J Sign Process Syst
DOI 10.1007/s11265-010-0537-y

Optimized Implementation of RNS FIR Filters Based on FPGAs

Salvatore Pontarelli · Gian Carlo Cardarilli · Marco Re · Adelio Salsano

Received: 12 November 2008 / Revised: 13 September 2010 / Accepted: 17 September 2010

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, modulo m constant and general multipliers, input and output converters are presented. These architectures are based on moduli sets chosen in order to optimally use the 6-input 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 of RNS together with suitable moduli sets optimally fits the 6-input LUTs in the last generation FPGAs architectures.

Keywords Residue Number System · FPGA · FIR filters

S. Pontarelli (✉)
(ASI) Italian Space Agency, Viale Liegi, 26 00198 Rome, Italy
e-mail: pontarelli@ing.uniroma2.it

G. C. Cardarilli · M. Re · A. Salsano
Department of Electronic Engineering, University of Rome "Tor Vergata", Via del Politecnico 1, 00191, Rome, Italy

G. C. Cardarilli
e-mail: g.cardarilli@ieee.org

M. Re
e-mail: marco.re@ieee.org

A. Salsano
e-mail: salsano@ing.uniroma2.it

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 (http://www.xilinx.com/company/history.htm) to the newest generations. The major evolution was related to the structure of the interconnect, the topology of the basic cell i.e., the Logic Element (LE), 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 LE architecture has been the introduction of 6-inputs LUTs as the main block for the implementation of combinatorial functions [2]. In this paper it is shown how this characteristic is exploited when a RNS representation based on suitable moduli sets is used for the implementation of the elementary arithmetic operators. In fact, changes in the FPGAs architecture require changes in the synthesis algorithms in order to guarantee an optimum mapping on the available resources.

Even if FPGA implementation of RNS systems has been proposed in literature such as [3–5], these papers do not focuses the attention of the relationship between the choice of moduli set, the architectural choice of the basic arithmetic blocks of RNS systems and the result achievable on the FPGA.

In [6] the FPGA implementation of RNS filter on an FPGA by using moduli that are power of two or power of two minus one is presented. The paper exploit additional RAM block available on FPGA to implement

Published online: 30 September 2010

forward and reverse conversion and also coefficient multiplication. The drawbacks of this approach are:

- the limited number of RAM blocks available on FPGA. In fact, the number of used block RAM is dependent from the number of taps of the filter and therefore this method cannot be applied for filter with an high number of taps,
- no effort has been done to select the implementation of modular adder that better exploit the FPGA characteristic,
- no optimization of the whole structure of the modulo FIR filters has been proposed

Also here the block RAMs available on the FPGA are exploited but no effort in the choice of the moduli set that best fit the FPGA resources has been done. In this paper an analysis of the basic building blocks of RNS filter (modulo addition and multiplications) is carried out with emphasis of the relationship between the target technology (the six input LUTs) on one hand and the chosen architectures and moduli set on the other hand. After this analysis the design methodology of an optimized RNS filter is presented. The paper is organized as follows: in Section 2 a background on the RNS arithmetic is given. In Section 3 architectures and performance of 6-inputs LUT based implementations of modulo m arithmetic operators such as adders, constant multipliers and general multipliers are discussed together with a comparison with implementations based on 4-input LUTs. Section 4 illustrates the implementation of the RNS input and output converters and discusses the obtained area and speed results. Conclusions are drawn in Section 5.

2 Background on Residue Number System

A Residue Number System (RNS) composed of P moduli is defined by a set of P 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

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

where $\langle X \rangle_{m_i} = X \bmod m_i$. In the paper we also use the compact notation X_{mi}.

A comprehensive description of the RNS theory and its application to computer systems can be found in [7, 8], and [9]. In the RNS representation, operations, such as addition and multiplication, are executed in parallel on the different moduli

$$Z = X \text{ op } Y \xrightarrow{\text{RNS}} \begin{cases} Z_{m_1} = \langle X_{m_1} \text{ op } Y_{m_1} \rangle_{m_1} \\ \ldots \\ Z_{m_P} = \langle X_{m_P} \text{ op } Y_{m_P} \rangle_{m_P} \end{cases} \tag{2}$$

where Eq. 2 is valid until the results that must be converted in the Two's Complement representation (TCS) are in the range $[0, M-1]$. The conversion in TCS is accomplished by the Chinese Remainder Theorem (CRT)

$$Z = \left\langle \sum_{i=1}^{P} \langle Z_{m_i} \cdot k_i \rangle_{m_i} \cdot M_i \right\rangle_M \tag{3}$$

with $M_i = \frac{M}{m_i}$ and k_i, the multiplicative inverses, are obtained by $\langle M_i \cdot k_i \rangle_{m_i} = 1$.

Clearly, conversions 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 studied [10–12].

3 Modulo m Operations Based on 6-input LUTs

It is well known that the architecture of an FPGA is based on an array of logic blocks interconnected in a flexible way by using programmable interconnection resources. The basic logic resources on FPGAs are the LEs. In FPGAs LEs are often realized by using LUTs. The LEs of the last generation FPGAs are based on 6-input LUTs (useful to implement 6-input one output combinatorial functions) that can be also configured as double 5-input LUTs (useful to implement five inputs double output functions) [2, 13]. On the other hand it is well known that RNS arithmetic is particularly suitable for LUTs based implementations because it is based on computations that uses several small value moduli working in parallel (Section 2). So FPGAs, and in particular the new families of FPGAs are very good candidates for the implementation of RNS based arithmetic units. An optimum use of the FPGA resources in RNS implementations can be consequently obtained by a suitable selection of the moduli set i.e. by choosing m_i such that they can be represented by using maximum 5 or 6 bits (i.e. moduli in the range [17, 64]). In the

rest of the paper the following arithmetic blocks are analyzed

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

3.1 Modulo m Adders

In this subsection modular adders are addressed. Specifically the two most used architectures are illustrated and a new architecture based on LUTs is proposed. If a modulus m is chosen such that $2^{n-1} < m \leq 2^n$, the range of the results of operations mod. m is $[0, 2^n - 1]$ and therefore n bits are used to represent the intermediate results. The implementation of the operation $\langle X + Y \rangle_m$ is usually realized by the computation of the two intermediate results $S1 = X + Y$ and $S2 = X + Y - m$. Two different architectures can be used as shown in Fig. 1

