Residue Number Systems: a Survey

Nannarelli, Alberto; Re, Marco

Publication date:
2008

Document Version
Publisher's PDF, also known as Version of record

IMM-Technical Report-2008-04

Residue Number Systems: a Survey

Alberto Nannarelli[(1)], Marco Re[(2)]

[(1)]DTU Informatics
Technical University of Denmark
Kongens Lyngby, Denmark
Email: an@imm.dtu.dk

[(2)]Department of Electronic Engineering
University of Rome Tor Vergata,
Rome, Italy
Email: marco.re@uniroma2.it

May 30, 2008

Contents

List of Figures iii
List of Tables v
List of Abbreviations vi

1 Introduction **1**
1.1 History of Residue Number System 2
1.2 Basic Theory of Residue Number System 3
1.2.1 The Isomorphism Technique 4
1.3 Input/Output Conversion 6
1.3.1 Input Conversion 6
1.3.2 Output Conversion 7
1.4 Moduli Selection 8
1.5 Scaling 9
1.6 Quadratic Residue Number System 10
1.7 RNS and Fault-Tolerance 11

2 Implementation Aspects **13**
2.1 Modular Addition 13
2.2 Modular Multiplication 14
2.3 Reduction to Modulus m 16
2.4 Input/Output Conversion 17
2.4.1 Input Conversion 17
2.4.2 Output Conversion 22

3 Case Study: FIR Filters **29**
3.1 RNS Implementation of FIR Filters 32
3.2 FIR Filters in Transposed Form 32
3.2.1 Transposed FIR Filters in TCS 32

3.2.2 Transposed FIR Filters in RNS . . . 34
3.2.3 Transposed Truncated FIR Filters in TCS . . . 36
3.2.4 Transposed FIR Filters: Summary . . . 36
3.3 FIR Filters in Direct Form . . . 39
3.3.1 Direct FIR Filters in TCS . . . 39
3.3.2 Direct FIR Filters in RNS . . . 40
3.3.3 Direct FIR Filters: Summary . . . 41
3.4 RNS Coding Overhead . . . 41
3.4.1 Coding Overhead . . . 43
3.4.2 Design Space Exploration . . . 44
3.5 Carry-Save RNS Filter . . . 48
3.6 Low Power RNS Implementation . . . 50
3.6.1 Multi-Voltage RNS Filter . . . 50
3.6.2 Low Leakage Dual Threshold Voltage RNS Filter . . . 51
3.6.3 Results of Implementations . . . 54
3.7 Complex FIR Filters . . . 56
3.7.1 Comparison TCS vs. QRNS . . . 58
3.8 FPGA Implementation . . . 60
3.8.1 The Experimental Set-Up . . . 60
3.8.2 Results . . . 60
3.9 ASIC vs. FPGA Implementations . . . 64
3.9.1 ASIC-SC: Power Consumption Contributions . . . 65
3.9.2 Analysis of Experimental Results: ASIC-SC . . . 69
3.9.3 FPGA: Power Consumption Contributions . . . 70
3.9.4 Analysis of Experimental Results: FPGA . . . 72

4 Tools **73**
4.1 Tool Description . . . 73
4.2 Architecture Chooser . . . 75
4.3 Characterization . . . 75
4.4 Examples and Results . . . 76

5 Conclusions and Future Work **77**
5.1 DSP Functions of Interest and Applications . . . 77
5.2 Proposal for Further Developments . . . 79

Acknowledgments **81**

Bibliography **81**

List of Figures

1.1 Example of $\mathcal{N} \rightarrow$ RNS mapping. 4
1.2 The isomorphism table for $m = 11$. 5
1.3 The isomorphism multiplication rule. 5
1.4 Input converter generic architecture. 6

2.1 Architecture of the modular adder. 14
2.2 Structure of isomorphic multiplication. 15
2.3 Modulo reduction block architecture 17
2.4 Hardware implementation of the input converter for $N = 7$ and $m = 17$. 21
2.5 Output converter architecture. 27
2.6 Implementation of output converter. 28

3.1 FIR filters in transposed (top) and direct (bottom) form. 30
3.2 Truncation schemes. 31
3.3 Architecture of RNS FIR filters. 32
3.4 TCS FIR filter in transposed form. 33
3.5 Tap structure for the transposed TCS FIR filter. 33
3.6 RNS FIR filter in transposed form. 35
3.7 Trends for transposed FIR filters: area (top) and power dissipation (bottom). 38
3.8 TCS FIR filter in direct form. 39
3.9 RNS FIR filter in direct form. 41
3.10 Trends for direct FIR filters: delay critical path (top) and area (bottom). 42
3.11 Results of EXP-1: direct and transposed TCS vs. RNS 46
3.12 Results of EXP-2: transposed TCS vs. RNS 47
3.13 Results of EXP-3: complete filter TCS vs. RNS 48

3.14 Structure of FIR filter in transposed form and its critical path. . . 48
3.15 Tap structure for RNS carry-save. 49
3.16 Relay station. 49
3.17 The HS and LL cells mix depends on the synthesis timing constraint. 53
3.18 Summary of results for low power filters. 55
3.19 Dynamic (top) and static (bottom) power dissipation (TCS vs. RNS). 57
3.20 Structure of QRNS filter. 58
3.21 Structure of tap in complex TCS FIR filter. 59
3.22 FPGA Measurement set-up. 61
3.23 Plots of E_{TCS}, E_{RNS} and E_{CSRNS}. 62
3.24 Photo of the test bed. 63
3.25 Comparisons of TCS and RNS implementations of FIR filters (ASIC-SC) 66
3.26 ASIC-SC power contributions 67
3.27 Contributions to the node capacitance. 68
3.28 Increasing of node capacitances with complexity 69
3.29 Power consumption breakdown for FPGA implementation 71

4.1 Tool interface for the designer. 74
4.2 Structure of the tool. 75

List of Tables

1.1 Example of QRNS multiplication mod 13. 11

2.1 Example of isomorphic transformation for $m = 5$ ($q = 2$). 15
2.2 Input converter comparison. 20

3.1 FIR filters in transposed form: summary of results. 37
3.2 FIR filters in direct form: summary of results. 43
3.3 Moduli set and dynamic range for EXP-1. 45
3.4 Moduli set and dynamic range for EXP-2. 47
3.5 Delay, area and power dissipation per tap in RNS FIR. 50
3.6 Power dissipation for multiple supply voltage in tap. 51
3.7 Impact of leakage on technology scaling. 52
3.8 Filter dual-V_t implementations: results. 56
3.9 Complex filters: summary of results. 59
3.10 Measurements of average power consumption and E_c. 62
3.11 Expressions of E_c for the different filters. 63
3.12 Area, power and globality ratios for the ASIC-SC implementation . 69
3.13 Interconnects in FPGA. 70
3.14 Area, power and globality ratios for FPGA implementation. 72

4.1 Area estimation for generated filters. 76

List of Abbreviations

AC	Architecture Choser
ASIC-SC	Application Specific Integrated Circuit-Standard Cell
CPP	Cyclic Convolution Property
CRC	Cyclic Convolution Codes
CRT	Chinese Remainder Theorem
CS	Carry-Save
DFT	Discrete Fourier Transform
DIT	Direct Isomorphic Transformation
DSE	Design Space Exploration
DSP	Digital Signal Processing or Processor
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
GF	Galois Field
GI	Globality Index
HS	High Speed
IFFT	Inverse Fast Fourier Transform
IIT	Inverse Isomorphic Transformation
INTT	Inverse Number Theoretic Transform
LL	Low Leakage
LSB	Least Significant Bit
LUT	Look-Up-Table
MRC	Mixed Radix Conversion
MRS	Mixed Radix System (Number System)
MSB	Most Significant Bit
MV-RNS	Multi Voltage RNS
NTT	Number Theoretic Transform
OH	Coding Overhead
PFA	Prime Factor Algorithm
QRNS	Quadratic Residue Number System
RNS	Residue Number System
RNS-PC	Residue Number System Product Code
RRNS	Redundant Residue Number System

RS	Reed-Solomon
RTL	Register Transfer Level
TCS	Two's Complement System
VB	VHDL Builder
VHDL	VHSIC Hardware Description Language

Chapter 1

Introduction

The use of alternative number systems in the implementation of application specific Digital Signal Processing (DSP) systems has gained a remarkable importance in recent years. In particular, the renewed success of the Residue Number System (RNS) is mainly related to the request of low power consumption and high speed, but it is also related to the availability of hardware platforms, such as FPGAs, particularly suitable for the implementation of RNS arithmetic blocks. In fact, the RNS operations are often based on Look-Up Tables (LUTs) which are the basic building blocks in modern FPGAs. The main purpose of this report is

- introduction of the basic theoretical aspects of the Residue Number System (RNS)
- discussion of the DSP operations that can take advantage from the RNS arithmetic
- discussion of the critical operations
- description of the architecture of the basic building blocks used in DSP systems
- description of a "killer application" such as FIR filtering in order to prove the effectiveness of RNS in terms of area, speed and power consumption
- CAD tool that can free the designer from the RNS non conventional mathematical aspects
- overview of the state of the art in terms of patents and industrial applications
- motivations for future work.

1.1 History of Residue Number System

The basic modular arithmetic formal theory has been written by important mathematicians such as for example Fermat, Eulero and Gauss, during the sixteenth, seventeenth and nineteenth centuries. The introduction of the use of modular arithmetic in applied sciences and in particular in electronic engineering happened during the fifties together with the explosion of the computer technology. In particular, during the fifties the experimentation started both, in digital signal processing and computer arithmetic fields. In 1955 and the following years, the first computer based on residue arithmetic principles was built by Svoboda and Valach [1], [2],[3], in Czechoslovakia, by using the vacuum tube technology of the day. At the end of the fifties and in the first years of the sixties, at the Harvard Computation Laboratory [4] at the University of Harvard, at RCA [5], at Lockheed Missiles and Space Company [6] and at Westinghouse [7], important research started on the subject of modular arithmetic and on the Residue Number System. In those years, the main interest was focused on the design and implementation of On Board Processors (OBP) for space, avionic and military applications characterized by high speed and very high reliability. In fact, the RNS representation is a non positional number representation that is intrinsically parallel and consequently for some important arithmetic operations (such as for example addition and multiplication) it corresponds to very fast architectures.

Moreover, its arithmetic structure characterized by lack of communication between the modular processors suggests its use in high reliability systems. In the subsequent years before the explosion of the Digital Signal Processing (DSP) and Very Large Scale Integration (VLSI) era, the use of RNS based electronic systems was relegated to very special applications. After that, these two new fields pushed for rediscovering RNS. In fact the new DSP algorithmic techniques were based on basic arithmetic operators (e.g. linear combinations of integer numbers). Moreover, with the advent of VLSI the implementation of RNS operators were facilitated (a very efficient implementation of RNS is obtained by using LUTs). Two other important points that are the key of the renewed interest for RNS are

1. Low power consumption
2. High reliable systems (Fault Tolerance)

The importance of power consumption is related to two important field of applications

- Low power for portable multimedia applications and embedded computing
- Low power to permit the use of new devices such as for example Field Programmable Gate Arrays (FPGAs) in power critical applications. In fact, it is well known, that the flexibility and computational power of the modern FPGAs comes at the cost of very high power consumption.

The importance of high reliability in the design of complex digital systems also in the case of non special applications (such as space or military) is related to the extensive use of digital electronics in transportation, but more importantly, it is related to the scaling of the electronic technologies. It is well known, in fact, the effect of high energy particles and radiations (now these effects are important also at the sea level) for the modern nano-technologies that are used in the most advanced CMOS processes [8], [9]. For this reason, the use of a number representation that intrinsically enables low power, parallelism (speed) and fault tolerance (reliability) becomes a useful design choice for the IC designer.

1.2 Basic Theory of Residue Number System

A Residue Number System is defined by a set of relatively prime integers

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

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

$$M = m_1 \cdot m_2 \cdot \ldots \cdot m_P .$$

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

$$X \overset{RNS}{\rightarrow} (\langle X \rangle_{m_1}, \langle X \rangle_{m_2}, \ldots, \langle X \rangle_{m_P})$$

where

$$\langle X \rangle_{m_i} = X \bmod m_i \qquad \left(\text{ e.g. } \frac{X}{m_i} = q_i + \langle X \rangle_{m_i} \right)$$

and q_i is the integer quotient of $\frac{X}{m_i}$, and $\langle X \rangle_{m_i}$ is its remainder ($0 \leq \langle X \rangle_{m_i} < m_i$). A comprehensive description of the RNS theory and its application to computer systems can be found in [10], [11], [12] and [13].

Figure 1.1 illustrates an example of RNS, with base { 5, 7, 8 } and dynamic range $M = 5 \cdot 7 \cdot 8 = 280$, and how a positive integer from the set $[0, M-1]$ is mapped into the RNS.

When signed integers are mapped in RNS two cases arises:

1. M **odd:** in this case the range of the RNS representation is

$$-\frac{M-1}{2} \leq X \leq \frac{M-1}{2}$$

2. M **even:** in this case the range of the RNS representation is

$$-\frac{M}{2} \leq X \leq \frac{M}{2} - 1$$

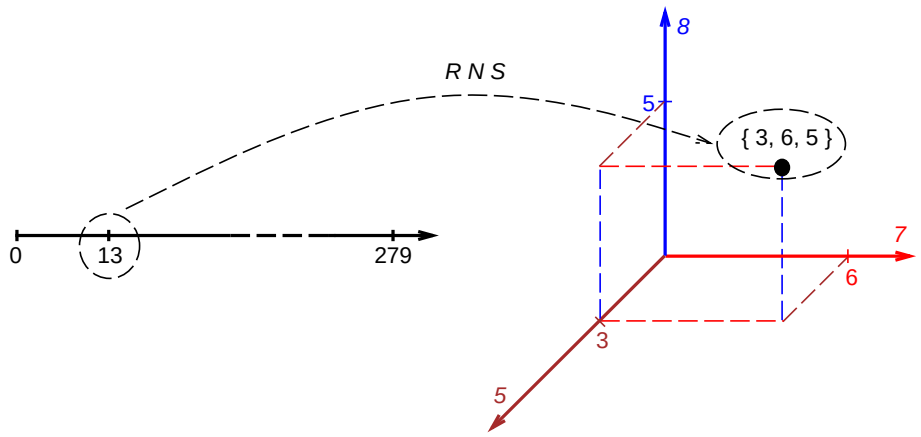


Figure 1.1: Example of $\mathcal{N} \rightarrow$ RNS mapping.

The computation of the residues in the case of negative numbers is obtained by complementing the residues i.e.

$$\langle X \rangle_{m_i} = \begin{cases} \langle X \rangle_{m_i} & \text{if } X \geq 0 \\ \langle m_i - \langle |X| \rangle_{m_i} \rangle_{m_i} & \text{if } X < 0 \end{cases}$$

Operations such as addition and multiplication, are done in parallel on the different RNS moduli

$$Z = X \ op \ Y \quad \overset{RNS}{\rightarrow} \quad \begin{cases} Z_{m_1} = \langle X_{m_1} \ op \ Y_{m_1} \rangle_{m_1} \\ Z_{m_2} = \langle X_{m_2} \ op \ Y_{m_2} \rangle_{m_2} \\ \dots \qquad \dots \qquad \dots \\ Z_{m_P} = \langle X_{m_P} \ op \ Y_{m_P} \rangle_{m_P} \end{cases} \tag{1.1}$$

and the dynamic range of each of the RNS processor channels is limited to $m_i - 1$.

1.2.1 The Isomorphism Technique

A special technique, based on isomorphic transformations [10], can be used in RNS to transform the modular multiplication into a simpler modular addition. It is based on the concept of indices that are similar to logarithms, and primitive roots r which are similar to logarithm bases. It is possible to demonstrate that if the number m is prime there exists a number of primitive radices (the number of the primitive radices can be computed by using the Euler's function) that share the following property: every element of the field $F(m) = \{0, 1, \dots, m-1\}$ excluding the zero element can be generated by using the following equation

$$F(m) = \left\langle r^k \right\rangle_m \tag{1.2}$$

F(11)	r=2	r=6	r=7	r=8
$\langle r^k \rangle_{11}$	k	k	k	k
1	0	0	0	0
2	1	9	3	7
3	8	2	4	6
4	2	8	6	4
5	4	6	2	8
6	9	1	7	3
7	7	3	1	9
8	3	7	9	1
9	6	4	8	2
10	5	5	5	5

Figure 1.2: The isomorphism table for $m = 11$.

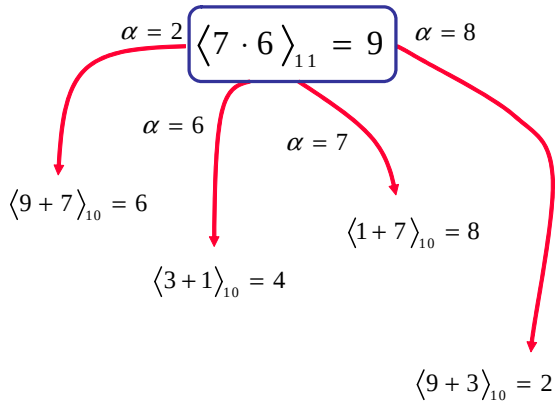


Figure 1.3: The isomorphism multiplication rule.

where k is an integer number. In Figure 1.2 the generation of the elements for $F(11)$ is shown. In this case the four primitive radices $\{2,\ 6,\ 7,\ 8\}$ can be utilized. The product of two elements a and b belonging to the field is implemented by

1. Forward transformation of a and b in the corresponding indices
2. Addition modulo $m - 1$ of the two indices
3. Reverse conversion of the result of the addition to obtain the final result of the modular product

An example of the multiplication algorithm for the different primitive radices in the case $m = 11$ is illustrated in Figure 1.3.

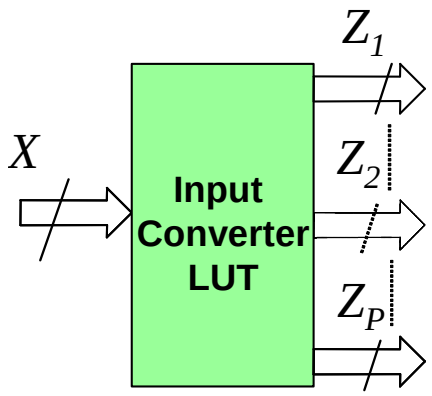


Figure 1.4: Input converter generic architecture.

1.3 Input/Output Conversion

In this Section, the basic methods for the input and output conversion in RNS systems are illustrated. A lot of work has been presented in the literature on these subjects [14]. Clearly, the conversions from $\mathcal{N}$ to RNS, and vice-versa, constitute a significant overhead in systems implemented in RNS. However, efficient methods to perform those conversions are presented in [15] [16], [17], and [18]. Anyhow, when in the RNS processor the number of operations to process each input sample increases the input and output conversion overhead becomes less important.

1.3.1 Input Conversion

The conversion of the input data in the RNS format is obtained by computing the integer remainder of the division of Z by m_i i.e. the quantity

$$Z_i = \langle Z \rangle_{m_i} \tag{1.3}$$

there are many works presented in the literature on this subject. They can be categorized depending on the class of used moduli i.e. special moduli-sets and general moduli-sets. As stated in Section 1.4, the best approach is the use of general moduli-sets. In this case, the use of LUTs is the most obvious possibility and the architecture of the converter is shown in Figure 1.4. If n is the number of input bits, the LUT size in terms of number of cells is 2^n and the number of bits that are needed for the residue binary coding is

$$b = \sum_{i=1}^{P} \lceil log_2 m_i \rceil \tag{1.4}$$

This approach can be efficiently used when the number of bits of the data to be

converted in RNS is not large. A careful choice of the moduli set helps in containing the memory resources by minimizing the coding overhead (Section 3.4.1).

In the case in which n is large, and consequently, the amount of memory becomes too large, the techniques that can be used are based on the observation that finding a residue of a number with respect to a given modulus is basically obtained by computing modular sums of different combinations of modular powers of two (as shown in Section 2.4.1). Usually these techniques are fast (systolic) and are useful if some latency can be tolerated.

1.3.2 Output Conversion

The conversion from the RNS representation of the set Z_i that represents the data encoded in RNS, can be accomplished by two different techniques the Mixed Radix Conversion (MRC) and the Chinese Remainder Theorem (CRT) [11]. The most used technique for high speed implementations is the CRT that is based on the following reconstruction formula

$$Z = \left\langle \sum_{i=1}^{P} \overline{m_i} \cdot \langle \overline{m_i}^{-1} \rangle_{m_i} \cdot Z_i \right\rangle_M = \langle H \rangle_M \tag{1.5}$$

with $\overline{m_i} = \frac{M}{m_i}$ and $\overline{m_i}^{-1}$ obtained by $\langle \overline{m_i} \cdot \overline{m_i}^{-1} \rangle_{m_i} = 1$.

To better explain the CRT, we perform the conversion RNS $\rightarrow \mathcal{N}$ for the example of Figure 1.1. The RNS base is $\{ 5, 7, 8 \}$ and its dynamic range is $M = 280$. We start by computing the values $\overline{m_i} = \frac{M}{m_i}$

$$\overline{m_1} = \frac{280}{5} = 56 \qquad \overline{m_2} = \frac{280}{7} = 40 \qquad \overline{m_3} = \frac{280}{8} = 35$$

To compute $\overline{m_i}^{-1}$, we have to find a number x such that

$$\langle \overline{m_i} \cdot x \rangle_{m_i} = 1 \tag{1.6}$$

For this reason, x is called the multiplicative inverse of $\overline{m_i}$ and indicated as $\overline{m_i}^{-1}$. By computer iterations, we find

$$\overline{m_1}^{-1} = 1 \qquad \overline{m_2}^{-1} = 3 \qquad \overline{m_3}^{-1} = 3$$

Finally, applying (1.5) to the set of residues $\{3, 6, 5\}$ we get

$$\left\langle \sum_{i=1}^{3} \overline{m_i} \cdot \langle \overline{m_i}^{-1} \rangle_{m_i} \cdot Z_i \right\rangle_{280} = \langle\ 56 \cdot 1 \cdot 3 + 40 \cdot 3 \cdot 6 + 35 \cdot 3 \cdot 5\ \rangle_{280} =$$

$$\langle\ 1413\ \rangle_{280} = 13$$

If positive and negative integers have been used in the RNS system the ouput conversion must be slightly modified to convert the residues in a binary negative and positive range. In particular due to the algorithm used for mapping the relative integers (section 1.2) the following rule is used

- H belongs to the range of the signed representation: in this case the correct value is $\langle H \rangle_M$.
- H does not belong to the range of the signed representation: in this case the result is $\langle H \rangle_M - M$.

