29234	17545	40

Table 4. Comparison of RNS and TCS filters

6. CONCLUSION

The optimization of Residue Number System (RNS) arithmetic to better exploit some of the architectural characteristic of the last generation FPGAs has been presented. Using an approach based on ROM modular adders different optimization techniques for the basic modular operations and for the basic blocks of RNS filters has been discussed. The choice of 5-bit moduli allows to implement high speed, low resource occupation RNS filters, as shown in the set of experiments discussed in the paper.

7. REFERENCES

[1] "2007 International Technology Roadmap for Semiconductors", http://public.itrs.net/.

[2] http://www.xilinx.com/company/history.htm#begin

[3] http://www.altera.com/products/devices/stratix3/st3-index.jsp

[4] Virtex-5 Family Overview LX, LXT, and SXT Platforms

[5] Logic Array Blocks and Adaptive Logic Modules in Stratix III Devices chapter in volume 1 of the StratixIII Device Handbook.

[6] I. Vinogradov, An Introduction to the Theory of Numbers. New York: Pergamon Press, 1955.

[7] N. Szabo and R. Tanaka, Residue Arithmetic and its Applications in Computer Technology. New York: McGraw-Hill, 1967.

[8] M. Sodestrand, W. Jenkins, G. A. Jullien, and F. J. Taylor, Residue Number System Arithmetic: Modern Applications in Digital Signal Processing. New York: IEEE Press, 1986.

[9] T. V. Vu, "Efficient implementation of the chinese remainder theorem for sign detection and residue decoding", IEEE Trans. Circuits Systems-I, vol. 45, pp. 667-669, June 1985.

[10] S.Piestrak, "A high-speed realization of a residue to binary number system converter", IEEE Trans. Circuits Systems-II Analog and Digital Signal Processing, vol. 42, pp. 661-663, Oct. 1995.

[11] G. Cardarilli, M. Re, and R. Lojacono, "A residue to binary conversion algorithm for signed numbers", European Conference on Circuit Theory and Design (ECCTD97), vol. 3, pp. 1456-1459, 1997.

[12] S. Bandyopadhyay, G.A. Jullien, A. Sengupta, A Systolic Array for Fault

[13] Mark H. Etzel and W. K. Jenkins Redundant Residue Number Systems for Error Detection and Correction in Digital Filters, IEEE Transactions on Acoustics, Speech and Signal Processing, vol. ASS-28, No 5, pp. 538- 544, October 1980.

[14] S. Pontarelli, G.C. Cardarilli, M. Re, A. Salsano "Totally Fault Tolerant RNS based FIR Filters", to be published in IEEE International On-Line Testing Symposium 2008.

[15] A. Nannarelli, M. Re, G. C. Cardarilli, "Tradeoffs between Residue Number System and Traditional FIR Filters", **IEEE** International Symposium on Circuits and Systems, ISCAS 2001, Vol. II, pp. 305-308, Sydney (Australia), May 6-9, 2001.