1. **Serial architecture:** it is composed by two n-bits adders and a multiplexer (Fig. 1a). The first adder computes $S1 = X + Y$, the second one uses $S1$ to compute $S2 = S1 - m$ and the 2×1 multiplexer selects $S1$ or $S2$ depending on the carry out of $S2$. If the carry is one then $S1 > m$ and $S2$ is selected as output, otherwise $S1$ is selected.
2. **Parallel architecture:** it is composed by a two operands adder, a three operands adder and a multiplexer (Fig. 1b). The two operands n-bits adder computes $S1 = X + Y$, the three inputs n-bits adder directly computes $S2 = X + Y - m$ and the 2×1 multiplexer selects $S1$ or $S2$ depending on the the carry out of $S2$. This structure requires more logic resources but due to the parallelism is faster than the serial one.

In this section a different architecture is presented to compute $\langle X + Y \rangle_m$ obtaining a delay comparable to that of the parallel architecture by using the same resources of the serial one. The architecture is shown in Fig. 2. The inputs X and Y are added obtaining S, then the mod. m operator is implemented by using a ROM. In the ROM locations, for each value of S, the corresponding value $S \; mod. \; m$ is memorized.

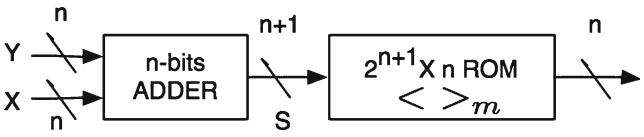

Figure 2 The ROM based modulo m adder.

For a 5 bits modulus the ROM can be directly implemented by using 6-input LUTs, in fact, the ROM size is $2^6 \cdot 5$ corresponding to five 6-input LUTs, while for a 6 bits modulo the size of the ROM is $2^7 \cdot 6$, corresponding to 12 6-input LUTs. The growth of the ROM size is exponential, but for moduli up to 64 this structure is slightly convenient with respect to the parallel implementation as shown in Table 1. This table shows the synthesis results in terms of number of LUTs and delay for different values of 5 and 6 bits moduli in comparison with the parallel modulo adder implementation.

In the case of 5 bits moduli the ROM-based implementation requires 33% fewer resources and the same delay of the parallel one, while for 6 bits moduli the comparison shows similar numbers between the two implementations. Moreover, it can be noticed that the synthesis tool simplifies the architecture of the modulo m adder when $m = 2^n$. The synthesizer uses LUTs also in the generation of the adders so, for a 5 bits adder 5 LUTs to implement the FAs are used while the carry chain is implemented by using fast carry chain resources available in the FPGA. Obviously, depending on the modulus value, the synthesizer is able to optimize the resources and the final number of LUTs is, in some cases, smaller than the value obtained by summing the number of LUTs for the ROM based modular extraction plus those used for the adder implementation.

Now we compare the used resources and delay of modular adders implemented on FPGA based on 4 and 6-input LUTs. The performance of papallel and

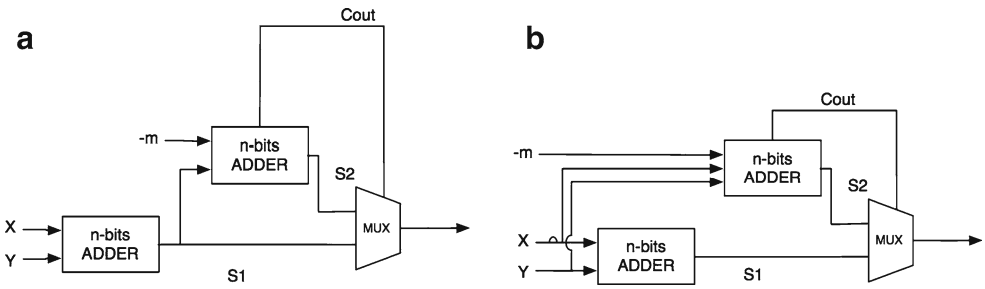

Figure 1 **a** The serial modulo m adder **b** The parallel modulo m adder.

Table 1 Area and delay of parallel and ROM based modulo adders implemented on a Xilinx Virtex V FPGA (6-input LUTs).

Number of bits	m	Parallel mod. adder		ROM based mod. adder	
		Delay (ns)	#LUT	Delay (ns)	#LUT
5	17	1.58	15	1.53	10
5	19	1.59	15	1.53	10
5	23	1.59	15	1.53	10
5	29	1.61	15	1.53	10
5	31	1.62	15	1.55	10
5	32	0.68	5	0.68	5
6	33	1.76	15	1.72	16
6	35	1.77	18	1.77	18
6	61	1.73	15	1.64	16
6	63	1.71	16	1.62	15
6	64	0.90	6	0.90	6

ROM based adders are presented in Table 1 for 6-input LUTs and in Table 2 for 4-input LUTs. The results shows that in the case of 4-input LUTs the parallel modulo adder is better because it exploits the use of the additional carry logic available in FPGAs, while the implementation based ROMs requires to interconnect several LUTs to achieve the address space. Instead, for 6-input LUTs implementations, the number of LUTs is reduced, compensating the advantage of the carry logic used in the parallel adder implementation. This result shows that for the dynamic ranges useful in DSP implementations the new FPGAs based on 6-input LUTs offer interesting advantages in the implementation of RNS arithmetic units.

3.2 Modulo m Multipliers: Constant Coefficients

In this section constant coefficients and general multipliers are presented. They are used to implement RNS FIR filters with constant coefficients. If n is the number of bits used to represent m, $\langle X \cdot K \rangle_m$ requires n output bits. If $n = 6$, it can be realized by using a $2^6 \times 6$ ROM that, in the case a Xilinx Virtex V FPGA is implemented by addressing six 6-input LUTs with a delay of about 0.8 ns. Due to extreme simplicity of such structure no optimization can be performed at the level of the single coefficient.