For example, given the moduli set $\{13,\ 15,\ 16\} \rightarrow M = 3120$ the representation range is $X \in [-1560,\ 1559]$.
The conversion of the negative value $X = -9 \overset{RNS}{\rightarrow} \{4,\ 6,\ 7\}$ is obtained by using the CRT formula where $\overline{m_i} = \{240,\ 208,\ 195\}$ and $\overline{m_i}^{-1} = \{11,\ 7,\ 11\}$ obtaining $\langle 3111 \rangle_{3120} = 3111$ that is out of the representation range. In this case the correct result is $X = 3111 - 3120 = -9$. Vice versa, in the case of the conversione of the value $X = 9$, we have $X = 9 \overset{RNS}{\rightarrow} \{9,\ 9,\ 9\}$. The recontructed value is $\langle 3129 \rangle_{3120} = 9$.

1.4 Moduli Selection

The choice of the moduli in a RNS based DSP system is of particular importance because of its impact on complexity, power consumption and speed. Moreover it is not easy to find a simple methodology for their selection due to conflicting conditions. For example in order to maximize the parallelism, the maximum number of moduli to obtain the required dynamic range should be used, but this choice is in conflict with the implementation complexity of some important operations such as for example Mixed Radix Conversion. In the following are indicated and briefly illustrated a number of important criteria

1. **Necessary condition:** The moduli in the chosen set must be coprime.
2. **Dynamic range:** The moduli selected must cover the dynamic range of the application.
3. **Isomorphism technique use:** in order to easily implement multiplications by using the isomorphism technique the selected moduli must be prime. These moduli are characterized by the existence of the primitive radices that are the base of the isomorphism technique.
4. **Balance:** The differences between the moduli should be as small as possible. By this way the hardware complexity and consequently the speed performance of each of the channel of the modular processor are similar.

5. **Low coding overhead:** Due to the fact that just one module can be a power of two and that the other modules are not powers of two the moduli selection should correspond to the minimum coding overhead.

6. **Simplification of the arithmetic operations:** The moduli can be selected in order to simplify arithmetic operations. For example moduli that are near to powers of two (such as for example $m_i = \{2^k - 1,\ 2^k,\ 2^k + 1\}$) are useful to simplify modular addition and multiplication.

7. **Size of individual moduli:** in order to better exploit the RNS characteristic, in terms of parallelism and consequently speed, the individual moduli should not be too large.

8. **Parallelism:** the use of a high cardinality set of moduli permits to exploit the parallelism of the RNS representation.

9. **Selection of the moduli to have unity multiplicative inverses[1]:** in order to simplify certain RNS operations such as for example, MRC and CRT, a moduli selection characterized by the maximum number of unity multiplicative inverses is an advantage.

1.5 Scaling

As stated above the RNS is very interesting for the implementation of some operations, perhaps commonly used in DSP but, on the other hand, several basic operations such as sign detection and truncation, which are trivial in two's complement, are not easily implemented in RNS. In DSP, the most common operation affected by this problem is the scaling operation (i.e. the division by a constant factor followed by rounding or truncation) used for reducing the dynamic range in DSP units. In large systems, dynamic range reduction might be needed in different parts of the datapath requiring several scaling blocks. For these reasons, the investigation of optimized RNS scaling techniques is an important aspect of the RNS implementation of DSP systems. The scaling techniques are mainly based on division remainder zero and base extension and have been presented in books such as, [19], [11]. Some specialized techniques have been presented in the literature. These techniques can be classified in three main groups:

1. Scaling by one or several moduli of the RNS base: [20], [21], [22], [23], [24], [25].

2. Scaling by an integer belonging to the RNS dynamic range [26].

3. Scaling by a power of two [27].

[1] In the CRT $\overline{m_i}$ has unity multiplicative inverse when in (1.6) $x = 1$.

In the first group, different algorithms are used to reduce the dynamic range by a scaling factor that is the product of some moduli of the RNS base. In the second group, a factorization of the CRT (see 1.5) is exploited to scale by an integer number S in the dynamic range of the RNS representation. Differently from the algorithms at points 1. and 2., the last technique does not require a binary or a Mixed Radix System (MRS) conversion since the scaling operation is completed in the RNS domain. Moreover, scaling by a power of two is the classical way for dynamic reduction in binary systems and it is possible to obtain a programmable scaler with a slightly increase in the hardware complexity.

1.6 Quadratic Residue Number System

In the case of the representation of complex numbers, we can transform the imaginary term into an integer if the equation $q^2 + 1 = 0$ has two distinct roots q_1 and q_2 in the ring of integers modulo m_i (Z_{m_i}). A complex number $x_R + jx_I = (x_R, x_I) \in Z_{m_i}$, with q root of $q^2 + 1 = 0$ in Z_{m_i} has a unique Quadratic Residue Number System (QRNS) [12] representation given by

$$\begin{array}{ll} (x_R, x_I) \overset{QRNS}{\rightarrow} & (X_i, \hat{X}_i) \quad i = 0, 1, \ldots, P \\ & X_i = \langle x_R + q \cdot x_I \rangle_{m_i} \\ & \hat{X}_i = \langle x_R - q \cdot x_I \rangle_{m_i} \end{array}$$

The inverse QRNS transformation is given by

$$\begin{aligned} x_R &= \langle 2^{-1}(X_i + \hat{X}_i) \rangle_{m_i} \\ x_I &= \langle 2^{-1} \cdot q^{-1}(X_i - \hat{X}_i) \rangle_{m_i} \end{aligned}$$

where 2^{-1} and q^{-1} are the multiplicative inverses of 2 and q, respectively, modulo m_i:

$$\langle 2 \cdot 2^{-1} \rangle_{m_i} = 1 \qquad \text{and} \qquad \langle q \cdot q^{-1} \rangle_{m_i} = 1 \ .$$

Moreover, it can be proved that for all the prime integers which satisfy

$$p = 4k + 1 \qquad\qquad k \in N$$

the equation $q^2 + 1 = 0$ has two distinct roots q_1 and q_2.

As a consequence, the product of two complex numbers $x_R + jx_I$ and $y_R + jy_I$ is in QRNS

$$(x_R + jx_I)(y_R + jy_I) \overset{QRNS}{\rightarrow} (\langle X_i Y_i \rangle_{m_i}, \langle \hat{X}_i \hat{Y}_i \rangle_{m_i}) \tag{1.7}$$

and it is realized by using two integers multiplications instead of four. Table 1.1 shows an example of QRNS multiplication in the ring modulo 13.

example for $m = 13$:	
$q = q_1 = 5 \leftrightarrow \langle 5 \cdot 5 \rangle_{13} = -1$	
$(x_R + jx_I)(y_R + jy_I) = (3 + j)(2 + j2) = 4 + j8$	
conversion to QRNS	
$X = \langle 3 + 5 \cdot 1 \rangle_{13} = 8$	$Y = \langle 2 + 5 \cdot 2 \rangle_{13} = 12$
$\hat{X} = \langle 3 - 5 \cdot 1 \rangle_{13} = 11$	$\hat{Y} = \langle 2 - 5 \cdot 2 \rangle_{13} = 5$
multiplications	
$X \cdot Y = \langle 8 \cdot 12 \rangle_{13} = 5$	$\hat{X} \cdot \hat{Y} = \langle 11 \cdot 5 \rangle_{13} = 3$
conversion from QRNS	
$Z_R = \langle 7(5 + 3) \rangle_{13} = 4$	being $2^{-1} = 7$
$Z_I = \langle 7 \cdot 8(5 - 3) \rangle_{13} = 8$	being $q^{-1} = 8$

Table 1.1: Example of QRNS multiplication mod 13.

1.7 RNS and Fault-Tolerance

In the last decade, RNS fault-tolerant techniques have been studied extensively. These techniques are normally based on a RNS representation with suitable redundancy. In this way the resulting RNS structure can detect and correct errors in the output data. There are two methods for adding redundancy to the RNS representation

1. addition of one or more redundant moduli to the normal RNS representation obtaining the so called Redundant Residue Number Systems (RRNS) representation [28]

2. use of the Residue Number System Product Code (RNS-PC) [29].

In the following a basic introduction on the general principles in RNS error detection and correction techniques is given. Given a P moduli RNS representation, the addition of K moduli, such that for each of the added moduli $m_i > m_P$ (being m_P the largest modulo in the RNS representation), gives a RRNS representation. The set of non-redundant moduli constitutes the application dynamic range M,

$$M = \prod_{i=1}^{P} m_i \tag{1.8}$$

The remaining K moduli form the set of the redundant moduli allowing for error detection and correction. M_K is the product of the redundant moduli

$$M_K = \prod_{i=P+1}^{K} m_i \tag{1.9}$$

From 1.8 and 1.9 derives that $M_T = M \times M_K$. The interval $[0, M_P - 1]$ is the legitimate range, while the interval $[M_P,\ M_T - 1]$ is the illegitimate range. This extended number of states to represent the legitimate range permits error detection and correction [11].

For example the addition to the normal moduli set of one more modulo gives the possibility to determine whether a single error has occurred just by checking if it belongs to the legitimate or the illegitimate range. The same principle can be used in order to obtain single error correction. It can be obtained by using a set of two redundant moduli where each one must be larger than any of the moduli in the basic set. As a general consideration, the use of these algorithms requires MRC or CRT and consequently it is very important to rely on optimized hardware implementations.

Chapter 2

Implementation Aspects

In this chapter, the architectures that implement the basic modular operations and the conversions are described. These are the building blocks for implementing the most important DSP algorithms such as, for example, FIR filters. Specifically, the architectures described are:

- modular addition
- modular multiplication
- reduction to modulus m
- input and output conversions

2.1 Modular Addition

The modular addition

$$\langle a_1 + a_2 \rangle_m$$

can be implemented by two additions. If the result of $a_1 + a_2$ exceeds the modulo (it is larger than $m - 1$), we have to subtract the modulo m. In order to speed-up the operation we can execute in parallel the two operations:

$$(a_1 + a_2) \quad \text{and} \quad (a_1 + a_2 - m).$$

If the sign of the three-term addition is negative, it means than the sum $(a_1 + a_2) < m$ and the modular sum is $a_1 + a_2$, otherwise the modular addition is the result of the three-term addition. The above algorithm can be implemented with two binary adders as shown in Figure 2.1.

The modular adder can be simplified when:

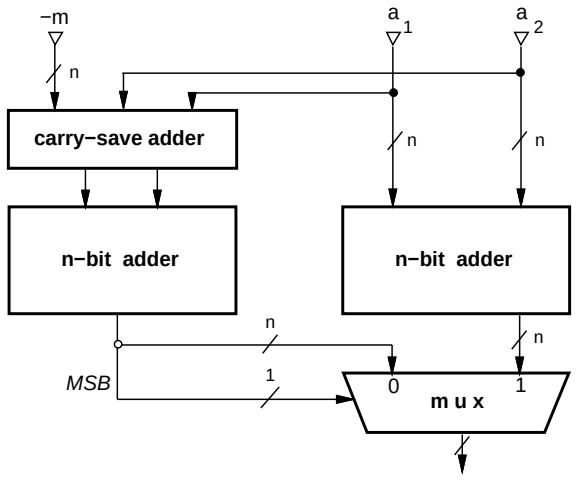


Figure 2.1: Architecture of the modular adder.

$m = 2^k$. In this case a modulo 2^k addition only requires the sum of the k least-significant bits.

$m = 2^k - 1$. In this case, the modulo $2^k - 1$ addition is computed as

$$a = \langle a_1 + a_2 \rangle_{2^k-1} = \langle\ a_1 + a_2 + MSB(a_1 + a_2)\ \rangle_{2^k}$$

and it can be implemented, eliminating the carry-save adder of Figure 2.1, with two adders in parallel, one with carry-in set to 0, the other to 1 and then selecting the correct result according to the MSB of the sum computed in the adder with carry-in=0.

2.2 Modular Multiplication

Because of the complexity of modular multiplication, it is convenient to implement the product of residues by the isomorphism technique (see Chapter 1.2.1). By using isomorphisms, the product of the two residues is transformed into the sum of their indices which are obtained by an isomorphic transformation. According to [10], if m is prime there exists a primitive radix q such that its powers modulo m cover the set $[1, m-1]$:

$$n = \langle q^w \rangle_m \qquad \text{with } n \in [1, m-1] \text{ and } w \in [0, m-2].$$

An example of isomorphic transformation is shown in Table 2.1 for $m = 5$. In this case, the primitive radix is $q = 2$.

n	w	$\langle q^w \rangle_m = n$
0	N/A	
1	0	$\langle 2^0 \rangle_5 = 1$
2	1	$\langle 2^1 \rangle_5 = 2$
3	3	$\langle 2^3 \rangle_5 = 3$
4	2	$\langle 2^2 \rangle_5 = 4$

Table 2.1: Example of isomorphic transformation for $m = 5$ ($q = 2$).

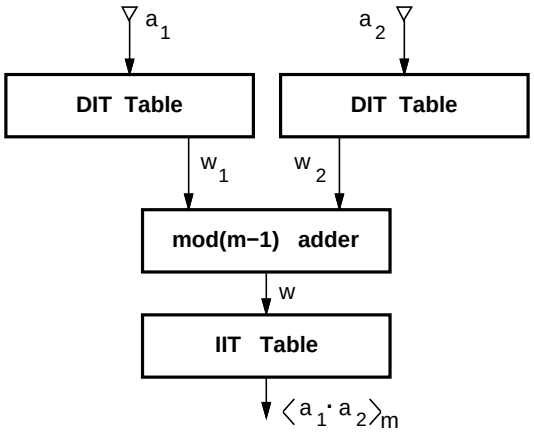


Figure 2.2: Structure of isomorphic multiplication.

Both transformations $n \rightarrow w$ and $w \rightarrow n$ can be implemented with $m-1$ entries look-up tables, if the moduli are not too large (less than 8-bit wide). Therefore, the product of a_1 and a_2 modulo m can be obtained as:

$$\langle a_1 \cdot a_2 \rangle_m = \langle q^w \rangle_m$$

where

$$w = \langle w_1 + w_2 \rangle_{m-1} \qquad \text{with } a_1 = \langle q^{w_1} \rangle_m \text{ and } a_2 = \langle q^{w_2} \rangle_m$$

In order to implement the modular multiplication the following operations are performed:

1) Two Direct Isomorphic Transformations (DIT) to obtain w_1 and w_2;

2) One modulo $m-1$ addition $\langle w_1 + w_2 \rangle_{m-1}$;

3) One Inverse Isomorphic Transformations (IIT) to obtain the product.

The architecture of the isomorphic multiplier is shown in Figure 2.2. Special attention has to be paid when one of the two operands is zero. In this case there exists no isomorphic correspondence and the modular adder has to be bypassed.

For example, for the modular multiplication $\langle 3 \cdot 4 \rangle_5 = 2$ using the isomorphic transformation of Table 2.1, we have

$$\begin{array}{ll} 1) & 3 = \langle 2^3 \rangle_5 \overset{DIT}{\rightarrow} w_1 = 3 \\ & 4 = \langle 2^2 \rangle_5 \overset{DIT}{\rightarrow} w_2 = 2 \\ 2) & \langle 2 + 3 \rangle_4 = 1 \\ 3) & 1 \overset{IIT}{\rightarrow} \langle 2^1 \rangle_5 = 2 \end{array}$$

2.3 Reduction to Modulus m

The sum modulus m of N operands is a useful operation in residual processors in order to avoid the growth of the data during the computations. In the following, it is illustrated a simplified architecture for the modulus extraction introduced in [30]. The sum of N addends mod m in general exceeds m:

$$\sum_{j=1}^{N} p_j = S > m \qquad \text{with} \qquad 0 \leq S \leq N(m-1) \ .$$

For small values of $N(m-1)$ (e.g. $N(m-1) \leq 2^7$), a $N(m-1)$-entry LUT can be used to associate to each S the corresponding $\langle S \rangle_m$.

For $N(m-1) > 2^7$, the technique shown in [30] is used. The basic idea is that by choosing an integer term k such that $N \leq 2^k$, it is possible to compute the following equation

$$\hat{S} = \frac{\sum_{j=1}^{N} \langle p_j \cdot 2^k \rangle_m - \alpha \cdot m}{2^k} \qquad \text{with} \qquad \begin{cases} 0 \leq & \alpha & < 2^k \\ -m \leq & \hat{S} & < m \end{cases}$$

where the term α is chosen to obtain an integer value of $\hat{S}$ when divided by 2^k. The term $\langle S \rangle_m$ can be obtained as

$$\langle S \rangle_m = \begin{cases} \hat{S} & \text{if } \hat{S} \geq 0 \\ \hat{S} + m & \text{if } \hat{S} < 0 \end{cases}$$

The architecture of the modulus reduction block is illustrated in Figure 2.3. In the figure, we assume the input to the modulus reduction block in carry-save (CS) representation: a redundant format that delays the carry-propagation to the last stages of the architecture. The delay of the scheme in Figure 2.3 is

$$t_{mod-red}(N, m) = c_2 \lceil \log_2 N \rceil + c_1 \lceil \log_2 m \rceil + c_0$$

where c_2, c_1 and c_0 are constants depending on the implementation technology.

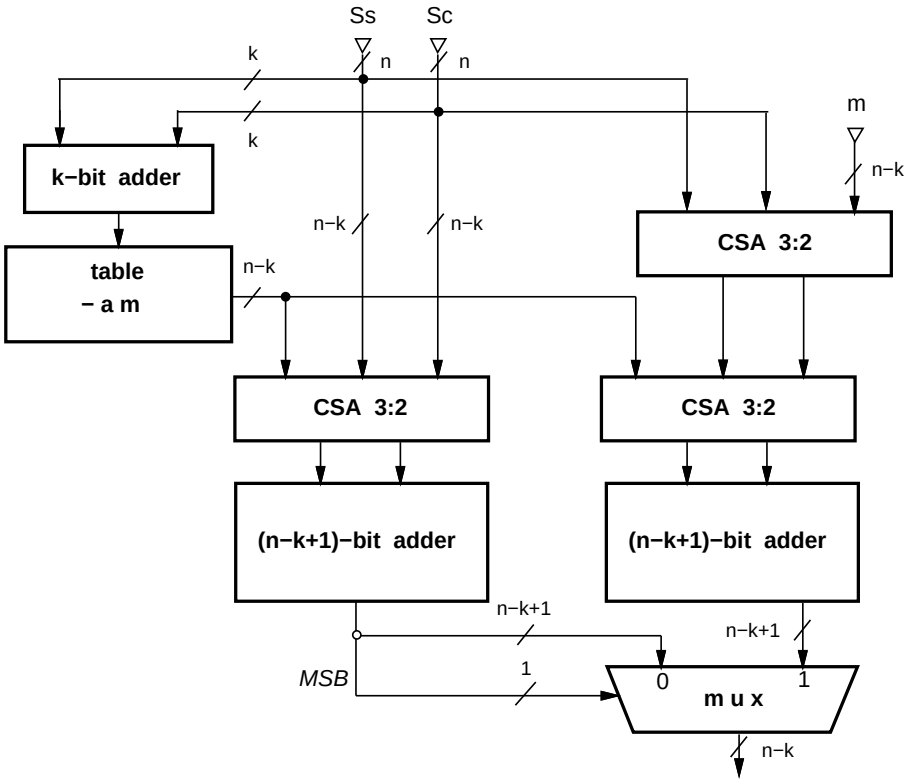


Figure 2.3: Modulo reduction block architecture

2.4 Input/Output Conversion

The advantages of the use of RNS can be limited by the input and output conversions required for the translation from the binary to the RNS representation, and vice-versa, especially when the number of operations to be executed for each input sample it is not high. In fact, the implementation of the converters constitutes a fixed overhead on the total area, delay and power dissipation. In this part of the report some considerations on the architectures for the input and output conversion are described.

2.4.1 Input Conversion

The input conversion, is implemented by using different techniques, from Look-Up Table (LUT) [12] to systolic architectures [15]. Informations about different conversion techniques can be found in a recent book [19] and some basic mathematics is shown in Section 1.3.1. In the following, it is illustrated a general technique for the input conversion for an odd modulus m. The architecture is based on the

structure presented in [15], and significant improvements are obtained by introducing a new algorithm for the computation of the modulo reduction stage. In [15], the conversion from binary to RNS (i.e. the modulus extraction $\langle X \rangle_m$) is obtained by applying three computational steps:

1. Initial mapping (one stage).
2. Dynamic range reduction (obtained by using reduction stages whose number and structure depend on the modulo m and the number of input bits).
3. Final mapping to the *mod* m ring based on LUT.

This architecture uses more than one stage for the dynamic range reduction step. These stages introduce latency and require resources. However, the dynamic range reduction step can simply be accomplished by introducing a scaling factor. As described in [15], the modulo operation can be expressed as

$$\begin{aligned} \langle X \rangle_m &= \left\langle \sum_{i=0}^{N-1} b_i 2^i \right\rangle_m = \left\langle \sum_{i=0}^{N-1} b_i \left\langle 2^i \right\rangle_m \right\rangle_m \\ &= \left\langle \sum_{i=0}^{L-1} p_i 2^i \right\rangle_m = \sum_{i=0}^{\lceil \log_2 (m-1) \rceil} d_i 2^i \end{aligned} \tag{2.1}$$

where N is the number of bits of the input data, $\langle \ \rangle_T$ is the *mod* T operator, b_i are the digits of the binary representation of the number X and $L = \left\lceil \log_2 \sum_{i=0}^{N-1} \left\langle 2^i \right\rangle_m \right\rceil$ is the number of bits needed to represent the maximum value of the summation (when all the digits b_i are 1).

Multiplying both sides of (2.1) by the scale factor 2^k, we obtain

$$k = \left\lceil \log_2 \left(\left\lfloor \frac{P_{max}}{m} \right\rfloor \right) \right\rceil \tag{2.2}$$

In (2.2) P_{max} represents the maximum value assumed by the term $\sum_{i=0}^{N-1} b_i \left\langle 2^{i+k} \right\rangle_m$. This value can be evaluated by simulation. However if 2 is a primitive radix of m, the bound for P_{max} can be found analytically. In fact, the period of the sequence $\langle 2^i \rangle_m$ is $P = m - 1$ and the summation of its elements is $\frac{m(m-1)}{2}$, therefore

$$P_{max} < \left(\left\lfloor \frac{N}{P} \right\rfloor + 1 \right) \frac{m(m-1)}{2} \tag{2.3}$$

Multiplying both the sides of (2.1) by 2^k we obtain

$$\langle X2^k \rangle_m = \left\langle \sum_{i=0}^{L-1} b_i \left\langle 2^{i+k} \right\rangle_m \right\rangle_m \tag{2.4}$$

and finally

$$X = \frac{\sum_{i=0}^{L-1} b_i \left\langle 2^{i+k} \right\rangle_m - \alpha m}{2^k} = \frac{H - \alpha m}{2^k} \tag{2.5}$$

In (2.5), the term αm is related to the *mod* m operation and its evaluation is accomplished by using a LUT addressed by the k least significant bits (lsb's) p_i of the term H [18]. The quantity H is evaluated by using a systolic architecture as in [15], while the modulo extraction is performed by using the technique presented in [18]. If the result of (2.5) is negative, a final addition of the quantity m must be performed.

This algorithm can be extended for dealing with signed numbers. In a N bits two's complement representation a negative number x is represented by the positive number X defined as

$$X = 2^N + x \tag{2.6}$$

The extension can be performed by considering the binary expression of (2.6)

$$X = 2^N + x = \sum_{i=0}^{N-1} b_i 2^i \tag{2.7}$$

where b_{N-1} represents the sign bit. That bit is 0 for positive numbers and 1 for the negative ones. From the above expression we can obtain the number x and applying the *mod* m operator on both the terms we get

$$\langle x \rangle_m = \langle b_0 + b_1 2^1 + \cdots + b_{N-1}(2^{N-1} - 2^N) \rangle_m = \left\langle \sum_{i=0}^{L} p_i 2^i \right\rangle_m \tag{2.8}$$

The expression is similar to that of (2.1), the only difference is that the bit b_{N-1} is used to add the quantity $\langle 2^{N-1} - 2^N \rangle_m$. As consequence the same architecture can be used to convert signed and unsigned numbers and just a modification of the constants factors is required.

In the following a numerical example shows the algorithm. Let us consider the case of $N = 4$ input bits. In the case of unsigned representation the range is $[0, 15]$. Instead, for two's complement representation the range is $[-8, 7]$. If we choose the modulus $m = 3$, for two complement numbers equation (2.8) becomes

$$\langle X \rangle_3 = \langle b_0 + b_1 \cdot 2^1 + b_2 \cdot 1 + b_3 \cdot 1 \rangle_3 \tag{2.9}$$

Applying the above equation to the input value $x = 5$, whose binary representation is $b_0b_1b_2b_3 = 1010$ we get

$$\langle 5 \rangle_3 = \langle 1 + 0 \cdot 2^1 + 1 \cdot 1 + 0 \cdot 1 \rangle_3 = \langle 2 \rangle_3 = 2 \tag{2.10}$$

On the other hand, if the input assumes the value -5, corresponding to the binary number $b_0b_1b_2b_3 = 1101$ eq.(2.9) becomes

$$\langle -5 \rangle_3 = \langle 1 + 1 \cdot 2^1 + 0 \cdot 1 + 1 \cdot 1 \rangle_3 = \langle 4 \rangle_3 = 1 \tag{2.11}$$

The Hardware Architecture

The converter hardware architecture in the case of $m = 17$ and $N = 7$, is shown in Figure 2.4. The LUT used to obtain the value αm, addressed by the bits p_0 and p_1, is embedded in the systolic architecture. The final addition of the quantity m (if the term H is negative) has been obtained by using a final systolic hardwired adder. A comparison with the architecture proposed in [15] is shown in Table 2.2, for $m = 17$, $N = 7$.

In this table, T_{Cell} represent the delay of the FA cell and T_{Rom} is the ROM access time. The presented architecture requires less hardware resources and has a smaller latency with respect to the architecture presented in [15]. In particular, a LUT of 4 cells (corresponding to $k = 2$) is required while using the method [3] a LUT of 32 cells must be used. A so small LUT can be easily implemented by using very simple structures for example based on mux trees. Due to this fact, the proposed algorithm permits to embed the LUT into the systolic architecture without loss of speed. The advantage over the approach presented in [15] is more evident when the number of input bits N becomes large.

HW resources	This impl.	Impl. [15]
Registers	65	84
FA	8	11
HA	16	8
LUT (N. of bits)	16 bits	160 bits
Latency	15	17
T_c	T_{cell}	$T_{Cell} + T_{Rom}$

Table 2.2: Input converter comparison.

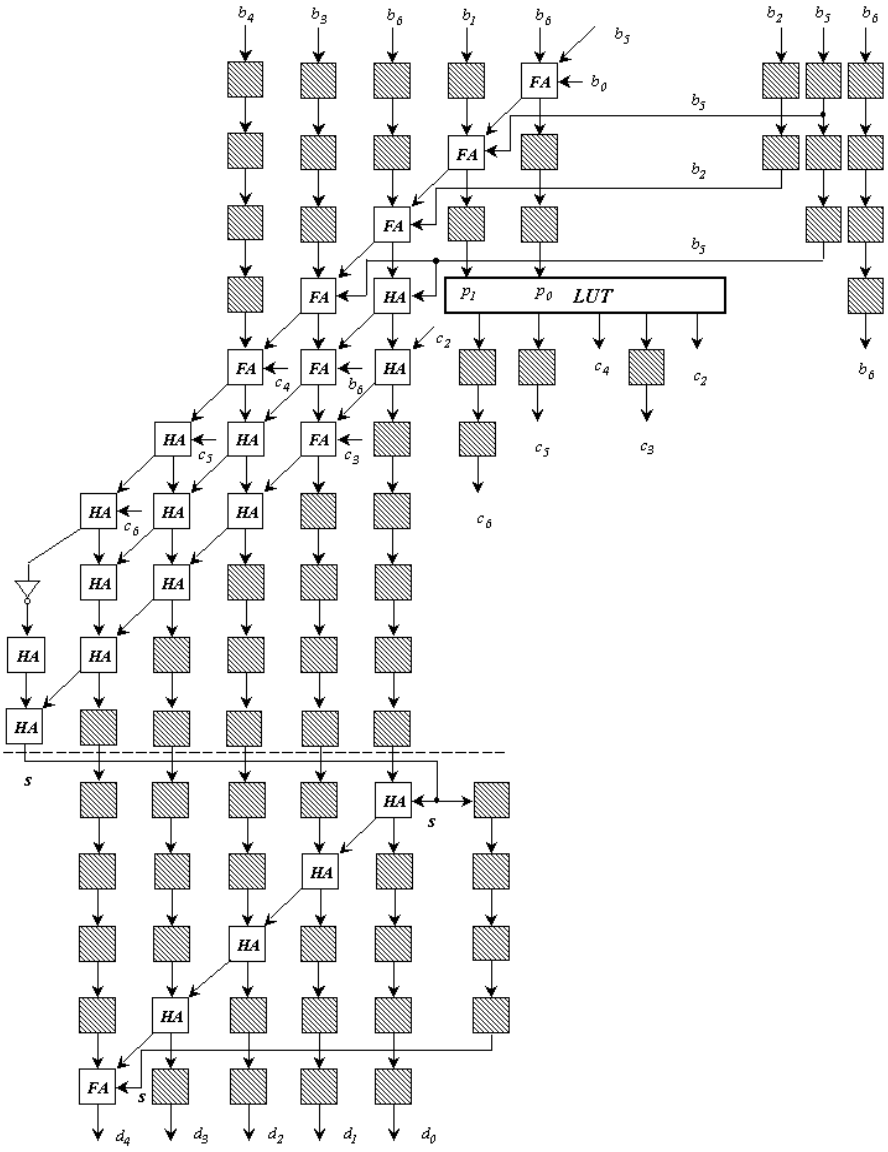


Figure 2.4: Hardware implementation of the input converter for $N = 7$ and $m = 17$.

2.4.2 Output Conversion

As shown in Section 1.3.2 the output conversion, is usually performed by using the Chinese Remainder Theorem and still appears to be a crucial point in the realization of competitive RNS systems specially when the number of operations to be implemented in the RNS before the conversion is not high.

For the output conversion, some authors proposed the use of three moduli sets [31], [32], [17], [33], [34] to obtain simpler and more efficient conversion architectures. For example in [33] and [34] the set $(2n-1,\ 2n,\ 2n+1)$ has been considered. Of course, this approach reduces the exploitation of the RNS properties (the maximum advantages are obtained by using a lot of small value moduli as shown in Section 1.4).

In fact, P grows with the desired wordlength and, consequently, the resulting modular processor becomes slower. On the other hand, high speed and low power multimedia applications require computations with large dynamic range and fine granularity in the wordlength selection (this aspect is related to the wordlength optimization phase in the fixed point optimization step).

Of course these requirements cannot be fully matched by using three moduli. To overcome these problems, in a lot of applications moduli sets with more than three moduli are required.

In [18] an efficient method for the RNS-Binary conversion, based on a set of P moduli, has been proposed. Although this method does not limit the number of moduli, it imposes an important limitation because only odd moduli can be used. This reduces the RNS advantages because power of two modular arithmetic exhibits very efficient implementations (for this reason, normally the greatest modulo is chosen of the form 2^h).

When (1.5) is implemented by a digital circuit two problems arise. The first one concerns the complexity of the involved arithmetic operations (a set of modulo additions and modulo multiplications). There are a number of methods to efficiently implement the computation of the term H. In [35] look-up tables (LUT) are used to compute the terms and a tree of carry save adders implements the summation. The second problem is related to the computation of the modulo M operation. This operation is complex [36] due to the large value of M and to the dynamic range of the term H. In fact, from (1.5) we obtain the following bounds

$$0 \leq H = \sum_{i=1}^{P} \overline{m}_i \langle \overline{m}_i^{-1} \cdot Z_i \rangle_{m_i} \leq \sum_{i=1}^{P} \frac{M}{m_i} \cdot (m_i - 1) < P \cdot M \qquad (2.12)$$

Equation (2.12) shows the relation between the range of H and P. Moreover, the methodologies used for the modulo computation of specific modulus set (as those based on moduli close to powers of two) do not appear to be useful for this modulo operation. Indeed, if we maintain the generality of the procedure, the final modulo cannot be constrained. To obtain a more suitable form for the *mod* M

operation, let us consider the number $Z \cdot 2^k$ being k a suitable integer quantity. Multiplying both the members of (1.5) by 2^k we obtain

$$\langle Z \cdot 2^k \rangle_M = \left\langle \sum_{i=1}^{P} \overline{m}_i \langle \overline{m}_i^{-1} \cdot Z_i \cdot 2^k \rangle_{m_i} \right\rangle_M \tag{2.13}$$

The terms of the summation in (2.13) have the same dynamic range as given by (2.12) since the factor 2^k appears inside a *mod* m_i operation. Equation (2.13) can be rewritten as

$$Z \cdot 2^k = \sum_{i=1}^{P} \overline{m}_i \langle \overline{m}_i^{-1} \cdot Z_i \cdot 2^k \rangle_{m_i} - \alpha \cdot M \tag{2.14}$$

where α comes from the external modulo operation. From (2.14) we get

$$Z = \frac{\sum_{i=1}^{P} \overline{m}_i \langle \overline{m}_i^{-1} \cdot Z_i \cdot 2^k \rangle_{m_i} - \alpha \cdot M}{2^k} = \frac{H - \alpha \cdot M}{2^k} \tag{2.15}$$

Properties of (2.15) has been exploited in [18]. Due to the presence of a power of two modulus, this expression cannot be directly used for the computation of the output conversion. In the present case, (2.14) must be modified taking into account that one of the residues, is a power of two (we suppose $m_P = 2^h$). In this case, we have

$$\langle Z \rangle_{2^h} = Z_P \tag{2.16}$$

From (2.16) derives that the h least significant bits of Z correspond to the h bits of Z_P. This means that the reconstruction of these bits does not require any operation in the residue to binary conversion process. In this case, the main task of the converter is the reconstruction of the remaining most significant bits of Z. These bits correspond to the number ε defined as

$$\varepsilon = \frac{Z - \langle Z \rangle_{2^h}}{2^h} = \frac{Z - Z_P}{2^h} \tag{2.17}$$

Starting from this value the converted value Z can be obtained by

$$Z = \varepsilon \cdot 2^h + Z_P. \tag{2.18}$$

The ε value can be computed by introducing (2.15) in (2.17)

$$\varepsilon = \frac{\frac{H - 2^k Z_P}{2^h} - \alpha \widetilde{M}}{2^k} \tag{2.19}$$

where $\widetilde{M} = M/2^h$. Since the definition of the term H implies that

$$\langle H \rangle_{2^h} = \left\langle 2^k Z_P \right\rangle_{2^h} \tag{2.20}$$

the first term of the numerator of (2.19) is an integer quantity $\widetilde{H}$ given by

$$\widetilde{H} = \frac{H - 2^k Z_P}{2^h} \tag{2.21}$$

Using (2.21), (2.19) can be rewritten as

$$\varepsilon = \frac{\widetilde{H} - \alpha \cdot \widetilde{M}}{2^k} \tag{2.22}$$

Due to the scaling by the factor 2^h, this expression requires for its computation a reduced dynamic range. Eq.(2.22) is similar to (2.15) and, as we show later, a simplified method can be used to select the value $\alpha\widetilde{M}$. In the following, all the expressions are defined in terms of ε, $\widetilde{H}$, $\widetilde{M}$.

The most difficult task, in the evaluation of (2.22), is the computation of the term $\alpha\widetilde{M}$. To solve this problem, we firstly evaluate the dynamic range of the term $\widetilde{H}$. Starting from (2.21) we obtain

$$-2^k < \widetilde{H} < P \cdot \widetilde{M} \tag{2.23}$$

consequently, the factor α belongs to the interval

$$-2^k < \alpha < P \tag{2.24}$$

Starting from this result, (2.22) suggests an efficient method to find the right value $\alpha \cdot \widetilde{M}$ to be subtracted to $\widetilde{H}$. In fact, in order to obtain integer values of ε (the reconstructed value), the quantity $\widetilde{H} - \alpha \cdot \widetilde{M}$ must be a multiple of the factor 2^k. This means that the k least significant bits of $\widetilde{H} - \alpha \cdot \widetilde{M}$ must be equal to zero. Starting from this observation, we can derive that the correct value of the term α belongs to the subset

$$\Upsilon = \{\alpha \in I : \langle \alpha \cdot \widetilde{M} \rangle_{2^k} = \langle \widetilde{H} \rangle_{2^k}\} \tag{2.25}$$

Where I is the set of integer numbers. This subset only depends on the k least significant bits of $\widetilde{H}$. Unfortunately, using these bits we are able to select only 2^k values of $\alpha \cdot \widetilde{M}$, out of the $P + 2^k + 1$ possible values, according with (2.23). If k is chosen such that

$$2^k \geq P - 1 \tag{2.26}$$

the values of $\alpha \cdot \widetilde{M}$ can be computed starting from the 2^k positive values stored in a very small LUT. In fact, since ε must be a positive number, the quantity $\widetilde{H} - \alpha \cdot \widetilde{M}$ must be positive. If this does not happens, the obtained value of $\alpha \in \Upsilon$ is incorrect. From (2.23) and (2.25) the correct value is obtained by subtracting

2^k from the incorrect one. So, if α' is the incorrect value addressed by the LUT and α is the correct one, ε is obtained by

$$\varepsilon = \frac{\widetilde{H} - \alpha \cdot \widetilde{M}}{2^k} = \frac{\widetilde{H} - \alpha' \cdot \widetilde{M}}{2^k} + \widetilde{M} \tag{2.27}$$

The procedure deriving from (2.27) can be summarized by the following steps:

1. The term $\alpha' \cdot \widetilde{M}$ is read from the LUT addressed by the k least significant bits of $\widetilde{H}$.
2. The sum $\widetilde{H} - \alpha \cdot \widetilde{M}$ is computed and the k least significant bits are discarded.
3. If the obtained result is negative the quantity $\widetilde{M}$ is added.

The conversion into a two's complement representation can be easily performed by using the following conventions for the RNS representation of signed numbers. Since M is even, positive numbers are into the range $[0, (M/2) - 1]$ and negative ones are in $[M/2,\ M - 1]$.

The signed conversion must translate these ranges into the ranges of the two's complement representation. This translation can be performed by considering the following procedure. As first step we add, *mod* M, the quantity $S = M/2$. This operation translates the positive numbers into the range $[M/2,\ M - 1]$, while the negative ones are now in the interval $[0,\ (M/2) - 1]$. As a final step, the two's complement value of the output can be reconstructed through the binary subtraction of the value $M/2$ from the final result.

This procedure has been embedded in the algorithm and in order to reduce the computation steps, the final subtraction has been merged with the conditional subtraction required for the α correction. Therefore if the reconstructed value $Z' = \varepsilon' \cdot 2^h + \langle Z_N + S \rangle_{2^h}$ is positive we only subtract the value S. Otherwise, for negative values, the quantity $M - S = M/2$ is added. The above algorithm can be summarized in the following steps

1. Compute the quantity $\widetilde{H}$ using the modified residue $Z_i' = \langle Z_i + S \rangle_{m_i}$.
2. Compute the quantities ε', $Z' = \varepsilon' \cdot 2^h + \langle Z_P + S \rangle_{2^h}$.
3. Compute the quantity Z. If Z' is negative the two's complement output result is obtained as $Z = Z' + M - S$ otherwise $Z = Z' - S$

In the following, a numerical example is given. Let us consider the case of a RNS representation based on the moduli set,

$$m_i = \{3,\ 5,\ 7,\ 8\}$$

where $Z_4 = 2^3$ (i.e. $h = 3$). The number of moduli is four therefore, from (2.26), $k = 2$. For this set we have

$$\overline{m}_i = \{280,\ 168,\ 120,\ 105\}, \quad \overline{m}_i^{-1} = \{1,\ 2,\ 1,\ 1\}, \quad M = 840$$
$$\widetilde{M} = 105, \qquad\qquad S = 420$$

and

$$H = 280\langle 1{\cdot}2^k{\cdot}(Z_1{+}S)\rangle_3 - 168\langle 2{\cdot}2^k{\cdot}(Z_2{+}S)\rangle_5 + 120\langle 1{\cdot}2^k{\cdot}(Z_3{+}S)\rangle_7 + 105\langle 1{\cdot}2^k{\cdot}(r_4{+}S)\rangle_8$$

Consider the value $Z = -209 \xrightarrow{RNS} \{1,\ 1,\ 1,\ 7\}$.

$$H = 1684, \quad \widetilde{H} = \frac{1684 - 4{\cdot}\langle 7+420\rangle_8}{8} = 209$$

The correct α value is 1. Consequently we have $\varepsilon' = 26$ and for Z' we obtain $Z' = 26 \cdot 8 + \langle 7 + 420\rangle_8 = 211 > 0$.

In this case we have to subtract the term $S = 420$ obtaining $Z = 211 - 420 = -209$.

Implementation

The converter architecture is sketched in Figure 2.5. The P LUTs are addressed by the residues Z_i and store the terms

$$\overline{m}_i \langle \overline{m}_i^{-1} \cdot 2^k \cdot (Z_i + S)\rangle_{m_i}$$

The LUT-P stores the term $\overline{m}_P \langle \hat{m}_P^{-1} 2^k (Z_P + S)\rangle_{m_P} - 2^k \langle Z_P + S\rangle_{2^h}$. A Carry-Save Adder (CSA) is used to compute $\widetilde{H}$. The k least significant bits of $\widetilde{H}$ are used to address the LUT $\alpha\widetilde{M}$ that stores the multiples $\alpha'\widetilde{M}$. The selected multiple is added to $\widetilde{H}$ in order to obtain the value ε'. The h least significant bits of the value $\langle Z_P + S\rangle_{2^h}$ are directly juxtaposed with ε' to obtain the value Z'. The correct signed value Z is obtained by a final summation. Depending on $Sgn(Z')$ the value $-S$ or $M - S$ is conditionally added to Z'.

A VLSI implementation based on the moduli set $\{3,\ 5,\ 7,\ 11,\ 17,\ 64\}$ for a 20 bit converter has been implemented (Figure 2.6). The architecture requires six small LUTs. In fact the input LUTs are related to the moduli wordlength that can be chosen sufficiently small for the most common dynamic ranges. The computation of the term $\widetilde{H}$ has been obtained by using a Carry-Save Adder (CSA), and a carry-save representation has been maintained where possible. A fast Carry-Propagate Adder (CPA) has been used to obtain the address to the LUT-$\alpha\widetilde{M}$. In the architecture, two different results are computed in parallel and the correct one is selected by using $Sgn(\epsilon')$. The architecture has been mapped on a XILINX-V1000-6 FPGA. The number of used Configurable Logic Blocks (CLB) is 80 and the maximum delay is 14 nsec (taking into account also the routing delays).