Table 2 Area and delay of a parallel and ROM based modulo adders implemented on a Xilinx Virtex IV (LE based on 4-input LUTs).

Number of bits	m	Parallel mod adder		ROM based mod adder	
		#LUT	Delay (ns)	#LUT	Delay (ns)
5	17	16	3.187	23	3.088
5	19	16	3.187	24	3.086
5	23	16	3.187	23	3.088
5	29	16	3.187	22	3.084
5	31	16	3.187	23	3.086
5	32	5	1.525	5	1.525
6	33	19	3.231	44	3.763
6	35	19	3.231	46	3.853
6	61	19	3.231	37	3.550
6	63	19	3.231	33	3.172
6	64	6	1.559	6	1.559

3.3 Modulo m Multipliers: Variable Coefficients

In this subsection are analyzed the implementation of general multipliers using a 6-input LUTs FPGA. If m is a prime number (6 bit) the isomorphism technique [7] 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 of characteristic m and therefore

1. each element different from zero has a multiplicative inverse
2. 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 $\langle x \times y \rangle_m = \langle \alpha^i \times \alpha^j \rangle_m = \langle \alpha^{i+j} \rangle_m$ and because $\alpha^m = \alpha$ the addition of $i + j$ is performed mod. $m - 1$.

The architecture of the isomorphic multiplier is shown in Fig. 3. 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 manages the case in which one or both the operands are zero. The mod. $m - 1$ adder can be implemented by using either the parallel and the ROM

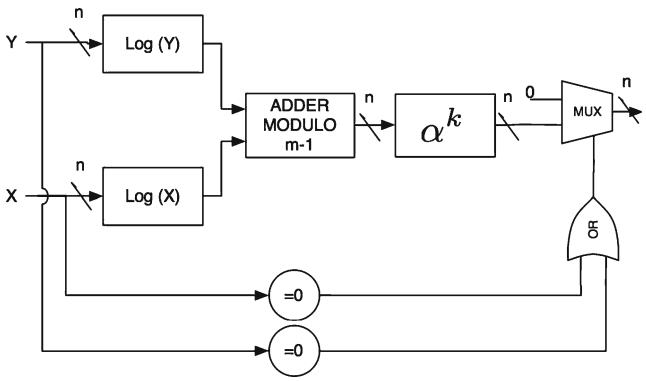

Figure 3 The modulo m multiplier based on the isomorphism technique.

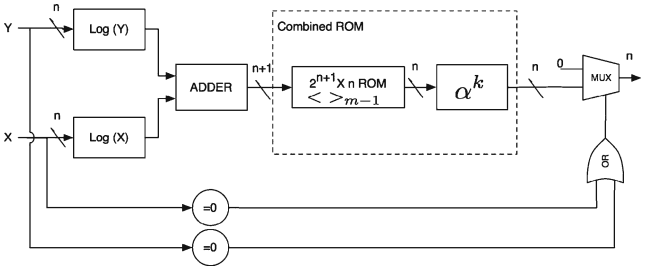

Figure 4 The isomorphic based modulo m multiplier with ROM based modulo addition.

based modulo adder. If the ROM based modulo adder is used, the two ROMs, the first one performing the operation $\langle\cdot\rangle_{m-1}$, the second one the inverse isomorphism, can be combined in a single ROM obtaining an optimization with respect to the multiplier based on the parallel modulo adder (Fig. 4).

Synthesis results show that this implementation requires the use of about 30 LUTs and a delay of about 3 ns instead while the parallel implementation requires 36 LUTs with a delay of 3.8 ns. Therefore, by combining the two ROMs 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

$$y(n) = \sum_{k=0}^{N-1} h_k \cdot x(n-k) \tag{4}$$

Its fixed point implementation, in transposed or direct form, is obtained by using multipliers adders and registers and, 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-off between resource savings and signal to noise ratio worsening [14]. The implementation of RNS FIR filters is a direct consequence of Eqs. 2 and 4 obtaining

$$\begin{aligned} \langle y(n)\rangle_{m_1} &= y_{m_1} = \left\langle \sum_{k=0}^{N-1} \left\langle \langle h_k\rangle_{m_1} \cdot \langle x(n-k)\rangle_{m_1} \right\rangle_{m_1} \right\rangle_{m_1} \\ &\dots \\ \langle y(n)\rangle_{m_P} &= y_{m_P} = \left\langle \sum_{k=0}^{N-1} \left\langle \langle h_k\rangle_{m_P} \cdot \langle x(n-k)\rangle_{m_P} \right\rangle_{m_P} \right\rangle_{m_P} \end{aligned} \tag{5}$$

i.e. the filter is implemented by P FIR filters working in parallel, as sketched in Fig. 5 (P = 3).

The input conversion is obtained by the reduction modulo m_i of $x(n)$, providing the residue digits x_{m_i}. The mod. m_i RNS filters compute the residues y_{m_i} defined in Eq. 5, while the output conversion based on Eq. 3 computes back $y(n)$. The implementation of FIR filters based on the RNS representation requires the following blocks

1. Binary to RNS converter
2. Mod. m_i FIR filters
3. RNS to binary converter

In this section these components are analyzed and an optimized RNS filter is proposed.

4.1 Input Converter

The input converter is composed by P independent blocks that compute

$$x_{m_i} = \langle x(n)\rangle_{m_i} = \left\langle \sum_{k=0}^{L-1} 2^k \cdot x_k(n) \right\rangle_{m_i} \tag{6}$$

where L is the number of bits used to represent the input samples. This equation can be rearranged in different ways depending on how the modulo operator is applied to the equation. The basic technique that can be used in a ROM based approach, is to partition the L bits input word into S sub-blocks each one composed by b bits. Each group of b bits will address a ROM. In our case, to fully exploit the characteristics of 6-input LUTs, up to 6 bits moduli can be used but the best performances are obtained by using 5 bits moduli. Consequently, in the following example where a 15 bits input converter is used to illustrate the ROM based

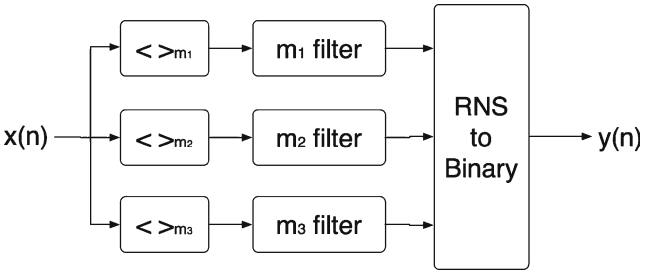

Figure 5 RNS implementation of a FIR filter.

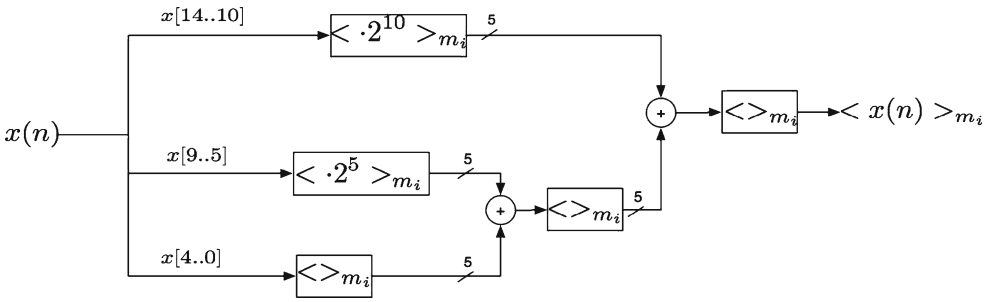

Figure 6 Architecture of a 15 bits modulo m reduction block.

technique, $b = 5$. In this example the reduction mod. m_i is computed by

$$x_{m_i} = \left\langle \left\langle \langle x_{[4..0]} \rangle_{m_i} + \langle x_{[10..5]} \cdot 2^5 \rangle_{m_i} \right\rangle_{m_i} + \langle x_{[14..10]} \cdot 2^{10} \rangle_{m_i} \right\rangle_{m_i} \tag{7}$$

The converter architecture is shown in Fig. 6 and it is implemented by using three $2^5 \times 5$ bits ROMs (9 LUTs, in fact the 6-input LUTs can be reconfigured as two out 5-input LUTs), two $2^6 \times 5$ ROMs (corresponding to 10 LUTs) and two adders, while the critical path is composed by three ROMs and two five bits adders.

By manipulating Eq. 7,

$$x_{m_i} = \left\langle x_{[4..0]} + \langle x_{[10..5]} \cdot 2^5 \rangle_{m_i} + \langle x_{[14..10]} \cdot 2^{10} \rangle_{m_i} \right\rangle_{m_i}$$

an optimized version (Fig. 7) of the modulo reduction block is obtained.

In this case, the used resources are: two $2^5 \times 5$ ROMs (6 LUTs), one $2^7 \times 5$ ROM (10 LUTs) and two adders. The number of LUTs to implement the ROMs decreases from 19 in the first architecture to 16 while the critical path is now composed by a six bits adder, a seven bits adder and two levels of ROMs.

4.2 Modulo m_i Filters

The filters mod m_i implementing Eq. 5 are depicted in Fig. 8. The shaded area in Fig. 8 is the basic building block i.e. the mod. m_i tap of the filter, x_{m_i} derives from the input converter and s_{in} is the output of the previous tap while the output of the tap is s_{out}.

The filter tap implements the equation

$$s_{\text{out}}(j) = x_{m_i} \cdot h_j + s_{\text{in}} = x_{m_i} \cdot h_j + s_{\text{out}}(j-1) \tag{8}$$

where $s_{\text{in}} = s_{\text{out}}(j-1)$ and the filter coefficients are $h_j = \langle h_j \rangle_{m_i}$. As stated above, the moduli set is composed by prime numbers and by one power of two modulus. In the case $m_i = 2^n$ the tap implementation resources and delay performance are shown in Table 3.

The modulo m_i tap, when $m_i \neq 2^n$, has been optimized by using a method similar to that used for the general multiplier presented in the previous section. In the following, the analysis is restricted to moduli being prime numbers to permit the use of the isomorphism technique. For constant coefficients filters, the tap (Fig. 8) requires a ROM and a modular adder that can be either a parallel or a ROM based adder.

Equation 8 can be rewritten as

$$s(j) = h_j \cdot \left(x_{m_i} + h_j^{-1} \cdot s(j-1) \right) \tag{9}$$

where h_j^{-1} is the multiplicative inverse of h_j mod. m_i.

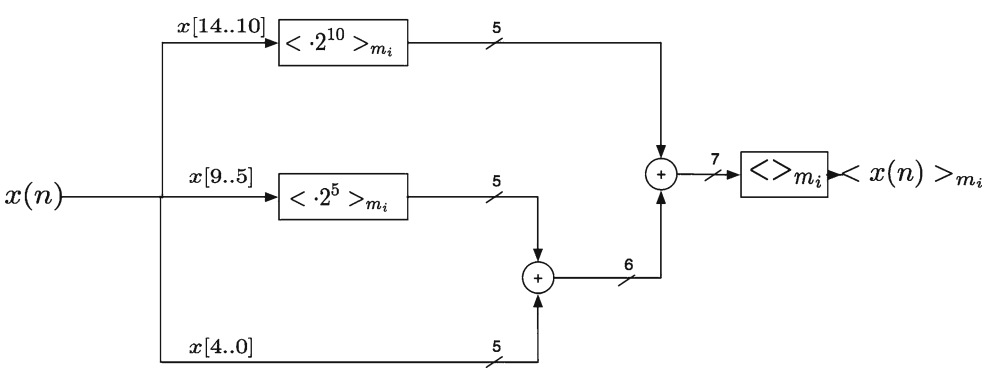

Figure 7 Optimized implementation of a 15 bits modulo reduction block.