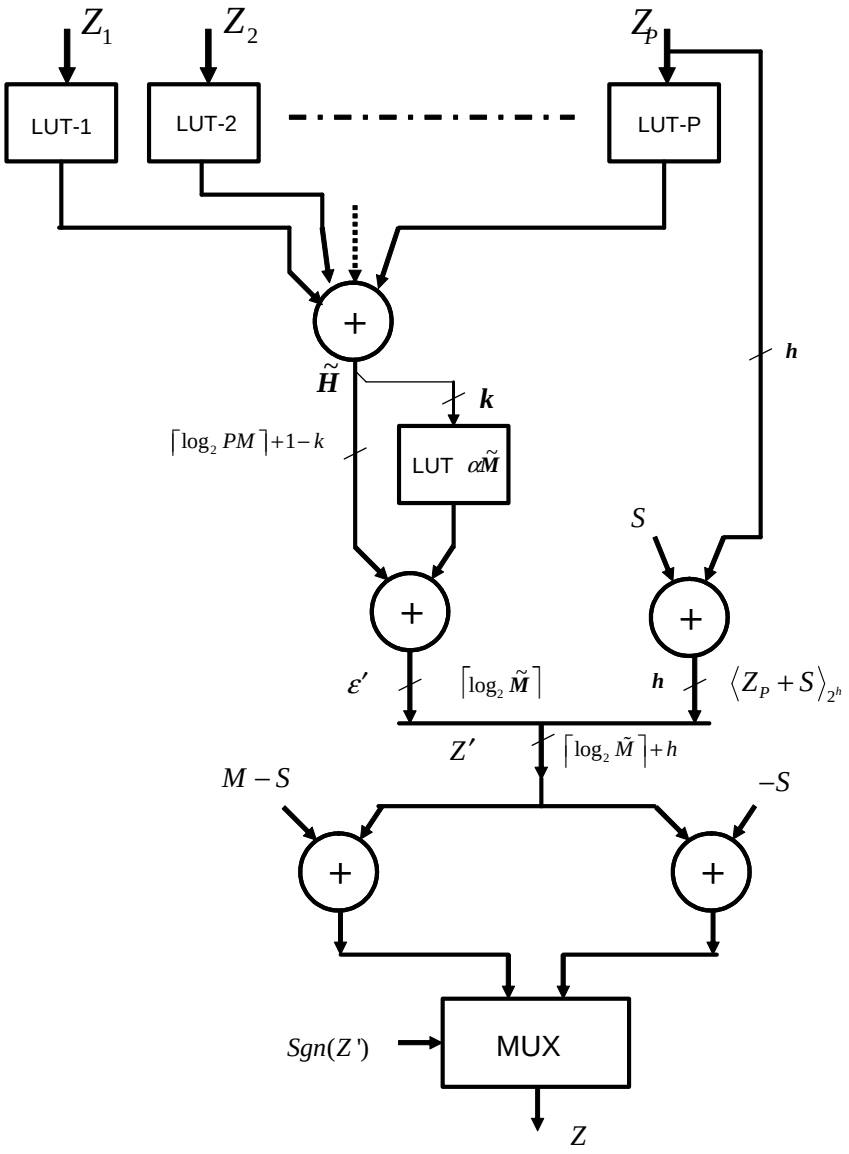


Figure 2.5: Output converter architecture.

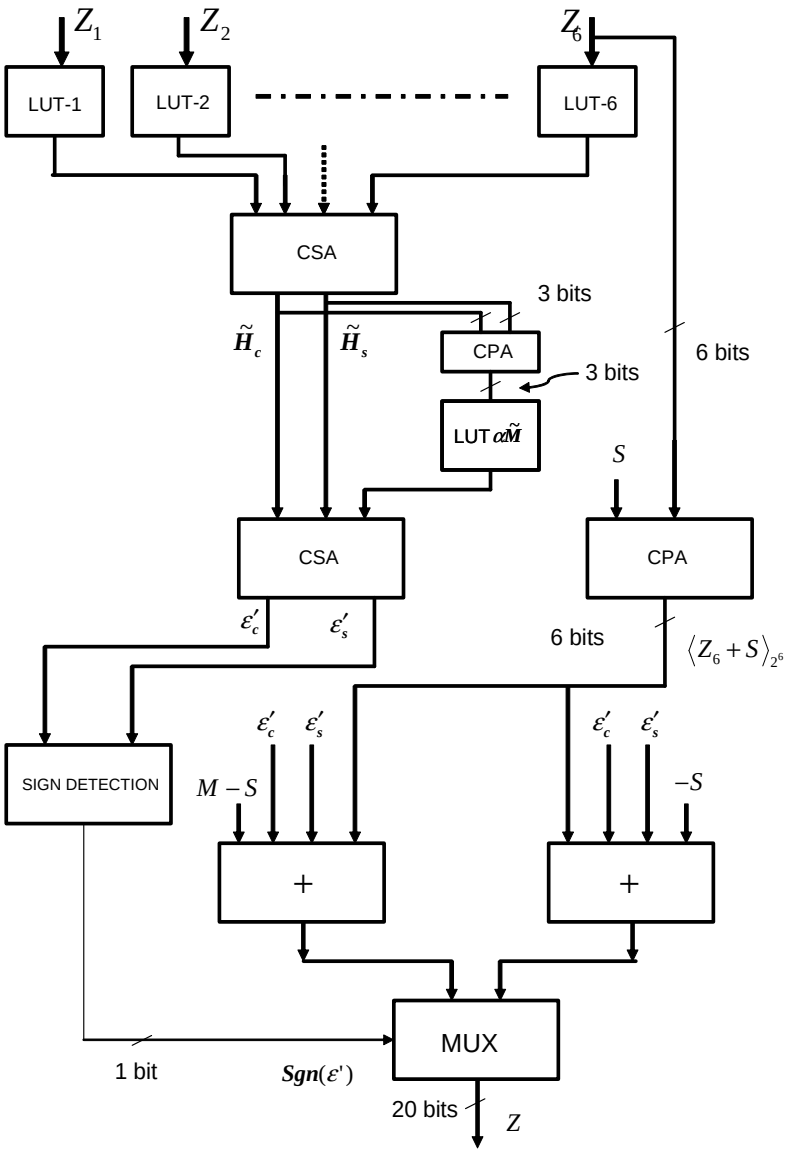


Figure 2.6: Implementation of output converter.

Chapter 3

Case Study: FIR Filters

In this chapter we present the case study of Finite Impulse Response (FIR) filters, for which the RNS implementation is quite convenient with respect to the traditional implementation in the Two's Complement System (TCS).

A Finite Impulse Response (FIR) filter of order N is described by the expression

$$y(n) = \sum_{k=0}^{N-1} a_k x(n-k) \tag{3.1}$$

FIR filters can be either realized in transposed or direct form (Figure 3.1).

The design a FIR filter is typically implemented by suitable CAD tools (such as Matlab for example) in Floating Point Arithmetic (FLP). In the majority of the cases, when requirements of power consumption and speed are stringent, the filter is implemented in Fixed Point Arithmetic (FXP). In this case, the filter coefficients are rounded to a specific number of bits (the number of used bits has an impact on the frequency mask of the filter) and two different strategies can be chosen in order to limit the growth in the number of bits

1. **Truncation after multiplication.** In this case, to limit the number of bits, truncation after multiplication is used (Figure 3.2.1). This approach is typically used in the parallel implementation of FIR filters to reduce hardware resources. Accurate FXP simulations must be done to evaluate the truncation effects on the filter performace and define the number of bits to be used for the representation of the internal variables.

2. **Truncation after accumulation.** In this case, to limit the number of bits, truncation after accumulation is used (Figure 3.2.2). This approach is typically used in the case of serial implementations of FIR filters (i.e. based on a Multiply and Accumulate Unit (MAC)). The final truncation or rounding are used to reduce the final number of bits. In this case, because

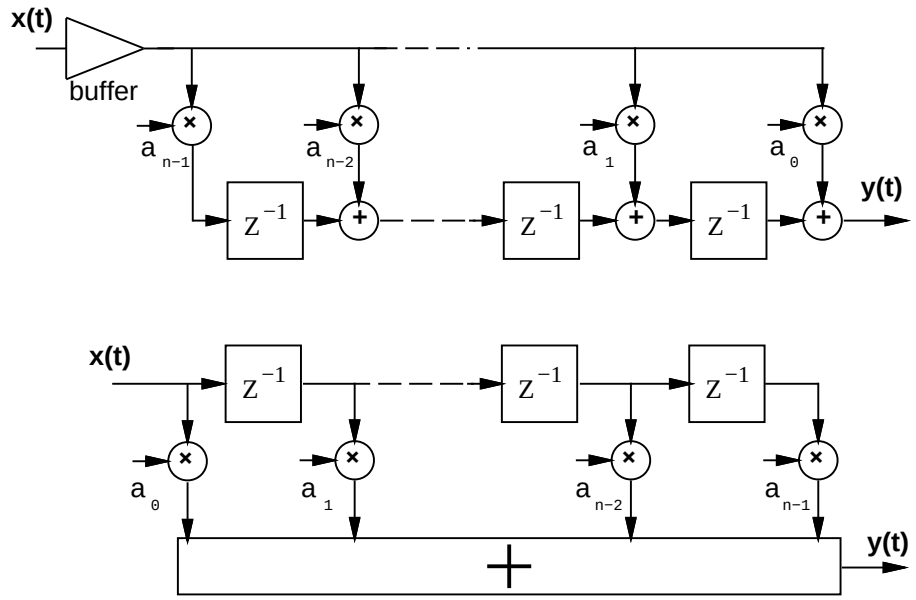
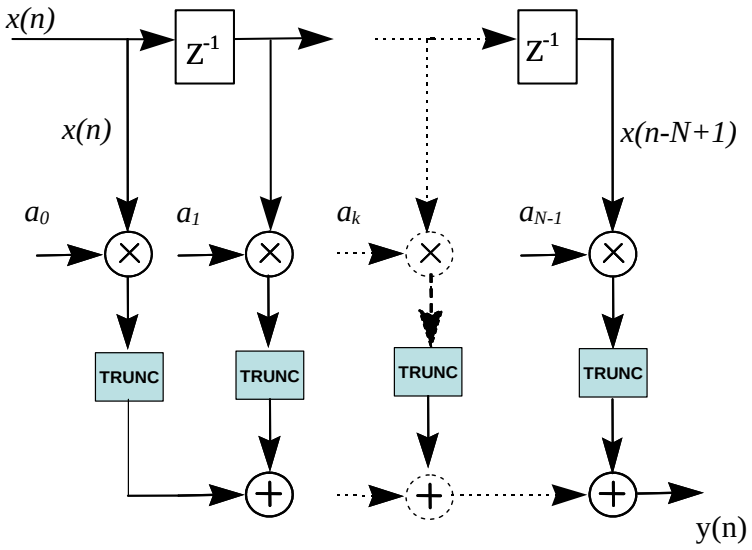


Figure 3.1: FIR filters in transposed (top) and direct (bottom) form.

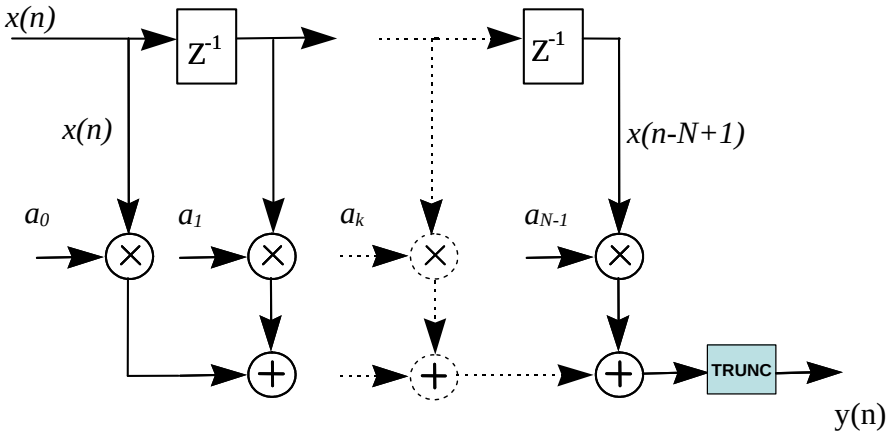
truncation, or rounding, is done only once, a smaller error is introduced and the filter will perform better than in case 1.

We compare filters over a vast range of architectures and show the results of the design with respect to delay, throughput, area and power dissipation.
The chapter describes the following filter designs and comparisons:

- FIR filters in transposed form with programmable/constant coefficients
- FIR filters in direct form with programmable/constant coefficients
- RNS carry-save (faster) filters
- Low power RNS filters
- Complex filters
- FPGA implementations of TCS and RNS filters
- Comparisons of results ASIC vs. FPGA.



1. Truncation after multiplication.



2. Truncation after accumulation.

Figure 3.2: Truncation schemes.

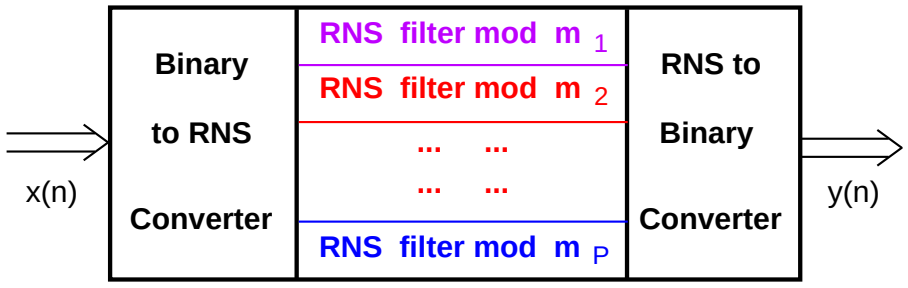


Figure 3.3: Architecture of RNS FIR filters.

3.1 RNS Implementation of FIR Filters

As a direct consequence of (1.1), expression (3.1) becomes in RNS:

$$y(n) = \sum_{k=0}^{N-1} a_k x(n-k) \overset{RNS}{\rightarrow} \begin{cases} Y_{m_1}(n) = \left\langle \sum_{k=0}^{N-1} \langle A_{m_1}(k) \cdot X_{m_1}(n-k) \rangle_{m_1} \right\rangle_{m_1} \\ Y_{m_2}(n) = \left\langle \sum_{k=0}^{N-1} \langle A_{m_2}(k) \cdot X_{m_2}(n-k) \rangle_{m_2} \right\rangle_{m_2} \\ \dots \quad \dots \quad \dots \\ Y_{m_P}(n) = \left\langle \sum_{k=0}^{N-1} \langle A_{m_P}(k) \cdot X_{m_P}(n-k) \rangle_{m_P} \right\rangle_{m_P} \end{cases} \tag{3.2}$$

and the filter can be implemented in RNS by decomposing it into P filters working in parallel, as sketched in Figure 3.3. In general, filters in different moduli paths can be implemented in different forms (direct or transposed), as long as the timing is consistent.

3.2 FIR Filters in Transposed Form

We start our case study with FIR Filters in transposed form (Figure 3.1 top).

3.2.1 Transposed FIR Filters in TCS

The implementation of (3.1) in transposed form is shown (for a portion of the filter) in Figure 3.4.

We assume to have a dynamic range of d bits, generated at the output of $\frac{d}{2} \times \frac{d}{2}$ square multipliers, which guarantees error-free operations for the given N.

The composing blocks of a FIR filter are multipliers, adders and registers. In the following, we describe the architectures chosen for implementing these composing blocks.

For the implementation of multipliers with the traditional binary system (TCS),

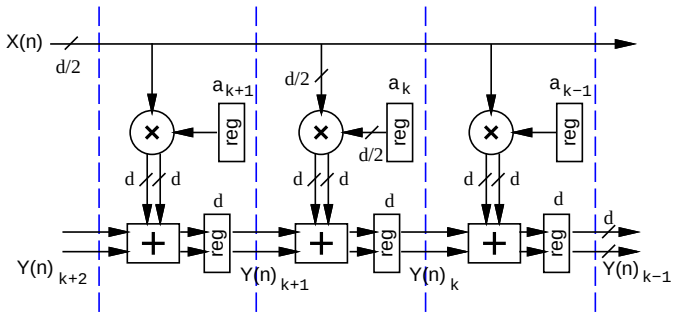


Figure 3.4: TCS FIR filter in transposed form.

we chose to keep the product in carry-save (CS) format to speed-up the operation, and delayed the assimilation of the CS representation to the last stage of the filter. For the FIR filter in transposed form (Figure 3.1 top), in each tap we need to add the CS representation of the product to the value stored in the register (previous tap). Again, to avoid the propagation of the carry, we can store the CS representation. For this reason, we need to implement the addition with an array of 4:2 carry-save adders (CSA), as shown in Figure 3.5.

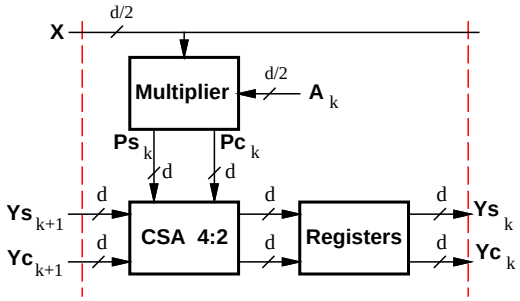


Figure 3.5: Tap structure for the transposed TCS FIR filter.

The CS representation is finally converted into the two's complement representation by a carry-propagate adder (realized with a carry-look-ahead scheme) in the last stage of the filter.

Figure 3.5 shows the implementation of the tap of a filter with programmable coefficients. For filters with constant coefficients (not programmable) the implementation scheme is the same with the only difference in the multiplier which is simplified.

Because the filter input $x(n)$ is broadcast to all taps for transposed form, buffering of $x(n)$ is necessary especially for high-order filters.

The critical path for the TCS filter in transposed form is

$$t_{TCS/tra} = t_{buffer} + t_{MULT} + t_{CSA-4:2} + t_{REG} \tag{3.3}$$

where

t_{buffer} is the delay of the buffer(s);

t_{MULT} is the delay of multiplier;

$t_{CSA-4:2}$, the delay of the 4:2 adder, is about $3 \times t_{xor}$ (delay of three XOR-2 gates);

t_{REG} is the sum of the register propagation delay and the set-up time.

The filter latency is one extra cycle needed to compute $y(n)$ from its CS representation ($y(n) = y_s(n) + y_c(n)$). The expression for the area as a function of the number of taps is:

$$A_{TCS/tra} = \left(A_{MULT} + d \cdot A_{CSA-4:2} + \frac{5}{2} d \cdot A_{FF} \right) N + A_{CPA} \tag{3.4}$$

where

A_{MULT} is the area of a $\frac{d}{2} \times \frac{d}{2}$ multiplier (output CS);

$A_{CSA-4:2}$ is the area of a 1-bit 4:2 CSA;

A_{FF} is the area of one flip-flop;

A_{CPA} is the area of the final carry-propagate adder.

For the power dissipation the expression at a fixed frequency f_0 is:

$$P_{TCS/tra} = P_{TAP-TCS/tra} \cdot N + P_{CPA} \tag{3.5}$$

with the limitation that the term $P_{TAP-TCS/tra}$ strongly depends on the switching activity [37] and consequently on the value of the filter coefficients.

3.2.2 Transposed FIR Filters in RNS

As a direct consequence of (3.2), a FIR filter is implemented in RNS by decomposing it into P filters working in parallel, as sketched in Figure 3.3. The bitwidth $s_i = \lceil \log_2 m_i \rceil$ in each filter mod m_i is such that $b = \Sigma_{i=1}^{P} s_i$. This b is generally larger than the dynamic range of the corresponding TCS filter

$$b \geq d \quad \text{with} \quad \prod_{i=1}^{P} m_i = M \geq 2^d$$

and this overhead is discussed in Section 3.4.1.

The RNS FIR filter is completed by an input and an output conversion block.

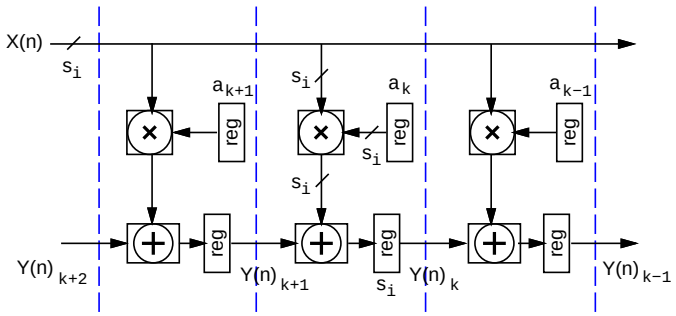


Figure 3.6: RNS FIR filter in transposed form.

Each of the P RNS filters can be implemented in transposed form with a scheme similar to that of Figure 3.4, and by replacing multipliers and adders with their modular counterparts.

By choosing prime moduli of limited wordlength ($m_i < 2^6$), the modular multiplication can be efficiently implemented by isomorphism (see Section 1.2.1 for the implementation). The resulting RNS filter architecture in transposed form is shown in Figure 3.6.

In case of filters with constant coefficients, the multiplication for a constant can be easily performed by a look-up table implemented with synthesized logic.

The addition is also a modular operation and a correction step is needed if the result of the binary addition exceeds the modulo (see Section 2.1 for implementation detail).

The critical path for the single modular filter in transposed form is:

$$t_{RNS/tra} = t_{buffer} + t_{modMULT} + t_{modADD} + t_{REG} \tag{3.6}$$

where

t_{buffer} is the delay of the buffer(s);

$t_{modMULT}$ is the delay of isomorphic multiplier modulo m_i;

t_{modADD} is the delay of the modular adder;

t_{REG} is the sum of the register propagation delay and the set-up time.

The delay of the critical path for RNS filters is the largest one among those of the P filters $t_{RNS/tra}$ and the in/out conversion delays. However, by pipelining the conversion units, normally the maximum delay is set by the slowest of the modular paths. We assume that this is the case in the rest of the chapter.

The latency is determined by how many pipeline stages are in the input and output conversion units.

The expressions for area and power are more complicated than those of the corresponding TCS filters. For the area, we have

$$A_{RNS/tra} = \left(\sum_{i=1}^{P} (A_{modMULTm_i} + A_{modADDm_i} + 2s_i \cdot A_{FF}) \right) N + A_{convIN} + A_{convOUT} \tag{3.7}$$

where

$A_{modMULTm_i}$ is the area of the modular multiplier mod m_i;

$A_{modADDm_i}$ is the area of the adder mod m_i;

s_i is the number of bits required for mod m_i;

A_{FF} is the area of one flip-flop;

A_{convIN} and $A_{convOUT}$ are the area of the converters. Both terms are a function of the filter dynamic range and the moduli selection.

Similarly, for the power dissipation we have

$$P_{RNS/tra} = \left(\sum_{i=1}^{P} P_{TAP-m_i/tra} \right) N + P_{convIN} + P_{convOUT} \tag{3.8}$$

where $P_{TAP-m_i/tra}$ is the average power dissipation in the tap of the filter mod m_i with the limitation that this average value depends on the filter coefficients values.

3.2.3 Transposed Truncated FIR Filters in TCS

Sometimes accuracy is traded with performance by implementing filters with truncated dynamic range. Truncation or rounding schemes are very tricky to be implemented in RNS, and for this reason we limited our investigation to TCS filters. We have tried different truncation schemes, but we report here only the case in which the results of multiplications are truncated after $d/2$ bits.

The expressions for delay, area and power dissipation are similar to those of (3.3), (3.4) and (3.5).

3.2.4 Transposed FIR Filters: Summary

To explore trade-offs for FIR filters realized in TCS and RNS we implemented a number of filters, measured their performance and interpolated the results to obtain the trend with respect to the filter order N (number of taps).

This first exploration is done by assuming a fixed dynamic range d. The filters are implemented in standard cells ($0.35\mu m$ library) and the delay, area and power dissipation are determined by Synopsys tools.

Filter 20-bit dyn. range	TCS				
	Cycle $[ns]$	Area (NAND2 equiv.)	N^*	Power [1] $[mW]$	N^*
var. coeff.	5.0	$1250N + 230$	6	$13.5N + 14.9$	6
const. coeff.[2]	4.9	$844N + 230$	8	$8.6N + 14.9$	8
TCS truncated					
var. coeff.	5.0	$860N + 120$	24	$9.2N + 7.1$	20
const. coeff.[2]	4.7	$565N + 120$	43	$5.8N + 7.1$	27

Filter 20-bit dyn. range	RNS		
	Cycle $[ns]$	Area (NAND2 equiv.)	Power [1] $[mW]$
var. coeff.	5.0	$745N + 2910$	$6.9N + 51.0$
const. coeff.[2]	4.3	$500N + 2910$	$4.2N + 51.0$

[1] Power at 100MHz assuming random input activity.
[2] Assuming 50% of bits set at 1 on average.

Table 3.1: FIR filters in transposed form: summary of results.

The chosen dynamic range is 20 bits, and to cover this range a possible choice of co-prime moduli is the following

$$m_i = \{3, 5, 7, 11, 17, 64\} \;\rightarrow\; \prod_{i=1}^{P} m_i > 2^{20}$$

This choice of moduli give best delay/area/power tradeoffs for the specific technology. The implementation results for filters in transposed form are summarized in Table 3.1.

By comparing the expressions of Table 3.1, we can determine what is the filter order N (number of taps) for which the RNS filter has a smaller area and consume less power than the corresponding TCS filter. For examples, for filter with variable coefficients the *"equal power N point"* N^* is

$$13.5N^* + 14.9 = 6.9N^* + 51.0 \quad \rightarrow \quad N^* = \frac{51.0 - 14.9}{13.5 - 6.9} \simeq 5.5$$

This means that for filters of order greater than 6 the RNS filter dissipates less power than the TCS filter, or, in other words, the overhead of the RNS input/output converters does not offset the power reduction when the filter has more than 6 taps. Even if we compare the truncated TCS filters with error-free RNS filters, the RNS filters show smaller area and power dissipation for the typical size of filters ($N \gg 43$). The trends for the area and the power dissipation as a function of N are shown in Figure 3.7.

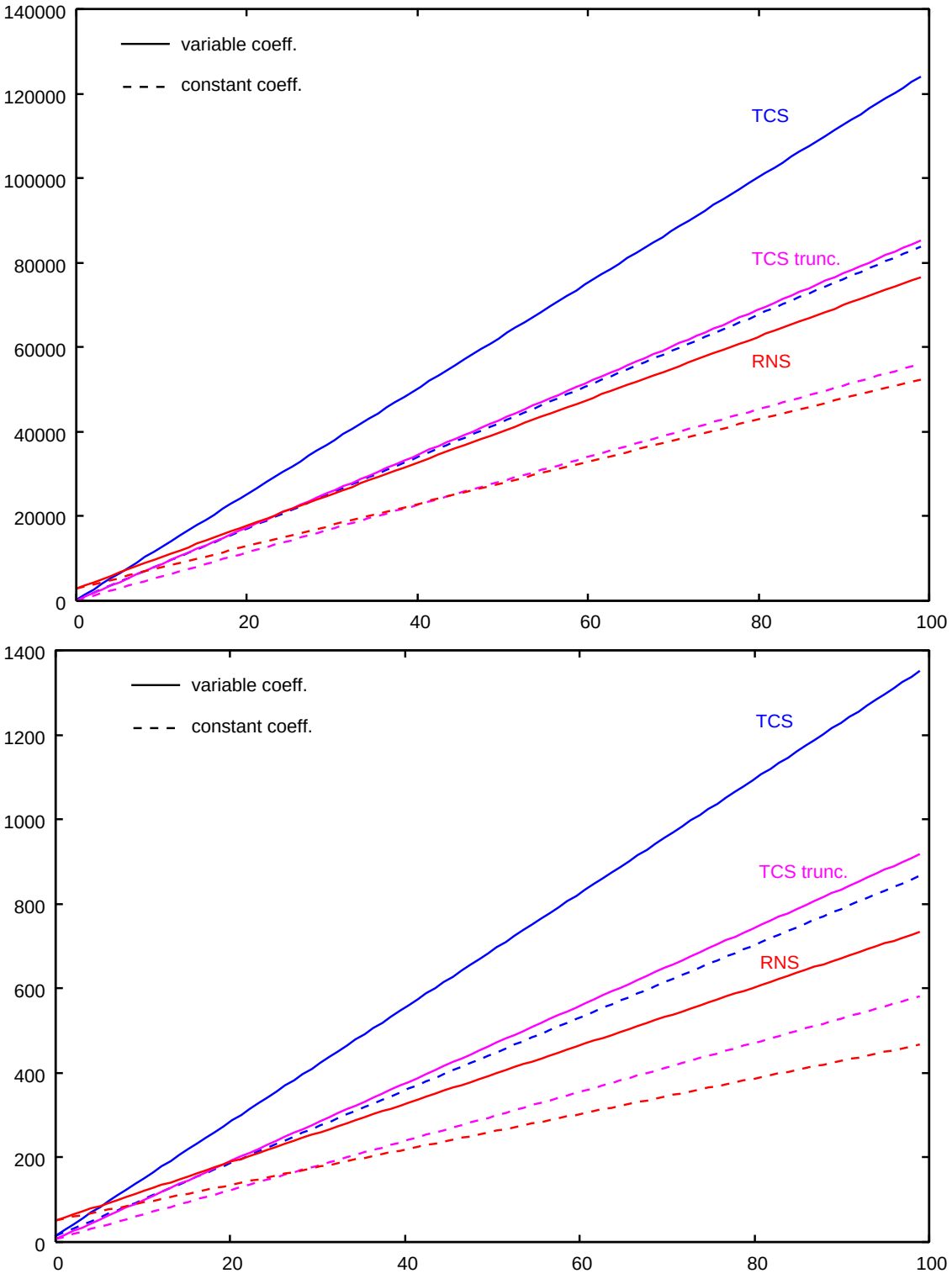


Figure 3.7: Trends for transposed FIR filters: area (top) and power dissipation (bottom).

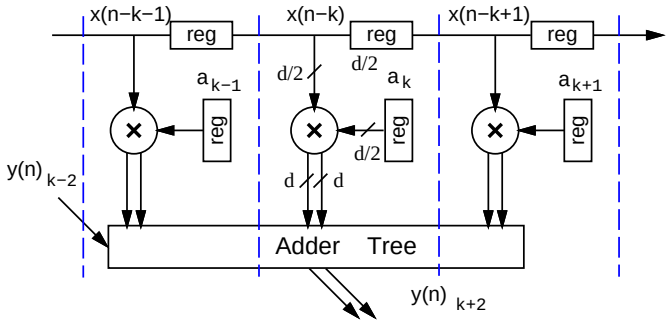


Figure 3.8: TCS FIR filter in direct form.

3.3 FIR Filters in Direct Form

For the FIR filter in direct form (Figure 3.1 bottom), it is convenient to implement the addition with a Wallace's tree to speed-up the operations. However, the use of the tree makes the design of the filter not modular in the number of taps and the filter latency changes with the filter order N.

3.3.1 Direct FIR Filters in TCS

The implementation of the filter in direct form is shown (for a portion of the filter) in Figure 3.8.

Similarly to the TCS filters in transposed form, we assume a dynamic range of d bits for error-free operations for the given N. The output of the $\frac{d}{2} \times \frac{d}{2}$ square multipliers is carry-save. By opting for the CS representation of the products, for a N-tap filter we have to add $2N$ addends. The resulting critical path is

$$t_{TCS/dir} = t_{MULT} + t_{CSA-4:2} \cdot k + t_{REG} \tag{3.9}$$

where

$k = \lceil \log_2 N \rceil$ assuming the tree realized with 4:2 CSAs;

t_{MULT} is the delay of multiplier;

$t_{CSA-4:2}$, the delay of the 4:2 adder;

t_{REG} is the sum of the register propagation delay and the set-up time.

The approximated expression for the area is

$$A_{TCS/dir} = (A_{MULT} + d \cdot A_{FF})\, N + d \cdot A_{CSA-4:2}(2^{\lceil log_2 N \rceil} - 1) + d \cdot A_{FF} + A_{CPA} \tag{3.10}$$

where

A_{MULT} is the area of the CS multiplier;

$A_{CSA-4:2}$ is the area of a 1-bit 4:2 CSA;

A_{FF} is the area of one flip-flop;

A_{CPA} is the area of the final carry-propagate adder.

If N is close to a power of 2 ($2^{\lceil log_2 N \rceil} \simeq N$), the area is linear with respect to N:

$$A_{TCS/dir} = [A_{MULT} + d(A_{FF} + A_{CSA-4:2})]\, N - d \cdot A_{CSA-4:2} + d \cdot A_{FF} + A_{CPA} \tag{3.11}$$

For the power, we obtain an expression similar to (3.10) as a function of N. However, because the switching activity in the tree is strongly dependent on its size (due to glitches), the values obtained by the formula are not quite accurate.

3.3.2 Direct FIR Filters in RNS

A single filter mod m_i can be implemented in direct form as shown in Figure 3.9 for a portion of the filter. A N-input Wallace's tree, followed by a block to compute the modulo (see Section 2.3), is needed. At the output of the tree, the dynamic range is increased from s_i to $s_i + k$ ($k = \lceil \log_2 N \rceil$). The expressions for the delay of the critical path in the mod m_i filter is

$$t_{RNS/dir} = t_{modMULT} + t_{CSA-4:2} \cdot (k-1) + t_{mod-red} + t_{REG} \tag{3.12}$$

where

$k = \lceil \log_2 N \rceil$ assuming the tree realized with 4:2 CSAs;

$t_{modMULT}$ is the delay of isomorphic multiplier modulo m_i;

$t_{CSA-4:2}$, the delay of the 4:2 adder;

$t_{mod-red}$ is the delay of the circuit to extract mod m_i;

t_{REG} is the sum of the register propagation delay and the set-up time.

The tree in the RNS direct filter is one level shallower than the TCS filter. However, the extra modulo operation impacts the critical path (and area and power as well). The expressions for area is

$$A_{RNS/dir} = \sum_{i=1}^{P} [\ (A_{modMULTm_i} + 2s_i \cdot A_{FF})N + A_{treeRNS} + A_{mod-red}\ ] + A_{conv} \tag{3.13}$$

where

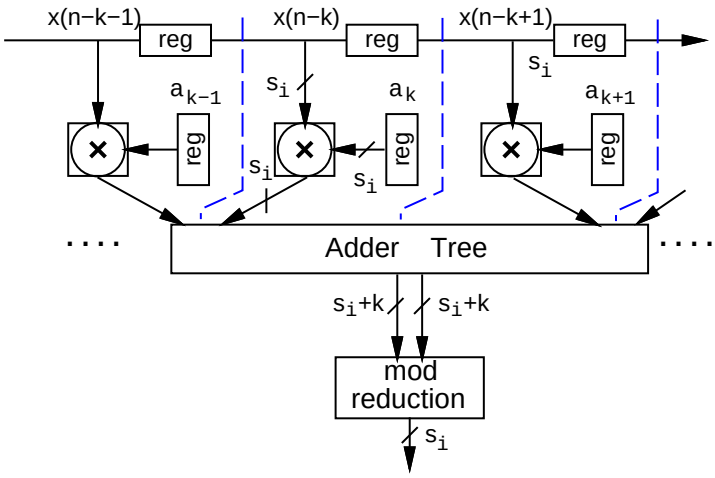


Figure 3.9: RNS FIR filter in direct form.

$A_{treeRNS} = (s_i + k)A_{CSA-4:2} \cdot (\frac{N}{2} - 1)$

A_{conv} is the area of the in/out converters.

Similarly to the TCS case, also for the RNS, the power dissipation expressions are not very accurate for filters in direct form. Therefore, they are omitted.

3.3.3 Direct FIR Filters: Summary

Table 3.2 summarizes the results for the different FIR in direct form implementations. The critical path depends on the number of levels in the tree (and consequently, on the number of taps). We can introduce pipeline latches to speed-up operations, but we increase area and power. For the error-free with variable coefficients filters the critical path of RNS and TCS filter is almost the same, but the RNS is smaller and consumes less power for $N > 8$. For the error-free with constant coefficients filters, the RNS is faster (about 1.7 ns less), and its area start to be smaller for more than 16-tap filters. The worst break-even N is for constant coefficients TCS truncated vs. RNS ($N^* = 97$). The trends for the delay of the critical path and the area as a function of N are shown in Figure 3.10.

3.4 RNS Coding Overhead

In this section we address the overhead due to the coding of the RNS base with respect to the filter dynamic range, and delay-area tradeoffs. The Design Space Exploration (DSE) and its results, are helpful in evaluating the effects of the RNS coding overhead and to choose an efficient filter architecture trading-off filter order, dynamic range, clock frequency and area.

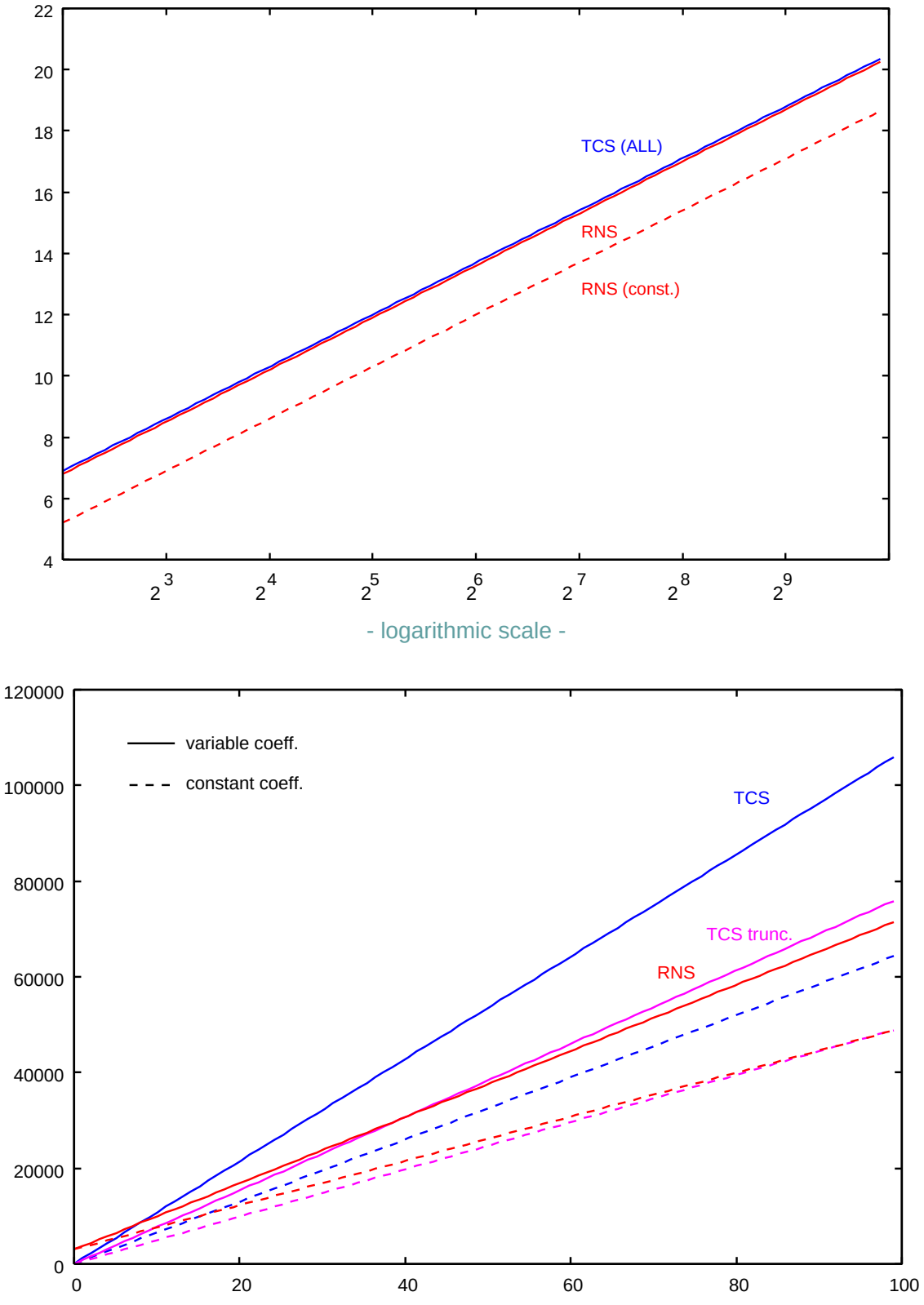


Figure 3.10: Trends for direct FIR filters: delay critical path (top) and area (bottom).

Filter 20-bit dyn. range	TCS			
	Crit. path $[ns]$	Area (NAND2 equiv.)	N^*	Power [1] $[mW]$
var. coeff.	$3.5 + 1.7k$	$1069N + 71$	8	81.4
const. coeff.[2]	$3.5 + 1.7k$	$649N + 71$	16	68.8
TCS truncated				
var. coeff.	$3.5 + 1.7k$	$766N + 56$	40	70.1
const. coeff.[2]	$3.5 + 1.7k$	$493N + 56$	97	51.7

Filter 20-bit dyn. range	RNS		
	Cycle $[ns]$	Area (NAND2 equiv.)	Power [1] $[mW]$
var. coeff.	$5.1 + 1.7(k-1)$	$691N + 3045$	80.6
const. coeff.[2]	$3.5 + 1.7(k-1)$	$462N + 3045$	73.5

[1] Power at 100MHz assuming random input activity.
[2] Assuming 50% of bits set at 1 on average.

Table 3.2: FIR filters in direct form: summary of results.

3.4.1 Coding Overhead

By defining D the dynamic range of an application, which corresponds to d bits ($D = 2^d$) in the two's complement number system (TCS), the RNS base is chosen such that

$$\prod_{i=1}^{P} m_i = M \geq D = 2^d \ .$$

Because each modulus of the RNS base is encoded in binary for a total number of bits

$$b = \sum_{i=1}^{P} \lceil \log_2 m_i \rceil \ ,$$

we define $\mathrm{OH} = b - d$ as the *RNS coding overhead*.

The effect of this overhead is investigated as a function of the dynamic range in the implementation of FIR filter architectures. Specifically, we evaluate all combinations of P prime moduli covering the required dynamic range, select the suitable moduli set with different criteria (see Section 3.4.2), and compare in terms of delay and area the RNS implementation of the filter with the TCS implementation. We do not consider power-of-two moduli ($m_i = 2^k$) because they do not add coding overhead with respect to the TCS.

3.4.2 Design Space Exploration

The design space exploration is carried out to identify the tradeoffs between filters realized in TCS and RNS with respect to dynamic range, clock frequency (throughput) and area.

We use the architectures presented in the previous sections for TCS and RNS filters in direct and transposed form. For filters in direct form, the size and depth of the adder tree depend on the filter order N. For this reason, we have characterized TCS and RNS direct form filters of the same order (number of taps).

In the DSE, we run the following *experiments*:

EXP-1 : In this experiment, we consider the RNS moduli selected to obtain the *"worst case"* coding overhead (OH) conditions for the RNS filters.

EXP-2 : In this experiment, we consider the RNS moduli selected to obtain the best tradeoff delay-area for the given dynamic range.

EXP-3 : The moduli are chosen as in EXP-2, but the evaluation is carried on a complete N-tap filter.

In EXP-1 and EXP-2, we do not include the TCS/RNS/TCS conversion and consider the area value per tap, while in EXP-3 we take into account the impact of the converters on FIR filters of the same order N.

The first two experiments are based on the characterization of the filter composing blocks (multipliers, adders, registers, etc.) performed by Synopsys Design Compiler and the STM 90 nm library of standard cells [38]. The results of EXP-3 are obtained by synthesis of the actual filters.

EXP-1: Worst Case OH

In this experiment, the choice of moduli (RNS base) is done by selecting the set that for the given dynamic range has the largest RNS coding overhead $OH = b - d$. This is done in two steps:

1. The group of RNS bases for which $M \geq 2^d$ are first selected;
2. The base which has the largest b is selected. In case of tie, the set with the largest number of moduli P is chosen.

The results of this selection are shown in Table 3.3.

In EXP-1, we consider a very large clock period so that the evaluation is based on the results of the characterization of circuits optimized for minimum area.

Because for the RNS bases of Table 3.3 there is not a significant difference in area per tap between the filters in direct and transposed form, in Figure 3.11 we

d	**Moduli Set**	b	$b-d$
12	{ 5, 7, 11, 17 }	15	3
16	{ 5, 7, 11, 17, 19 }	20	4
20	{ 3, 5, 11, 17, 19, 37 }	25	5
24	{ 3, 5, 11, 13, 17, 19, 37 }	29	5
28	{ 3, 5, 11, 13, 17, 19, 23, 29 }	33	5
32	{ 3, 5, 7, 11, 13, 17, 19, 37, 41 }	38	6
36	{ 3, 5, 7, 11, 13, 17, 19, 23, 29, 37 }	42	6
40	{ 3, 5, 11, 13, 17, 19, 37, 41, 43, 47 }	47	7
44	{ 3, 5, 11, 13, 17, 19, 23, 29, 37, 41, 43 }	51	7
48	{ 3, 5, 7, 11, 17, 19, 23, 29, 31, 37, 41, 43 }	55	7

Table 3.3: Moduli set and dynamic range for EXP-1.

plot the curves[1] of direct and transposed TCS (D-TCS and T-TCS) and just one curve for the RNS filters. The area unit is 1 μm^2.