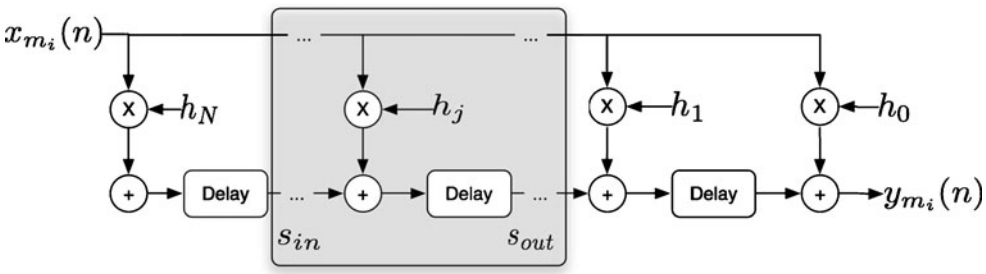

Figure 8 Architecture of a modulo m_i FIR filter.

Defining $\tilde{s}(j) = h_{j+1}^{-1} \cdot s(j)$ and substituting $\tilde{s}(j)$ and $\tilde{s}(j-1)$ in Eq. 9 we obtain:

$$h_{j+1} \cdot \tilde{s}(j) = h_j \cdot \left(x_{m_i} + \tilde{s}(j-1)\right) \tag{10}$$

and defining $\tilde{h}_j = h_{j+1}^{-1} \cdot h_j$ Eq. 10 becomes:

$$\tilde{s}(j) = \tilde{h}_j \cdot \left(x_{m_i} + \tilde{s}(j-1)\right) \tag{11}$$

In this way for the intermediate slices the tap can be implemented as depicted in Fig. 9.

The optimized slice of the modulo m_i filter is implemented using a ROM based modular adder. The inputs of the adders are $x_{m_i}(n)$ and $\tilde{s}(j-1)$ and the output of the sum addresses the ROM. The ROM stores the values of $\left\langle \tilde{h}_j \cdot (x_{m_i} + \tilde{s}(j-1)) \right\rangle_{m_i}$.

For a 5 bits modulo the resource usage is 10 LUTs (five 6-input LUTs for the modulo extraction and 5 for the adder implementation) and the delay is about 1.5 ns, while for a 6 bits modulo the resource usage is about 16 LUTs and the delay is about 1.7 ns. In the case of a variable coefficient filter two different optimizations can be used: to reduce the resources and to reduce the critical path. Figure 10 shows a tap for the variable coefficients filter.

The *Log* operators are implemented by $2^n \times n$ ROMs, the α^k 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.

4.3 Reverse Converter

The reverse conversion is based on Eq. 3 and requires three steps

1. computation of $\left\langle Z_{m_i} \cdot k_i \right\rangle_{m_i} \cdot M_i$. It is realized by using P ROMs of size $2^n \times l$, where n is the number of bits used to represent the modulo and $l = \lceil \log_2(M) \rceil$. As described before, n is fixed to 5 bits, while the values of P and l depend on the dynamic range of the FIR filter to be implemented. By using $P = 8$ a dynamic range of up to 33 bits is obtained by using the set of prime numbers $\{31, 29, 23, 19, 17, 13, 11, 7\}$.
2. addition of the P values from step 1. It is implemented by an adder tree (36 bits when $P = 8$).
3. reduction modulo M of the the value obtained in step 2. This step is implemented in a way similar to that used for the modulo reduction in the input converter.

For $P = 8$ the reduction is performed as follows:

$$y_M = \left\langle \left\langle y_{[32..0]} \right\rangle_M + \left\langle y_{[35..33]} \cdot 2^{33} \right\rangle_M \right\rangle_M$$

Table 3 Area and delay for a tap in the case $m = 2^n$.

n	# LUTs	Delay (ns)
5	5	0.6
6	7	0.72
7	9	1.25
8	12	1.25
9	18	1.38
10	50	3.22

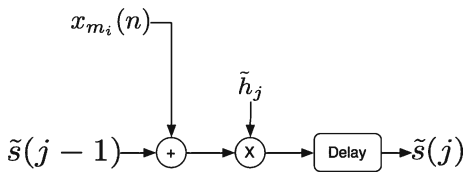

Figure 9 Optimized slice of a modulo m_i FIR filter with constant coefficients.

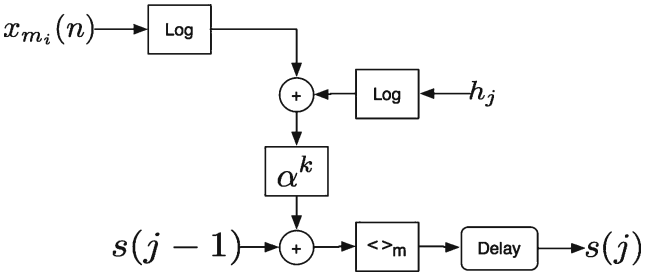

Figure 10 Optimized architecture of a slice for a modulo m_i variable coefficients FIR filter.

where $y_{[35..0]}$ is the output of the step 2. The most significant bits of the summation are therefore multiplied for the constant value 2^{33} modulo M using a $2^3 \times 33$ ROM and the last modulo M addition is performed by using a parallel modulo adder.

The architecture of the output converter is shown in Fig. 11. Each of the 8 ROMs has been implemented by using 17 LUTs configured in dual output mode, while the output LUT containing the term $\langle \cdot 2^{l+1} \rangle_M$ requires 17 LUTs. The adder tree and the final parallel modulo adder require 222 LUTs and so the final number of LUTs is 358. The delay of the circuit is 5.149 ns, corresponding to a maximum clock frequency of 194 MHz. To speed up the architecture, a pipelined version of the CRT has been implemented by using a four stage pipeline. The obtained delay is 1.829 ns corresponding to a maximum frequency of 545 MHz.