Figure 3.11 show that for dynamic ranges d from 12 to 24, the area is roughly the same for the four architectures (D-TCS is slightly better), and that for $d > 24$ bits RNS filters are significantly smaller than TCS.

EXP-2: Best Delay-Area Tradeoff

The results of EXP-1 are obtained for the worst case choice of the RNS base and considering a very long clock period. In this experiment, we make more realistic assumptions by introducing a timing constraint on the clock frequency (500 MHz) and by selecting the moduli according not to the worst case OH, but to the results of the characterization of delay/power/area done with a RNS filter design tool [39]. Given a target clock period, the tool selects the RNS base which guarantees the smallest area (or lower power dissipation). The selected RNS bases are reported in Table 3.4.

For filters in direct form (see Figure 3.8 and Figure 3.9) the timing constraint might cause modifications in the filter architecture when for deep adder trees (large N) it might be necessary to break the critical path and introduce pipeline registers. For this reason, we limit EXP-2 to the case of filters in transposed form, where the critical path is independent[2] of the number of taps (see Figure 3.4 and Figure 3.6).

The results of the experiment are plotted in Figure 3.12. The dotted curves

[1] Actually the curve is a visual aid to better display the distribution of the experiment's discrete points.

[2] We do not consider the increased buffering on x for large N in this experiment.

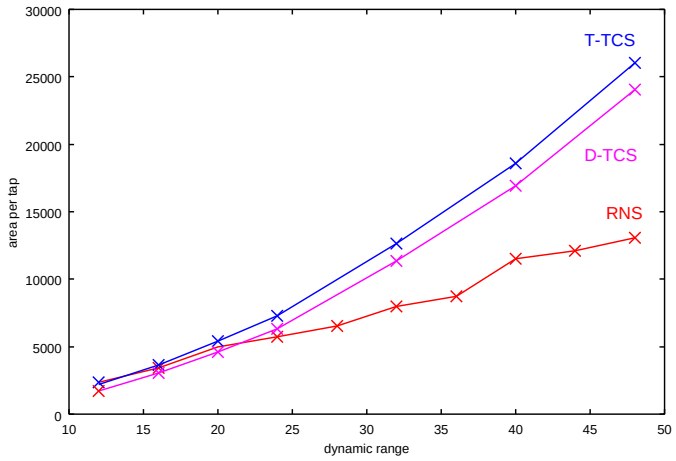


Figure 3.11: Results of EXP-1: direct and transposed TCS vs. RNS

refer to the corresponding results of EXP-1 (Figure 3.11). The figure clearly shows that, when a timing constraint ($T_C = 2.0\ ns$) is introduced, the taps in the TCS filter grow larger (tradeoff speed-area), while for the RNS this growth is compensated by a choice of the moduli set which minimizes the area. The difference is larger as the dynamic range increases. For $d = 48$ the area of the TCS is double than the RNS filter.

Moreover, by tracing horizontal lines in Figure 3.12, we can roughly determine the truncated TCS tap equivalent to the error-free RNS tap. For example, for $d = 40$ the RNS area per tap is $10,000\ \mu m^2$. This area is equivalent to that of a truncated TCS with dynamic range $d = 24$, that is a TCS filter with error $\geq 2^{16}$ unit-in-last-position with respect to the RNS.

EXP-3: Complete N-order Filter

In this experiment we validate the results of the DSE with an actual implementation to see the impact of the TCS/RNS and RNS/TCS converters.

From Figure 3.12 we select $d = 16$ and determine the filter order N for which the RNS has a smaller area than the TCS filter. We implement 10, 50 and 100-tap complete filters in both TCS and RNS and plot the data-points, together with their trends in Figure 3.13. From the figure we can see that for $N^\star > 12$ taps the RNS filter is smaller than the TCS filter.

d	**Moduli Set**	b	$b-d$
12	{ 3, 7, 13, 17 }	14	2
16	{ 5, 7, 11, 13, 17 }	19	3
20	{ 3, 7, 11, 13, 17, 23 }	23	3
24	{ 3, 7, 11, 13, 17, 19, 23 }	28	4
28	{ 3, 5, 7, 13, 17, 19, 23, 31 }	32	4
32	{ 3, 5, 7, 11, 13, 17, 19, 29, 31 }	36	4
36	{ 5, 7, 11, 13, 17, 23, 29, 31, 41 }	40	4
40	{ 11, 13, 17, 19, 23, 29, 31, 37, 41 }	45	5
44	{ 17, 19, 23, 29, 31, 37, 41, 43, 47 }	49	5
48	{ 23, 29, 31, 37, 43, 47, 53, 59, 61 }	51	3

Table 3.4: Moduli set and dynamic range for EXP-2.

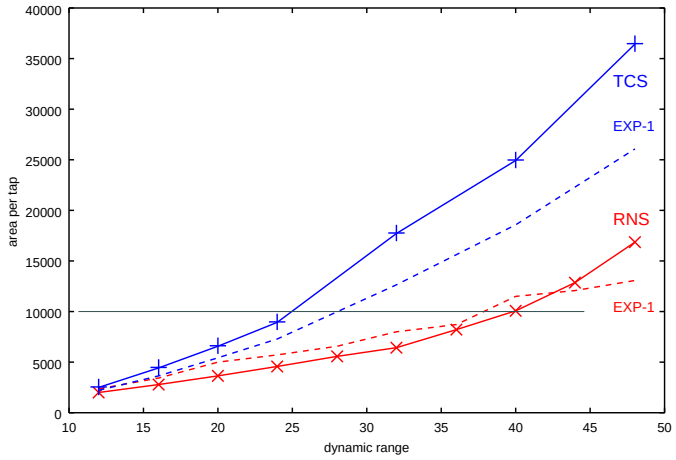


Figure 3.12: Results of EXP-2: transposed TCS vs. RNS

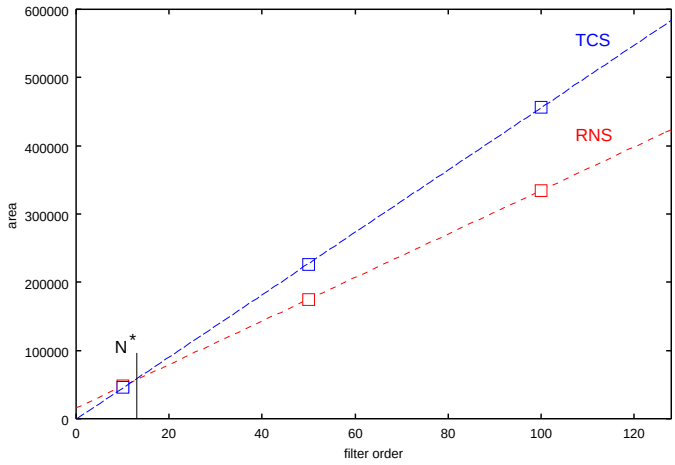


Figure 3.13: Results of EXP-3: complete filter TCS vs. RNS

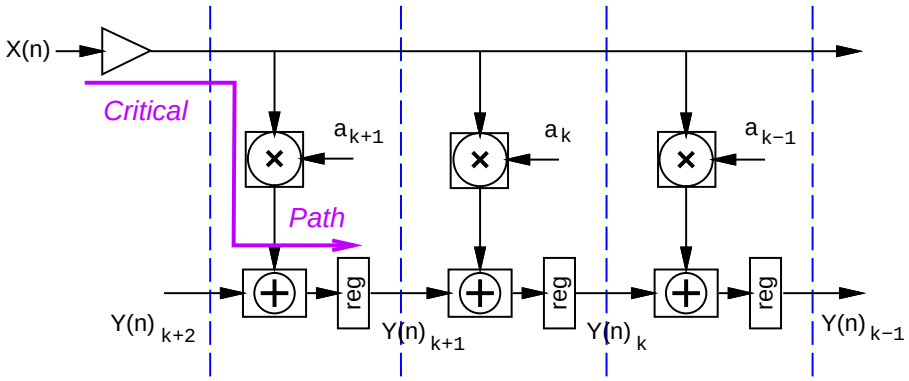


Figure 3.14: Structure of FIR filter in transposed form and its critical path.

3.5 Carry-Save RNS Filter

The delay of the critical path for a FIR filter mod m_i in transposed form is given in (3.6) and graphically shown in Figure 3.14. To speed-up the operations by making the clock period shorter, we can resort to a carry-save representation for the binary representation of residues (ys_k, yc_k) and avoid to compute the modular addition in every tap. The operands to be added in a tap are three: the product $p_k = \langle a_k x(n-k) \rangle_{m_i}$ and the carry-save representation of y_{k-1}

$$ys_k = sum(ys_{k-1}, yc_{k-1}, p_k),$$
$$yc_k = carry(ys_{k-1}, yc_{k-1}, p_k).$$

The structure of the carry-save RNS tap is shown in Figure 3.15. With the new

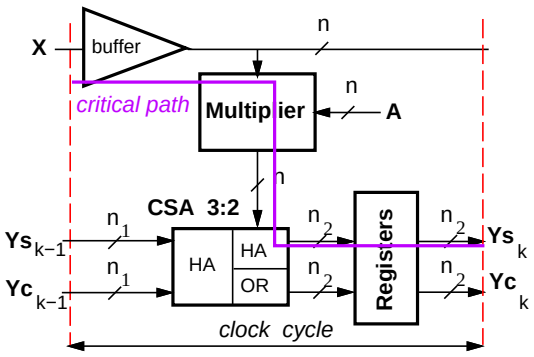


Figure 3.15: Tap structure for RNS carry-save.

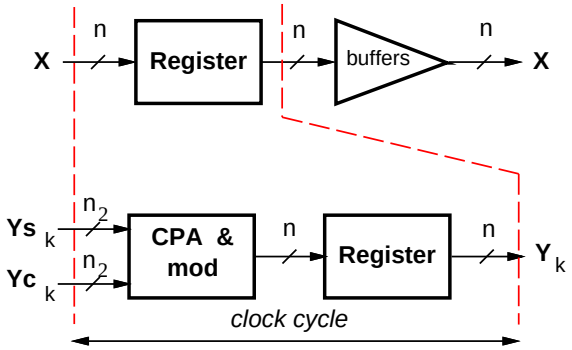


Figure 3.16: Relay station.

representation, we reduce the term t_{modADD} of (3.6) to the delay of a half-adder and the critical path is now:

$$t_{CS-RNS} = t_{buffer} + t_{modMULT} + t_{XOR} + t_{REG} \tag{3.14}$$

The critical path is essentially determined by the multiplier latency, being t_{buffer} and t_{REG} unavoidable, and being t_{XOR} the minimum delay attainable for a half-adder.

However, the CS-representation implies the doubling of the registers, and, as the number of taps increases, a logarithmic increase in the bit-width of CSAs and registers. For this reason, it might be convenient to insert some *relay stations* (RS), which assimilate the carry-save representation of y_k and extract its modulo m_i, to prevent the bit-width from growing too much. Relay stations are depicted in Figure 3.16 and their delay is

$$t_{RS} = t_{cpa\&mod} + t_{REG} \tag{3.15}$$

They introduce an extra cycle of latency, but they can also be used to better dimension the buffering of x (i.e. reduce t_{buffer}).

The spacing of relay stations (i.e. the number n of taps between two relay stations) can be determined by combining (3.14), in which the term t_{buffer} increases with n, and (3.15), where $t_{cpa\&mod}$ depends on n.

The implementation in carry-save RNS (CS-RNS) is compared with TCS and RNS in Section (3.6.3). In that implementation, the relay stations are spaced by 8 taps, because this configuration gives the best delay-area tradeoff for the technology and the set of moduli used.

3.6 Low Power RNS Implementation

In addition to the above presented implementations, we take advantage of the structure of the RNS filters (P parallel channels) to reduce its power dissipation without penalizing its speed.

3.6.1 Multi-Voltage RNS Filter

The power dissipated in a cell depends on the square of the supply voltage (V_{DD}) so that a significant amount of energy can be saved by reducing this voltage [37]. However, by lowering the voltage the delay increases, so that to maintain the performance this technique is applied only to cells not in the critical path.

Table 3.5 reports details on the implementation of one RNS filter for the different paths corresponding to the moduli m_i. Delay is normalized to the critical path (i.e. clock cycle), while area and power dissipation are normalized to their totals.

Modulus	Delay	Area	Power
3	0.35	0.05	0.04
5	0.70	0.10	0.10
7	0.75	0.13	0.12
11	1.00	0.24	0.25
17	1.00	0.23	0.26
64	0.65	0.25	0.24
total	-	1.00	1.00

Table 3.5: Delay, area and power dissipation per tap in RNS FIR.

Because the single tap delay of moduli 3, 5, 7 and 64 is less than the critical path, we can use the available time slack and reduce the supply voltage for these moduli without affecting the overall performance.

The library of standard cells we used for this example normally operates at $V_{DD} = 3.3\ V$. In Table 3.6 we report the possible supply voltage which can be used

in the modulo m_i filters without increasing the critical path. Table 3.6 also reports the power savings obtained with the listed supply voltage. These values have been computed assuming that the switching activity does not change when scaling the voltage. This assumption seems to be reasonable because an increased short-circuit energy, due to longer transition times, is compensated by a suppression of some glitches, due to a longer gate delay. The table shows a reduction of about 15% in the power dissipation per tap.

	single V_{DD}			multiple voltage		
Modulus	Delay	V_{DD}	Power	Delay	V_{DD}	Power
3	0.35	3.3 V	0.04	1.00	1.7 V	0.01
5	0.70	3.3 V	0.10	1.00	2.7 V	0.07
7	0.75	3.3 V	0.12	1.00	3.0 V	0.10
11	1.00	3.3 V	0.25	1.00	3.3 V	0.25
17	1.00	3.3 V	0.26	1.00	3.3 V	0.26
64	0.65	3.3 V	0.24	1.00	2.7 V	0.16
total power			1.00			0.85

Table 3.6: Power dissipation for multiple supply voltage in tap.

The use of a multiple supply voltage requires level-shifting circuitry when going from the lower voltage to the higher one [40]. In our case, voltage level shifters are only required for a few bits before the output conversion stage.

3.6.2 Low Leakage Dual Threshold Voltage RNS Filter

With the technology scaling, and the increased transistor's leakage due to sub-threshold currents, also the static power dissipation starts to play an important role in today's power budgets. Moreover, the increasing smaller CMOS transistors allow the hardware implementation of extra functions that before were executed in software, and the migration of complex system to portable devices. Because of the implementation of digital filters in ultra low power processors, such as the one used in tiny systems with limited available power, it is important the static power due to leakage is characterized and, possibly, reduced.

To have an idea of the impact of the device's leakage on power dissipation, we implemented a multiplier, which is the basic block of a FIR filter, in a 0.18 μm, a 0.12 μm and in a 90 nm library. We used the same timing constraint, the delay of 25 inverters with fanout of 4 (a standard measure of delay across different technologies) in their respective libraries. The results, shown in Table 3.7, indicate that the power dissipation due to leakage P_{stat} increases both in absolute value and as the percentage of the overall power dissipation P_{TOT}. By comparing the 0.18 μm and the 90 nm multipliers, we notice that P_{TOT} has decreased of about 70% (mostly due to the scaling of V_{DD}), but the static part P_{stat} has increased

	P_{stat}	P_{tot}	$P_{stat}/P_{tot} \times 100$
mult (180 nm)	0.25	945	0.03
mult (120 nm)	2.25	450	0.50
mult (90 nm)	3.59	299	1.20
P_{90nm}/P_{180nm}	14.36	0.32	40.0

Power dissipation in μW at 100MHz.

Table 3.7: Impact of leakage on technology scaling.

14 times and its contribution to the total 40 times. Moreover, for the 90 nm implementation, if the multiplier is used as often as 1% of the processor usage time the static power dissipation becomes dominant. Therefore, the design of systems in nanometer technologies must take into account methodologies to reduce the static power dissipation.

The standard cell library that has been used in this experiment provides two classes of cells: cells with devices with a reduced threshold voltage (V_t) to achieve High Speed, identified in the following as HS, and cells with devices with a higher V_t to provide Low Leakage identified as LL [38].

Moreover, we consider the cells operating at the typical conditions with a power supply $V_{DD} = 1.0\ V$ and a temperature of 25 C.

By comparing the data-book for the two classes of cells HS and LL, the following points emerge when comparing the same cell (e.g. a NOT gate):

- The HS cell is faster than the corresponding LL cell. For the 1X drive NOT gate loaded with 4 SL (library standard load) the HS gate is about 30% faster than the LL one.
- The area is the same.
- The input capacitance is slightly smaller for the LL cells: For the 1X drive NOT gate, the input capacitance in the LL cell is 10% smaller than in the HS cell.

The total power dissipation for a CMOS gate is

$$P_{TOT} = P_{load} + P_{sc} + P_{leak} \tag{3.16}$$

The term P_{load} is the power dissipated for charging and discharging the capacitive load C_L when the output toggles at a rate f_p [41]

$$P_{load} = C_L V_{DD}^2 f_p\ . \tag{3.17}$$

Therefore, HS and LL cells loaded with the same C_L and with the same switching activity consumes the same P_{load}. However, for clusters of cells of the same type

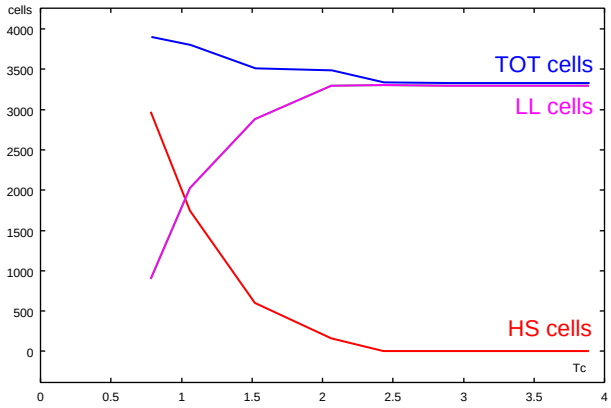


Figure 3.17: The HS and LL cells mix depends on the synthesis timing constraint.

(due to the reduced input capacitance) the LL cells show a lower P_{load}, if the activity is the same. For a chain of inverters toggling at the same rate, the HS cells dissipate about 5% more than the LL cells.

The power due to short circuit currents P_{sc} is

$$P_{sc} = \frac{\beta}{12}(V_{DD} - 2V_t)^3 \frac{t_{rf}}{t_p} \tag{3.18}$$

where $\beta_{HS} \geq \beta_{LL}$, t_{rf} is the average rise and fall time and $t_p = 1/f_p$ [41]. From (3.18), it is clear that for the LL cells P_{sc} is lower than for HS cells, although, due to the longer transition times, the term t_{rf} is higher.

The term P_{leak} is the static power dissipation

$$P_{leak} = V_{DD} I_{sat} \tag{3.19}$$

where I_{sat} is the reverse saturation current. This contribution is independent of the switching activity, but depends on the state (logic level of the inputs) of the gate. The LL cells are designed to have a P_{leak} several times smaller than the corresponding HS cells.

In summary, LL cells are slower than HS cells, but dissipate less dynamic (if the switching activity is the same) and static power than the corresponding HS cells.

The current version of Synopsys Design Compiler [42] can handle the synthesis of dual V_t standard cell libraries such as the one described above. The prioritized design constraint is the delay (or better the clock period for a synchronous sequential system), but the tool keeps the power dissipation down by substituting HS cells with LL when there is a sufficient time slack. Moreover, the dynamic power dissipation is optimized as indicated in [43].

Figure 3.17 shows the variations in the HS and LL cell mix for a system synthesized with different values of the timing constraint T_C. In the circuit synthesized with the smallest T_C (minimum delay), the number of HS cells is dominant over the LL cells. For the circuit synthesized with a longer T_C (right side of Figure 3.17), all cells are of LL type to have a reduced power dissipation.

3.6.3 Results of Implementations

In implementing these low power filters, we take advantage of state-of-the-art design automation tools [42] which handle libraries of standard cells with dual threshold transistors [38].

Multi-Voltage Implementation

First we compare in terms of power dissipation the following four schemes:

- TCS FIR filter in transposed form (Figure 3.4)
- RNS FIR filter in transposed form (Figure 3.6)
- RNS filter implemented in carry-save (CS-RNS)
- RNS filter implemented with multiple supply voltages (MV-RNS)

The filters are implemented in a $0.35\mu m$ library of standard cells. The results are shown in Figure 3.18 with the plots of the power dissipation vs. N. In the table, area is reported as number of NAND2 equivalent gates and power is computed at 100 MHz.

The table shows that the RNS filter can sustain the same throughput as the TCS one, but dissipates less power when the filter has order N larger than 4. Furthermore, the CS-RNS implementation of the filter can be clocked almost at double speed, with respect to the traditional one, without a significant increase in area and energy dissipated in a cycle (the power increases linearly with frequency). Finally, a possible implementation of the RNS filter with multiple supply voltage can lead to a further reduction of the power dissipated, without affecting the performance of the filter.

Dual-V_t Implementation

Because each logic function can be implemented with a HS or a LL cell, the first idea is to replace faster and power hungrier HS with LL cells when possible. By the RNS decomposition of Figure 3.3, the filter is divided into as many independent clusters of cells as the RNS moduli. Because of the different size of the moduli,

Filter	Cycle $[ns]$	Area (NAND2 equiv.)	N^*	Power @100 MHz $[mW]$	N^*
TCS	5.0	200+1400N	-	6.0 + 15.5N	-
RNS	5.0	3250+920N	8	25.0 + 7.8N	4
CS-RNS	3.0	3470+1100N	12	27.4 + 10.5N	6
MV-RNS	5.0	$\sim$ 3250+920N		(est.) 25.0 + 6.6N	3

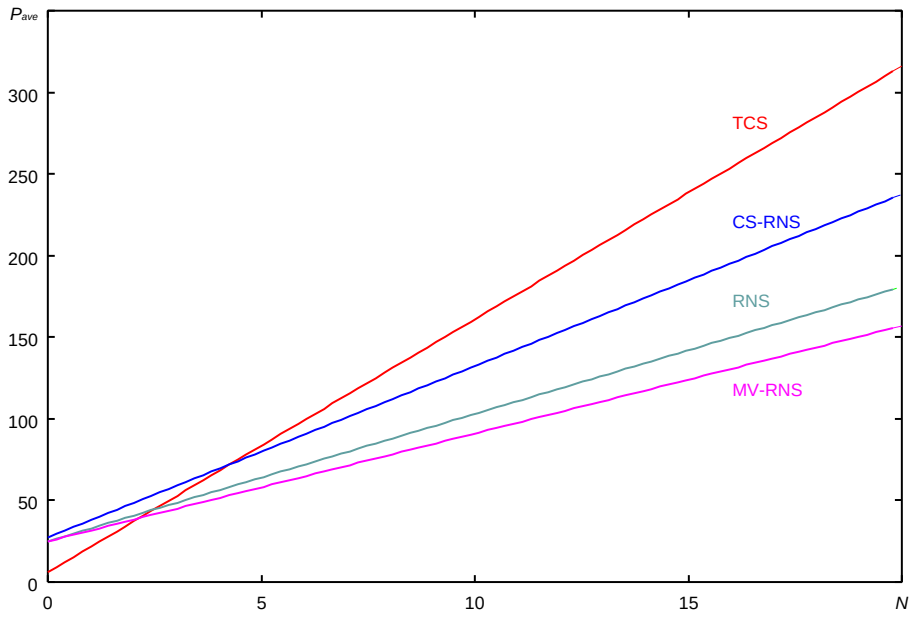


Figure 3.18: Summary of results for low power filters.

blocks	**TCS**			**RNS**		
n-taps	P_{stat}	P_{dyn}	P_{TOT}	P_{stat}	P_{dyn}	P_{TOT}
FIR 16-tap	60.5	6255.2	6396.5	32.4	4434.5	4466.9
FIR 32-tap	111.7	12448.3	12560.0	55.6	8206.8	8262.4
FIR 64-tap	214.5	23341.0	23554.5	105.3	15186.2	15291.5
FIR N-tap	9.1+	212.2+	221.8+	7.5+	944.0+	951.5+
	$3.21N$	$378.4N$	$381.5N$	$1.53N$	$223.2N$	$224.7N$
slope	3.21	378.4	381.5	1.53	223.2	224.7
ratio	1.00	1.00	1.00	**0.50**	**0.60**	**0.60**

Power dissipation in μW at 100MHz.

Table 3.8: Filter dual-V_t implementations: results.

the clusters have different maximum delays. The available time slack in the faster clusters (smaller moduli) allows to exchange HS with LL cells and reduce both the dynamic and static power dissipation. This is similar to what is done in [44] in the dual voltage approach.

In order to compare the power dissipation of the filters, we have implemented a 16, 32 and 64-tap error-free programmable FIR filter (20 bits dynamic range, transposed form) in the traditional two's complement system (TCS) and in RNS.

The comparison is carried out on filters implemented in the 90 nm STM library of standard cells ($V_{DD} = 1.0\ V$, at 25 C) [38], and the power dissipation has been computed by Synopsys Power Analyzer based on the annotated switching activity of random generated inputs. All the filters can be clocked at $f_{max} = 500\ MHz$. Table 3.8 (upper part) summarizes the results for the implemented filters. The power dissipation is computed at a clock frequency of 100 MHz.

By interpolating the results for static, dynamic and total power dissipation, we obtain expressions of the power as a function of the filter order N (Table 3.8, lower part). These trends are also plotted in Figure 3.19. The slopes $\frac{P}{\text{tap}}$ indicated in Table 3.8 (lower part) represent the average power dissipated per tap.

From these results, it is clear that the RNS decomposition in parallel paths allows a reduction per tap of 40% for the dynamic and 50% for the static power without delay penalty (throughput).

3.7 Complex FIR Filters

Due to the high number of multiplications needed in the case of the implementation of a complex filter (for a complex product we need four real multiplications and

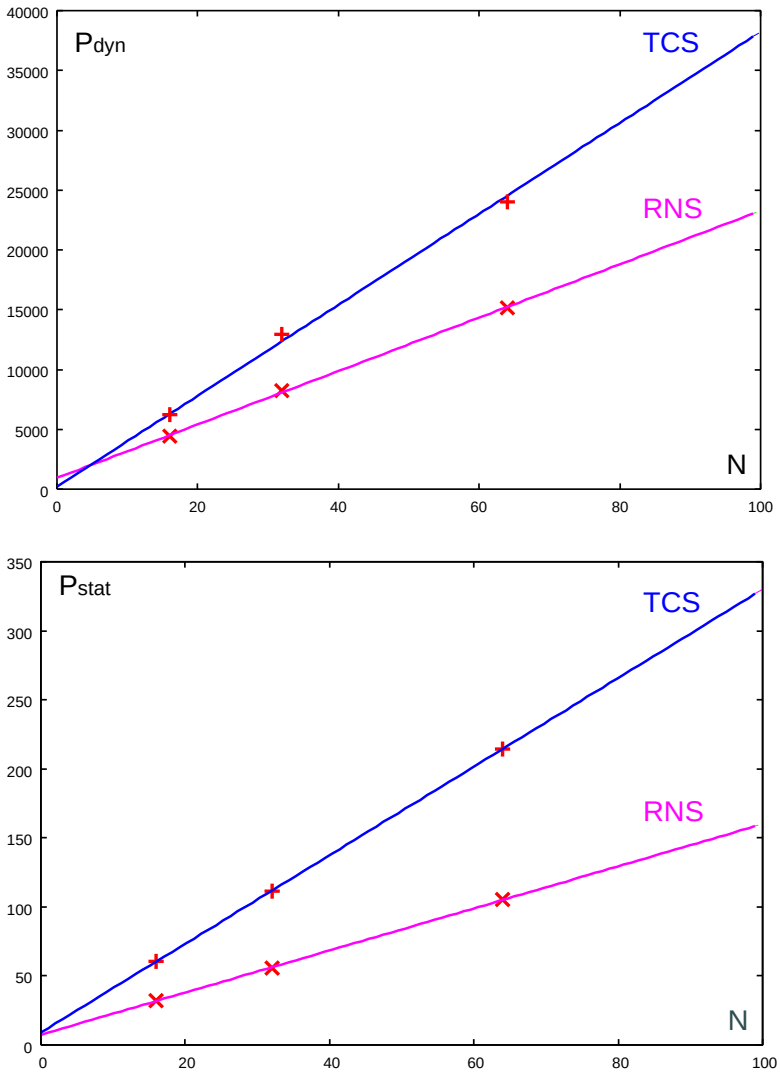


Figure 3.19: Dynamic (top) and static (bottom) power dissipation (TCS vs. RNS).

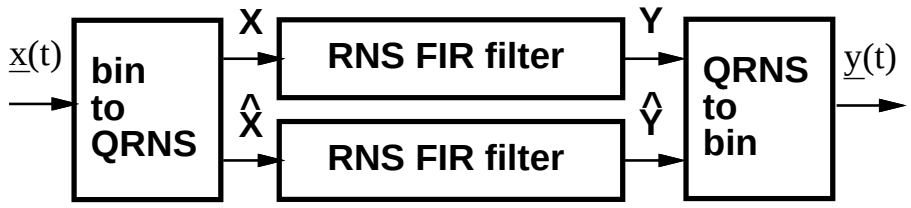


Figure 3.20: Structure of QRNS filter.

two additions, and if we use the Golub'a rule [45], three multiplications and five additions), the use of the RNS and in particular of the isomorphism technique appear to be very promising. In this section of the report, some results of the comparison of a TCS and a Quadratic RNS (QRNS) implementation are presented.

The QRNS property of (1.7), described in Section 1.6, can be extremely useful in applications characterized by intensive computations on complex numbers such as for example complex FIR filters. A complex N taps FIR filter is expressed by

$$\underline{y}(n) = \sum_{k=0}^{N-1} \underline{a}_k \underline{x}(n-k)$$

where $\underline{x}, \underline{y}, \underline{a}_k$ denotes complex quantities. From (1.7), it is easy to derive for the complex filter the structure shown in Figure 3.20, in which both portions of the filter are realized with P RNS filters working in parallel as for the case of real filters.

The modular multiplications and additions, and the input and output conversions in the QRNS filter are implemented with the architectures shown in Chapter 2.

3.7.1 Comparison TCS vs. QRNS

The filter implemented in this example, is a complex 64-taps FIR filter with a dynamic range of 20 bits. For this dynamic range, we can choose the following QRNS base:

$$m_i = \{5, 13, 17, 29, 41\}$$

such that

$$log_2(5 \cdot 13 \cdot 17 \cdot 29 \cdot 41) > 20 \ .$$

The tap structure of the complex TCS filter is sketched in Figure 3.21. The mod m_i filters are implemented as shown in Figure 3.9 (direct form).

Both the TCS and the QRNS filters are implemented in a $0.35\mu m$ standard cells library. Delay, area and power dissipation have been determined with Synopsys tools.

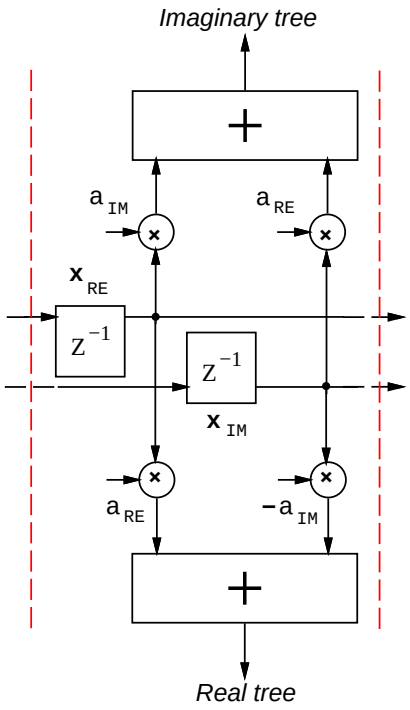


Figure 3.21: Structure of tap in complex TCS FIR filter.

Table 3.9 summarizes the results. In the table, area is reported as number of NAND2 equivalent gates and power is computed at 166 MHz.

Table 3.9 shows that the QRNS filter has a higher latency, due to the conversions, but it can be clocked at the same rate of the traditional filter, and consequently, it can sustain the same throughput. However, the QRNS filter is almost half the area on the traditional complex filter, and consumes one third of the energy.

The results obtained show that the QRNS filter can sustain the same clock rate, although it has a slightly longer latency. However, in terms of area and power the QRNS version is more convenient. Similarly to real filters, a better improvement is expected for filters with a larger number of taps.

Filter	Cycle $[ns]$	Latency (cycles)	Area (gate equiv.)	Power $[W]$
QRNS	6.0	$11 + 64$	$182,400$	2.5
Trad.	6.0	$6 + 64$	$315,700$	7.4
ratio	1.0	1.07	0.57	0.34

Table 3.9: Complex filters: summary of results.

3.8 FPGA Implementation

In this section the result of the implementation of TCS and RNS filters in Field Programmable Gate Arrays (FPGAs) is presented. Moreover, power consumption estimations based on the measurement of the average current absorption in the experimental set-up has been obtained. A comparisons of the results by experiments on FPGA with the results previously obtained for standard cells implementations in Section 3.2.4 are also illustrated.

3.8.1 The Experimental Set-Up

The experimental set-up used for the power consumption evaluation is shown in Figure 3.22. It is based on the following instruments and board:

FPGA board: Xilinx AFXPQ240-110 development board equipped with a Virtex V600E HQ240 FPGA [46]. Separate power supplies for the FPGA core (V_{CCINT}) and for the I/O banks (V_{CCO}) are available so power consumption measurements of the core and I/O banks can be easily carried out. A picture of the Xilinx development board is shown in Figure 3.24.

LSA and Pattern generator: Agilent 16720 pattern generator and logic state analyzer for the stimuli generation and for the evaluation of the filter behaviour.

Multimeter: Agilent 34401 digital multimeter for the mean current measurements.

The main target of the test bed has been the estimation of the core power consumption. Consequently, the mean current has been measured by the digital multimeter. The different FIR filters architectures (low pass frequency mask), have been stimulated by random samples (uniform probability density function) generated from the patter generator.

3.8.2 Results

The main purpose of this implementation is to check if the expressions shown in Figure 3.18 obtained for standard cells are also valid for an FPGA implementation. The power consumption evaluation has been obtained starting from measurements on FIR filters with a given number of taps. From this data an extrapolation of the power consumption for the same filters with any number of taps is obtained. For this reason, measurements have been interpolated to fit the following expression

$$P = P_1 \cdot N + P_0 \tag{3.20}$$

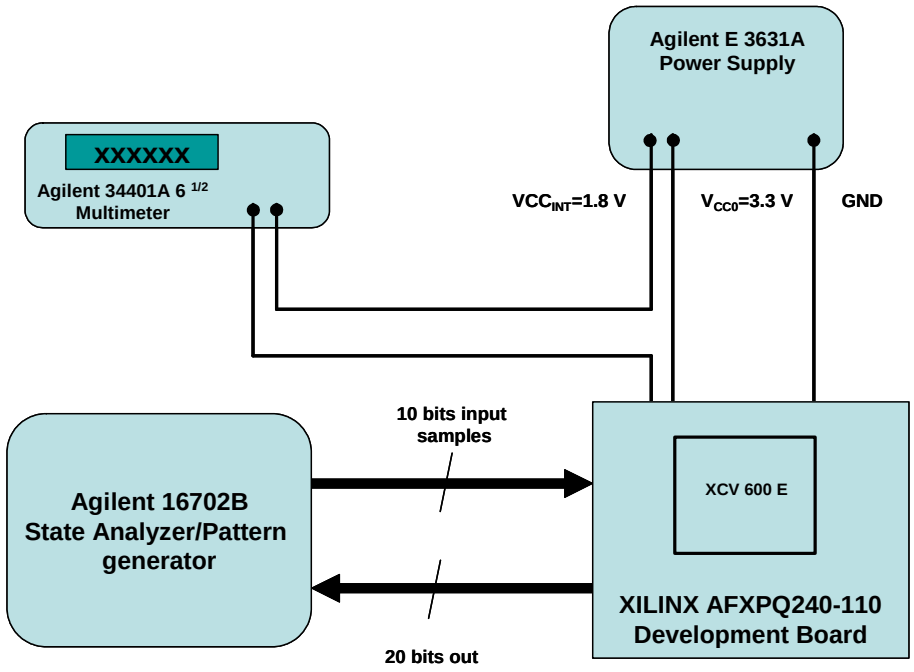


Figure 3.22: FPGA Measurement set-up.

which gives the average power dissipation for a N-tap filter.

The VHDL RT-level code for the different filters, has been synthesized and mapped on the FPGA device by using Xilinx Foundation. Measurements of average current absorption for six filters: 8-tap and 16-taps TCS, 8-taps and 16-taps RNS and 8-taps and 16-taps CS-RNS have been done (20 bits dynamic range). Table 3.10 shows the power consumption measurements at different clock frequencies (T_c). The average power dissipation has been computed from $\overline{P} = V_{CCINT} \cdot \overline{I}$, in which V_{CCINT} is the FPGA core voltage supply and $\overline{I}$ is the measured current. In CMOS technology, the dynamic power consumption scales with frequency as ($f = 1/T_c$), but some values of power consumption in Table 3.10 do not follow this rule due to inaccuracies of the measurement system. To improve the accuracy of the estimations, the power consumption values obtained for different T_c have been converted into energy per clock cycle:

$$E_c = \overline{P} \cdot T_c \qquad [nJ]$$

and then averaged obtaining the values shown in Table 3.10 (average). Therefore, instead of fitting Equation 3.20 the following equation has been fitted

$$E_c(N) = E_1 \cdot N + E_0 \quad . \tag{3.21}$$

In order to better evaluate the term E_0, a circuit composed by the serial

	TCS				RNS				CS-RNS			
	8-tap		16-tap		8-tap		16-tap		8-tap		16-tap	
T_c	$\overline{P}$	E_c	$\overline{P}$	E_c	$\overline{P}$	E_c	$\overline{P}$	E_c	$\overline{P}$	E_c	$\overline{P}$	E_c
$[ns]$	$[mW]$	$[nJ]$	$[mW]$	$[nJ]$	$[mW]$	$[nJ]$	$[mW]$	$[nJ]$	$[mW]$	$[nJ]$	$[mW]$	$[nJ]$
1,000	20.5	20.5	41.2	41.2	12.4	12.4	20.2	20.2	10.6	10.6	16.7	16.7
500	40.7	20.3	81.2	40.6	24.3	12.2	40.3	20.2	21.6	10.8	33.5	16.7
250	80.3	20.0	160.0	40.0	49.1	12.3	81.0	20.3	42.8	10.7	66.6	16.7
200	99.9	19.9	198.5	39.7	60.8	12.2	101.0	20.2	53.3	10.6	83.0	16.6
100	197.6	19.7	387.9	38.8	121.5	12.1	198.7	19.9	105.8	10.6	164.7	16.5
average		20.1		40.1		12.2		20.1		10.7		16.6
# slices	1240		2440		1364		2310		1358		2274	
(% area)	17%		35%		19%		33%		19%		32%	

Table 3.10: Measurements of average power consumption and E_c.

connection of the binary-RNS and RNS-binary converters has beeen mapped and measured obtaining $E_c(0) = 5.8\ nJ$

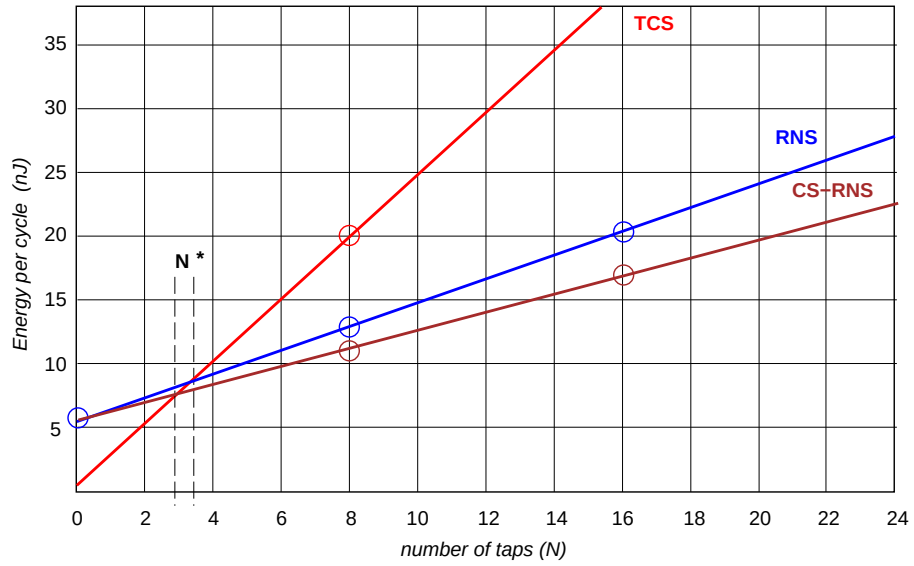


Figure 3.23: Plots of E_{TCS}, E_{RNS} and E_{CSRNS}.

Finally, the values of E_c (for the different filters architectures) has been fitted to obtain the expressions in Table 3.11 that has been plotted in Figure 3.23.

From Figure 3.23 we can see that for filters with more than 4 taps ($N^* = 4$) the RNS filter consumes less power. This result is similar to that of Section 3.6.3 ($N^* = 4$ in Figure 3.18). The value N^* obtained here seems to confirm that, due to the small dynamic range of the residues, RNS has shorter (or at least not longer) interconnections, and routing is more local than in TCS.

The result obtained for the CS-RNS filter is even more interesting: in a stan-

dard cells implementation (Figure 3.18) CS-RNS filters consume more energy than the corresponding plain RNS filters, while in the FPGA implementation CS-RNS filters consume less power.

In carry-save RNS filters, the modular sum

$$s_k = \langle s_{k-1} + a_k x(n-k) \rangle_{m_i}$$

is not done in each tap, but s_k (kept in carry-save format) is reduced modulo m_i every 8 taps.

Because additional registers are required to keep a carry-save representation of s_k-s, there is a tradeoff between combinational logic (adders) and flip-flops.

In standard cells, flip-flops consume considerably more energy than simple logic gates, therefore, the use of a carry-save representation, which replaces adders with registers, leads to an increase in power dissipation. In FPGAs, combinational functions are implemented with LUTs, which apparently consume more energy than flip-flops. Therefore, CS-RNS filters not only are faster than plain RNS (and TCS) filters, but are smaller in terms of area and consume less power.

	$E_c \quad [nJ]$	N^*
TCS	$2.5 \cdot N + 0.2$	-
RNS	$0.9 \cdot N + 5.6$	4
CS-RNS	$0.7 \cdot N + 5.7$	3

Table 3.11: Expressions of E_c for the different filters.

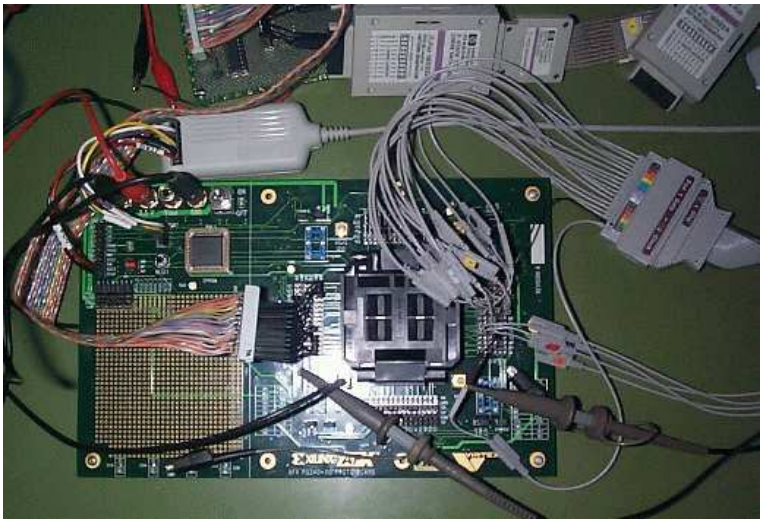


Figure 3.24: Photo of the test bed.