4.4 FIR Filters Experiments

A set of experiments for the characterization of FIR filters implemented in RNS by using the techniques presented in the paper are described. The experiments set up is

- **Type of filter:** Full precision arithmetic (no truncation), transposed form filters.
- **Coefficients/Input Samples wordlength:** 8, 12 bits,
- **Number of Taps:** from 16 to 256,
- **Moduli Set:** 5 bits prime numbers, the bigger modulus is a power of two. The use of a power of two modulus permits the simplification of the filter mod. 2^n (Table 3) and of the input and output converters.

In Table 4 the dynamic range of the filter and the chosen moduli set are shown.

For dynamic ranges up to 23 bits four moduli have been used while for the biggest dynamic range (32 bits) a maximum of seven moduli is required.

In Table 5 the performance of the filters in terms of speed and resource usage are shown. The maximum frequency of the 8 bits filters (FIR1 to FIR5) is bounded by the maximum operating frequency of the slice (about 435 MHz), while for the 12 bits filters (FIR6 to FIR10) the maximum frequency of the filter is limited by the output converter speed (about 300 MHz). This drawback can be easily overcome by a one level pipelining in the output converter.

In Fig. 12 the overhead due to the input and output converters is shown. Figure 12 show a comparison between the number of LUTs required by the converters and those required by the overall filter while in Fig. 13 the same results are represented in percentage.

Experiments FIR3 to FIR8 and FIR6 to FIR10 show that the area overhead due to the conversion becomes less than 10% when more than 64 taps are implemented

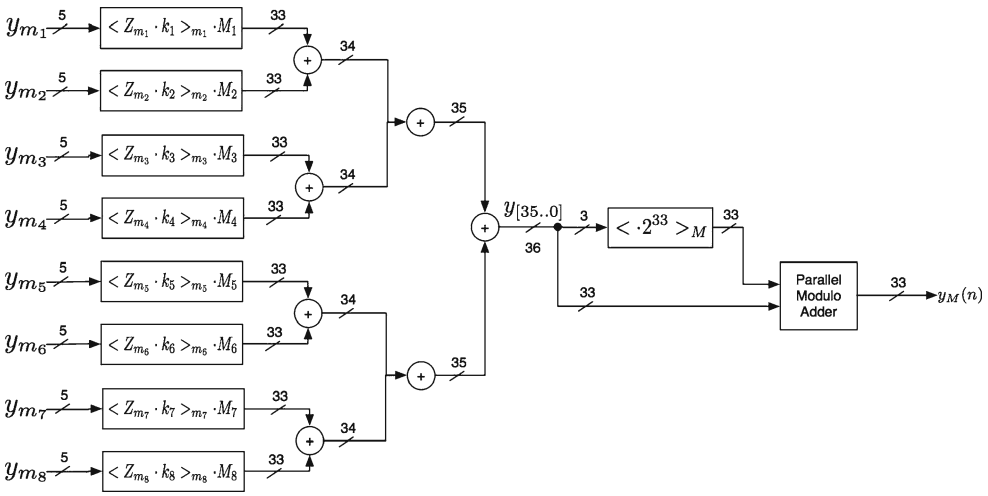

Figure 11 Implementation of the output converter.

Table 4 Description of the set of FIR filters synthesis experiments.

FIR	Input/coeff (bits)	N. taps	Number of bits	Moduli set	N. of moduli
FIR1	8	16	20	{64, 31, 29, 23}	4
FIR2	8	32	21	{128, 31, 29, 23}	4
FIR3	8	64	22	{256, 31, 29, 23}	4
FIR4	8	128	23	{512, 31, 29, 23}	4
FIR5	8	256	24	{64, 31, 29, 23, 19}	5
FIR6	12	16	28	{64, 31, 29, 23, 19, 17}	6
FIR7	12	32	29	{128, 31, 29, 23, 19, 17}	6
FIR8	12	64	30	{256, 31, 29, 23, 19, 17}	6
FIR9	12	128	31	{512, 31, 29, 23, 19, 17}	6
FIR10	12	256	32	{64, 31, 29, 23, 19, 17, 13}	7

Table 5 Resource usage and speed for the experiments.

FIR	Number of bits (MHz)	Max. freq. (#LUTs)	Taps (#LUTs)	In converter (#LUTs)	Out converter (#LUTs)	Total resources
FIR1	20	435	592	30	182	804
FIR2	21	435	1,248	30	182	1,460
FIR3	22	435	2,688	30	182	2,900
FIR4	23	435	6,144	30	182	6,356
FIR5	24	435	12,032	40	224	12,296
FIR6	28	300	912	70	270	1,252
FIR7	29	300	1,888	70	270	2,228
FIR8	30	300	3,968	70	270	4,308
FIR9	31	300	8,704	70	270	9,044
FIR10	32	303	17,152	84	309	17,545

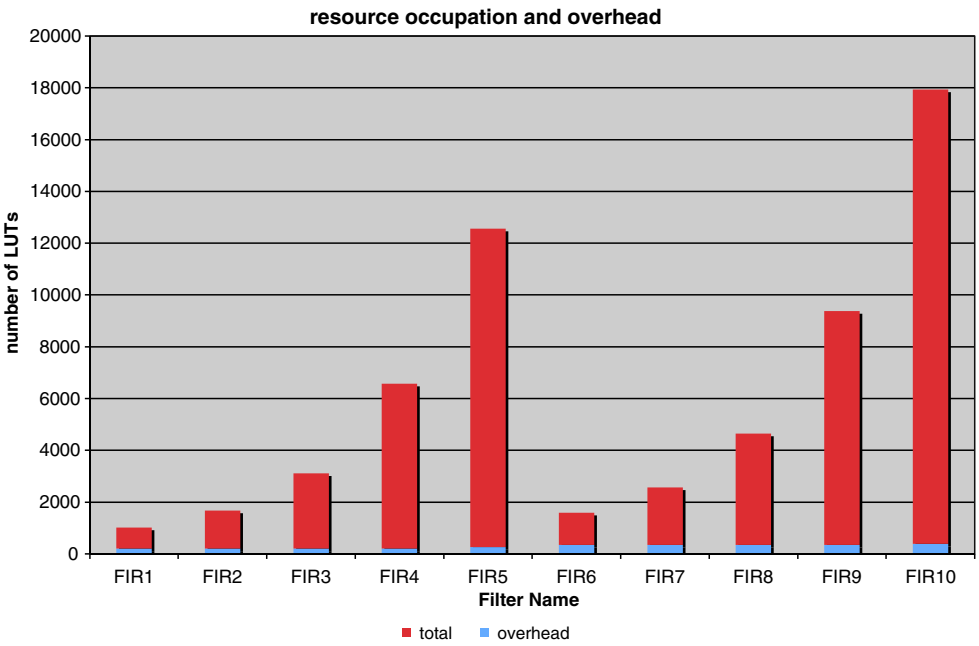

Figure 12 Converters overhead.

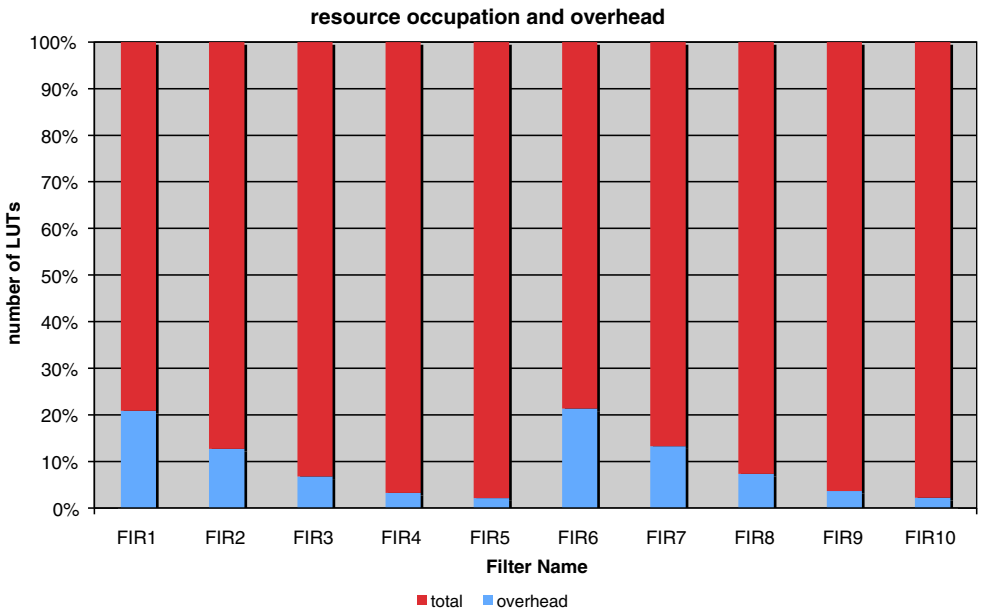

Figure 13 Converters overhead in percentage.

and consequently the change of representation does not represent a drawback in this kind of architectures.

Finally, we compare the obtained results with those obtained by implementing the same experiments using a TCS representation. In these comparison no truncation is performed in the FIR filter. As indicated in Section 2 usually truncation is used to limit the resources in TCS filters but it has been shown in the literature [18] that truncation does not offset the advantages of a RNS implementation in real applications where N (the number of taps) is usually high. Moreover, the RNS representation is often used to design filters with error detection and correction capabilities [15–17]. In those cases, the use of truncation is not allowed. The results of the comparison in terms of resource occupation are shown in Table 6.

Table 6 Resource Comparison of RNS and TCS filters.

FIR	TCS (#LUTs)	RNS (#LUTs)	Saving (%)
FIR1	788	804	−2
FIR2	1,800	1,460	18
FIR3	3,632	2,900	20
FIR4	6,966	6,356	8
FIR5	15,203	12,296	19
FIR6	1,899	1,252	34
FIR7	3,338	2,228	33
FIR8	6,555	4,308	34
FIR9	14,043	9,044	35
FIR10	29,234	17,545	40

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 the case of 8 bits the advantage depends on the filter length. 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 show that the presented techniques give interesting advantages for FIR filters characterized by high dynamic range and high number of taps.

5 Conclusion

An optimization of Residue Number System (RNS) arithmetic to better exploit some of the architectural characteristic of the last generation FPGAs has been presented. The basic arithmetic operations of RNS (addition and multiplication) has been implemented by using a ROM based approach. The paper shows that this ROM based implementation achieve performances better than the classical MUX based adders usually proposed for ASIC implementations. In particular the presented results show that the best performance are obtained when 5 bit moduli are used. Using this approach based on ROM modular adders and multipliers, different optimization techniques for the basic blocks of RNS filters (modulo m_i filters, forward and reverse converters) has been presented and discussed. The

proposed optimization techniques has been applied with a wide set of RNS filters showing that high speed, low resource occupation RNS filters can be obtained by using our techniques.

References

1. Allan, A., Edenfeld, D., Joyner, W., Kahng, A., Rodgers, M., & Zorian, Y. (2002). International technology roadmap for semiconductors, *IEEE Computer, 35*(1), 42–53.
2. Cosoroaba, A., & Rivoallon, F. (2006). *Achieving higher system performance with the Virtex-5 family of FPGAs, Xilinx WP245* (Vol. 1).
3. Re, M., Nannarelli, A., Cardarilli, G. C., & Lojacono, R. (2001). FPGA Realization of RNS to binary signed conversion architecture. In *2001 IEEE international symposium on circuits and systems, Sydney, Australia, 6–9 May 2001*.
4. Ramírez, J., Meyer-Bäse, U., Taylor, F. J., García, A., & Lloris-Ruíz, A. (2003). Design and implementation of high-performance RNS wavelet processors using custom IC technologies. *The Journal of VLSI Signal Processing, 34*(3), 227–237.
5. Ciet, M., Neve, M., Peeters, E., & Quisquater, J. J. (2003). Parallel fpga implementation of RSA with residue number systems—can side-channel threats be avoided? In: *Proc. 46th IEEE international midwest symposium on circuits and systems MWSCAS 03, 27–30 Dec 2003* (Vol. 2, pp. 806–810).
6. Kaluri, K., Leong, W. F., Tan, K.-H., Johnson, L., & Soderstrand, M. (2001). FPGA hardware implementation of an RNS FIR digital filter. In *Thirty-fifth asilomar conference on signals, systems and computers* (Vol. 2, pp. 1340–1344).
7. Vinogradov, I. (1955). *An introduction to the theory of numbers*. New York: Pergamon Press.
8. Szabo, N., & Tanaka, R. (1967). *Residue arithmetic and its applications in computer technology*. New York: McGraw-Hill.
9. Sodestrand, M., Jenkins, W., Jullien, G. A., & Taylor, F. J. (1986). *Residue number system arithmetic: Modern Applications in digital signal processing*. New York: IEEE Press.
10. Vu, T. V. (1985). Efficient implementation of the chinese remainder theorem for sign detection and residue decoding. *IEEE Transactions on Circuits Systems-I, 45*, 667–669.
11. Piestrak, S. (1995). A high-speed realization of a residue to binary number system converter. *IEEE Transactions on Circuits Systems-II Analog and Digital Signal Processing, 42*, 661–663.
12. Cardarilli, G., Re, M., & Lojacono, R. (1997). A residue to binary conversion algorithm for signed numbers. In: *European conference on circuit theory and design (ECCTD97)* (Vol. 3, pp. 1456–1459).
13. Logic array blocks and adaptive logic modules in Stratix III devices chapter in volume 1 of the StratixIII device handbook.
14. Mitra, S. K., & Kaiser, J. F. (1993). *Handbook for digital signal processing*. Wiley-Interscience.
15. Bandyopadhyay, S., Jullien, G. A., & Sengupta, A. (1988). A systolic array for fault tolerant digital signal processing using a residue number system approach. In: *Proceedings of the international conference on systolic arrays, 25–27 May 1988* (pp. 577–586).
16. Etzel, M. H., & Jenkins, W. K. (1980). Redundant residue number systems for error detection and correction in digital filters. *IEEE Transactions on Acoustics, Speech and Signal Processing, ASS-28*(5), 538–544.
17. Pontarelli, S., Cardarilli, G. C., Re, M., & Salsano, A. (2008). Totally fault tolerant RNS based FIR filters. In *IEEE international on-line testing symposium*.
18. Nannarelli, A., Re, M., & Cardarilli, G. C. (2001). Tradeoffs between residue number system and traditional FIR filters. In: *IEEE international symposium on circuits and systems, ISCAS 2001, Sydney, Australia, 6–9 May 2001* (Vol. II, pp. 305–308).

Salvatore Pontarelli received the Laurea degree in Electronic Engineering from the University of Bologna in 1999 and the Ph.D. in Microelectronics and Telecommunications Engineering from the University of Rome "Tor Vergata" in 2003. Currently he is have a post-doctoral fellowship with the Department of Electronic Engineering of the University of Rome "Tor Vergata". His research mainly focuses on fault tolerance, on-line testing and reconfigurable digital architectures.

Gian Carlo Cardarilli received the Laurea (summa cum laude) in 1981 from the University of Rome "La Sapienza". He works for the University of Rome "Tor Vergata" since 1984. At present he is full professor of Digital Electronics and Electronics for Communication Systems at the University of Rome "Tor Vergata". During the years 1992/1994 he worked for the University of L'Aquila. During the years 1987/1988 he worked for the Circuits and Systems team at EPFL of Lausanne (Switzerland). Professor

Cardarilli interests are in the area of VLSI architectures for Signal Processing and IC design. In this field he published over 140 papers in international journals and conferences. Scientific interests of Professor Cardarilli concern the design of special architectures for signal processing. In particular, he works in the field of computer arithmetic and its application to the design of fast signal digital processor. He also developed mixed-signal neural network architectures implementing them in silicon technology. Recently, he also proposed different new solutions for the implementation of fault-tolerant architectures.

Marco Re received the Laurea degree in Electronic Engineering from the University of Rome "La Sapienza" in 1991 and the Ph.D. in Microelectronics and Telecommunications Engineering from the University of Rome "Tor Vergata" in 1996. In 1998 he joined the Department of Electronic Engineering of the University of Rome "Tor Vergata" as Researcher. He was awarded two one-year NATO fellowships with the University of California at Berkeley in 1997 and 1998. His main interests and activities are in the area of DSP algorithms, fast DSP architectures, Fuzzy Logic hardware architectures, Hardware-Software Codesign, Number Theory with particular emphasis on Residue Number System, Computer Arithmetic and Cad tools for DSP, Fault Tolerant and Self Checking circuits. He has authored and coauthored more than eighty papers.

Adelio Salsano was born in Rome on December 26, 1941 and is currently full professor of Microelectronics at the University of Rome, "Tor Vergata" where he teaches the courses of Microelectronics and Electronic Programmable Systems. His present research work focuses on the techniques for the design of VLSI circuits, considering both the CAD problems and the architectures for ASIC design. In particular, of relevant interest are the research activities on fault tolerant/fail safe systems for critical environments as space, automotive etc.; on low power systems considering the circuit and architectural points of view; and on fuzzy and neural systems for pattern recognition. An international patent and more than 90 papers on international journals or presented in international meetings are the results of his research activity. At present he is the President of a national consortium named U.L.I.S.S.E., between ten universities, three polytechnics and several of the biggest national industries, as STMicroelectronics, ESAOTE, FINMECCANICA. He is responsible for contracts with the ASI, Italian Space Agency, for the evaluation and use in space environment of COTS circuits and for the definition of new suitable architectures for space applications. Professor Salsano is also involved in professional activities in the field of information technology and is also consultant of many public authorities for specific problems. In particular he is consultant of the Departments of the Research and of the Industry, of IMI and of other authorities for the evaluation of industrial public and private research projects. Professor Salsano was a member of the consulting Committee for Engineering Sciences of the CNR (National Research Council) from 1981 to 1994.