3.9 ASIC vs. FPGA Implementations

In this section, the properties of RNS implementations based on two different technologies: ASIC-SC (Standard Cells) and FPGA are compared in terms of power consumption.
As discussed above, the RNS representation offers great advantages in the implementation of DSP systems. These advantages are mainly related to the absence of carry propagation among modular processors and the possibility to avoid multiplications by using the isomorphism technique. For these reasons the RNS implementations are characterized by

1. structures that are simpler than the TCS ones (suitable for high-speed processing),

2. signals that are more local (signals are bounded inside each modular block)

In this Section, a power consumption model for FIR filter structures is developed. As general remark, we have to take into account that the different implementation technologies impact differently on the power consumption model, that is:

- ASIC-SC (Standard Cells) are characterized by very variable logic and interconnect structures.
- FPGAs, are characterized by a fixed structure based on CLBs, clock and interconnects resources.

Consequently, the evaluation of power consumption for ASIC-SC technologies is, in general, more complex.

In the previous Sections different FIR filtering structures have been compared. In the following analysis, A_x and P_x represent the area and the power consumption of the filter, and x identifies the number system representation used in the implementation namely RNS or TCS. In a FIR filter, A_x and P_x grow linearly with the number of taps (N), as shown in the following equation

$$\begin{aligned} A_x &= k_1^{A_x} + k_2^{A_x} N \\ P_x &= k_1^{P_x} + k_2^{P_x} N \end{aligned} \tag{3.22}$$

In Figure 3.25 plots of the area and power consumption are shown in the case of a ASIC-SC implementation [47]. In this example, FIR filter wordlength in TCS representation is 10 bits for input data (including both input samples and coefficients) and 20 bits for tap outputs. Equivalent wordlength is used in RNS

implementation.
The offset value in RNS implementation is related to the needs of input and output converters.

3.9.1 ASIC-SC: Power Consumption Contributions

The contributions to the dynamic power consumption in an ASIC-SC implementation are highly dependent on the architecture of the digital system considered. In the case of a microprocessor [48] the power consumption contributions are shown in Figure 3.26.a and are categorized in terms of local and global interconnects (both for clock and signals). This picture shows that local interconnects play a fundamental role in power consumption, followed by local and global clock distributions.

The breakdown for the local interconnect power consumption is shown in Figure 3.26.b and it is composed by three terms 1) power for gate charging (requiring about 50% of the overall local power), 2) power for interconnect capacitance charging (about 30%), and 3) power for charging diffusion capacitances (about 20%). Consequently, in ASIC-SC technology, the RNS characteristic can give advantages in terms of:

1. Reduction of complexity (number of gates - area)
2. Reduction of the interconnection capacitances

To evaluate these effects a model that links power consumption to circuit complexity (area) and interconnects *locality* is developed. In this model A is the area in terms of number of (NAND2) equivalent gates and N_L is the number of nodes. At each node i a capacitance $C(i)$ is connected. Considering a constant activity factor α, the power consumption is expressed by

$$P_{tot} = \sum_{i=1}^{N_L} \alpha(i) C(i) F V_{DD}^2 = \alpha C_{tot} F V_{DD}^2 \tag{3.23}$$

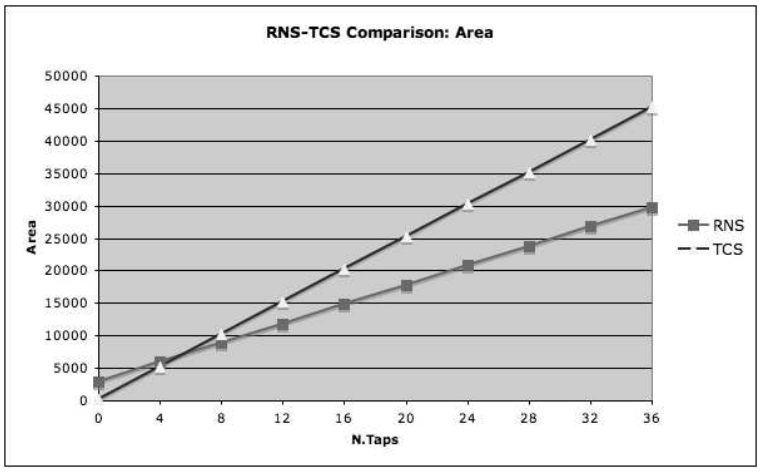
If α, F, and V_{DD}^2 can be considered constant values, we obtain that P_{tot} is proportional to the total capacitance C_{tot}.

In order to better understand the term C_{tot} in Figure 3.27 the contributions to the capacitance of the node i are shown. The node capacitance can be expressed as

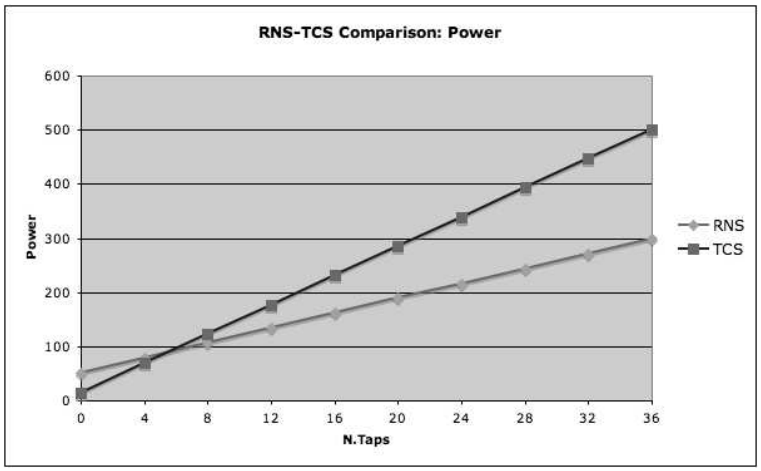
$$C_{node} = C_{line} + C_{FO} = \alpha_C L_{line} + \beta_C FO_{node} \tag{3.24}$$

where the line length L_{line} and the node fan-out FO_{node} contributions are multipled by the coefficients (α_C and β_C). The mean value of this capacitance is

$$C_{mean} = \alpha_C L_{mean} + \beta_C FO_{mean} \tag{3.25}$$

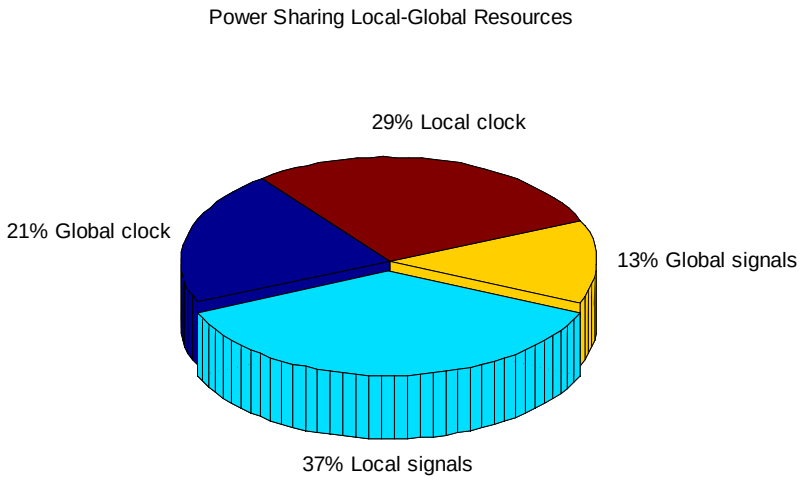


a) RNS vs. TCS: Area comparison

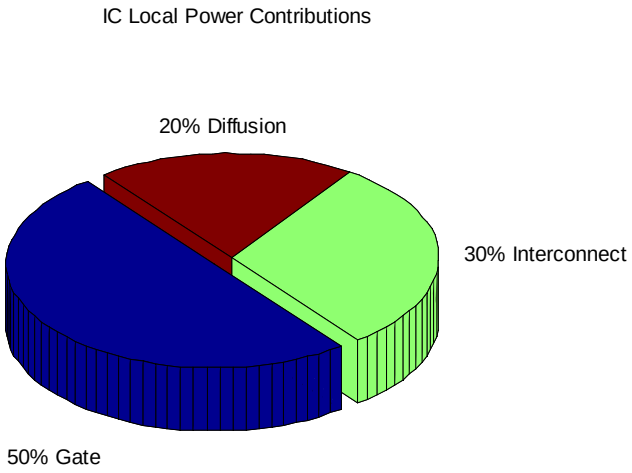


b) RNS vs. TCS: Power comparison (ASIC-SC)

Figure 3.25: Comparisons of TCS and RNS implementations of FIR filters (ASIC-SC)



a) IC power contributions



b) IC local contribution breakdown

Figure 3.26: ASIC-SC power contributions

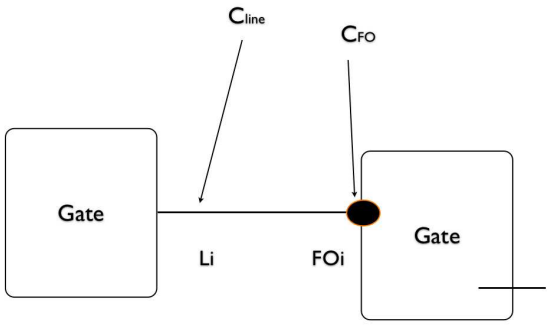


Figure 3.27: Contributions to the node capacitance.

Consequently, the total capacitance is

$$C_{tot} = N_L C_{mean} \tag{3.26}$$

where N_L is the number of nodes of the circuit.

From the above expressions, the following value of power consumption is obtained

$$P_{tot} = \alpha N_L C_{mean} F V_{DD}^2 \tag{3.27}$$

If we consider an increase in the number of gates, the global interconnect capacitance changes for two reasons

1. Increase of the number of nodes (number of outputs).
2. Increase of the interconnect length and Fan Out (corresponding to an increase in signal globality).

In order to link the circuit complexity and global interconnect effects to the capacitance value, we can assume that

1. N_L is proportional to the circuit complexity (number of gates or area A): $N_L = \gamma A$
2. L_{mean} & FO_{mean} are related to the signal locality (through the Globality Index or GI)$L_{mean} = \phi_1 GI; FO_{mean} = \phi_2 GI$

obtaining $C_{tot} = N_L C_{mean} = \gamma A(\alpha_C \phi_1 + \beta_C \phi_2)GI = K \cdot A \cdot GI$ with $K = \gamma(\alpha_C \phi_1 + \beta_C \phi_2)$ and finally

$$P_{tot} = \alpha C_{tot} F V_{DD}^2 = \alpha \cdot K \cdot A \cdot GI \cdot F \cdot V_{DD}^2 \tag{3.28}$$

In this equation P_{tot} the effects of the complexity and of the locality of the interconnect are take into account by A and the Globality Index GI.

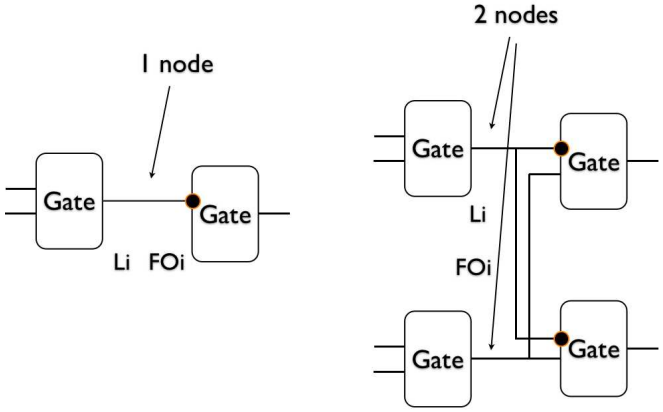


Figure 3.28: Increasing of node capacitances with complexity

Experiment	Description	A_R	P_R	GI_R
1	64-taps transp. FIR	0.630	0.560	0.890
2	8-taps direct FIR	0.994	0.990	0.995
3	64-taps complex FIR	0.578	0.340	0.589
4	Polyphase Filter	0.747	0.624	0.850

Table 3.12: Area, power and globality ratios for the ASIC-SC implementation

3.9.2 Analysis of Experimental Results: ASIC-SC

In this Section eq. 3.28 is used to compare the RNS and TCS implementation of the filters based on ASIC-SC technology obtaining the the power consumption ratio

$$\frac{P_{RNS}}{P_{TCS}} = \frac{\alpha K F V_{DD}^2 A_{RNS} GI_{RNS}}{\alpha K F V_{DD}^2 A_{TCS} GI_{TCS}} \tag{3.29}$$

By solving this equation with respect to $\frac{GI_{RNS}}{GI_{TCS}} = GI_R$ we obtain

$$GI_R = \frac{GI_{RNS}}{GI_{TCS}} = \frac{P_{RNS}}{P_{TCS}} / \frac{A_{RNS}}{A_{TCS}} = \frac{P_R}{A_R} \tag{3.30}$$

The term GI_R can be consequently computed by taking into account that in our experiments the area ratio and the power (P_R) ratios are obtained by simulations. The final results of this evaluation are shown in Table 3.12.

With the exception of the experiment 3, the results show that locality (GI) doesn't play a significant role in power saving for ASIC-SC implementation (GI_R

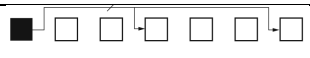
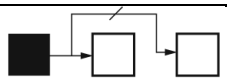
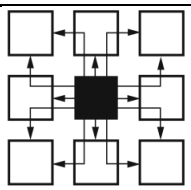
Name	Description	Architecture	Capac.
Long lines	span the full height & width		26.10 pF
Hex lines	route signal to every 3rd or 6th block		18.40 pF
Double lines	route signal to every 1st or 2nd block		13.20 pF
Direct Conn.	route signal to neighboring blocks		v.low

Table 3.13: Interconnects in FPGA.

is close to 1). Experiment 3, based on a Quadratic RNS (RNS) implementation, shows a different behavior. This fact can be related to the activity factor that is noticeably less in the QRNS representation (see [49]).

3.9.3 FPGA: Power Consumption Contributions

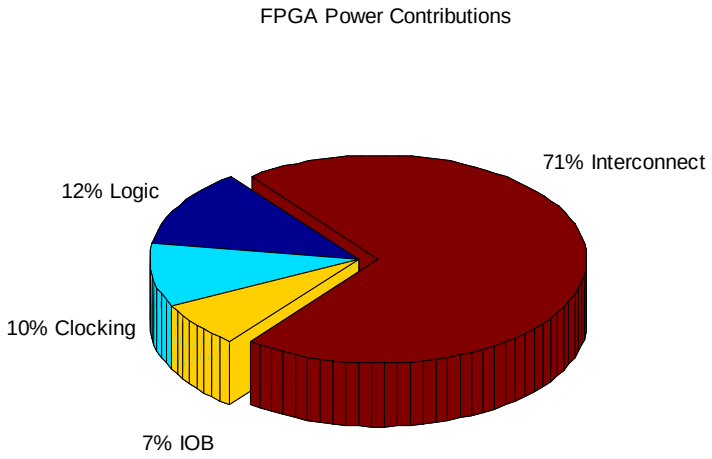
In the case of FPGA implementations the contributions to the power consumption are quite different. It is possible to identify three main contributions ([50]).

1. logic and IOB
2. clocking structures
3. interconnect

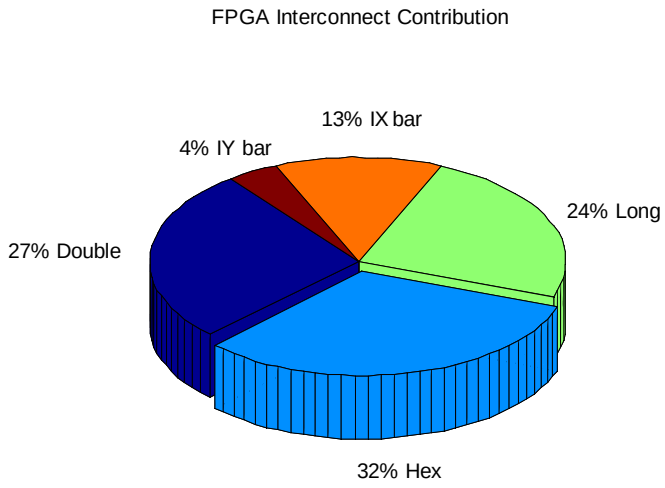
The breakdown of these contributions is shown in Figure 3.29. Figure 3.29.a shows that, for this technology, the role of the interconnections is important.

Moreover, in the modern FPGA architectures, different interconnect structures are used for routing signals characterized by different degrees of locality. Their contribution to the capacitance is shown in Table 3.9.3 where it can be seen that capacitances are noticeably different for the local and the global lines.

Finally Figure 3.29.b, shows that global interconnects give a big contribution to the overall power consumption in FPGAs. Starting from this observation, we can foreseen a greater advantage exploiting RNS locality by implementing circuits in FPGA technology.



a) FPGA power contributions



b) FPGA power interconnect

Figure 3.29: Power consumption breakdown for FPGA implementation

	Description	$\frac{A_{RNS}}{A_{TCS}}$	$\frac{P_{RNS}}{P_{TCS}}$	$\frac{GI_{RNS}}{GI_{TCS}}$
1	8-taps FIR	1.1	0.612	0.554
2	16-taps FIR	0.947	0.51	0.539
3	8-taps CS FIR	1.095	0.533	0.487
4	16-taps CS FIR	0.93	0.416	0.447

Table 3.14: Area, power and globality ratios for FPGA implementation.

3.9.4 Analysis of Experimental Results: FPGA

The analysis of the effect of locality in FPGAs has been carried out by using the same model used for the ASIC-SC technology.
In this case the area corresponds to the number of slices used in the implementation.

The effects of the locality of the RNS implementation are evident in Table 3.14. In fact for FPGA technology the GI ratio (GI_R) is about 0.5.

Chapter 4

Tools

Although digital filters based on the Residue Number System (RNS) show high performance and low power dissipation, RNS filters are not widely used in DSP systems, because of the complexity of the algorithms involved. Consequently, a tool to design RNS FIR filters which

- hides the RNS algorithms to the designer,
- generates a synthesizable VHDL description of the filter taking into account several design constraints such as: delay, area and energy

is important to help designers to exploit this technology (see GUI in Figure 4.1). The tool discussed in this section, is able to help in the design of both programmable and constant coefficients FIR filters in transposed form. Moreover, the tool chooses the set of RNS moduli which cover the given dynamic range and best fit the design constraints. The design space exploration is based upon a characterization of the blocks composing the filter, and it is done for the different technologies supported: standard cells and FPGAs. Differently from the tool for RNS design presented in [51], we only target FIR filters but offer a wide range of design choices and an automatic selection of the set of moduli which best fit the design constraints. The tool is suitable for an IP oriented design of System-on-Chips offering to the designer the performances of RNS, but completely hiding its complexity.

4.1 Tool Description

The structure of the tool is shown in Figure 4.2. It is divided into three main blocks:

1. **Front-end:** which generates a list of parameters specifying the filter characteristics such as: dynamic range (i.e. wordlength), filter order, etc.. The

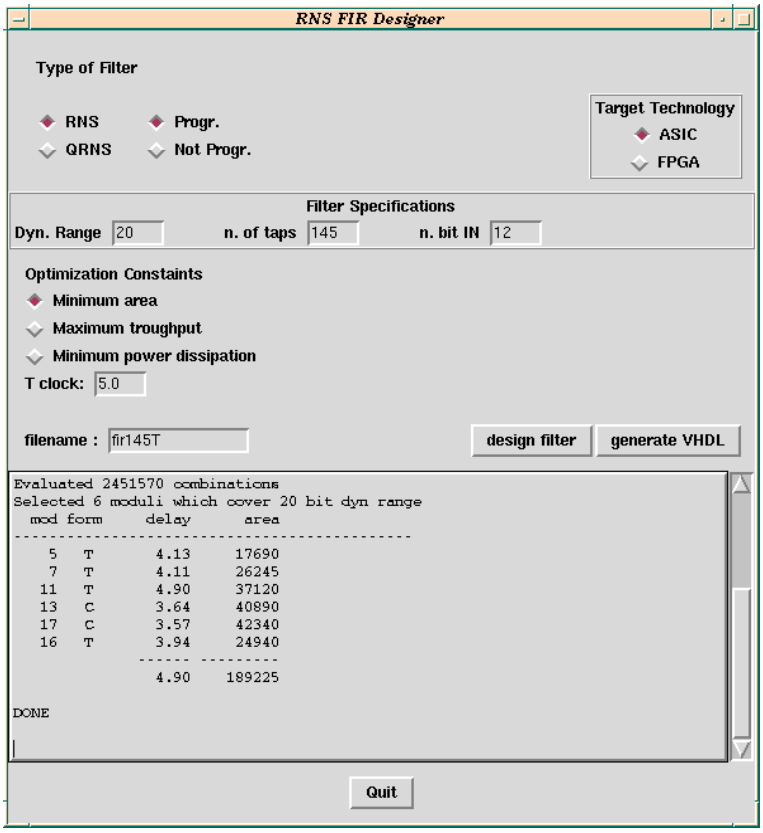


Figure 4.1: Tool interface for the designer.

front-end could be a commercial tool, such as MATLAB.

2. **Architecture Chooser (AC):** AC chooses the filter architecture, selected among a set of supported ones, that minimizes a given cost function. The selection is done according to the filter parameters passed by the front-end, the target technology library specifications, and the design constraints. Moreover, AC generates a set of instructions describing the detail of the selected architecture to be passed to the builder.

3. **VHDL Builder (VB):** generates the VHDL descriptions of the elementary blocks (modular multipliers and adders, converters, ...), and the top-level filter netlist. Moreover, it builds a VHDL test bench file to verify the filter functionality (e.g. to check if the VHDL simulations match the MATLAB fixed point simulation results), and a synthesis script for the synthesizer (only Synopsys Design Compiler is supported at this time).

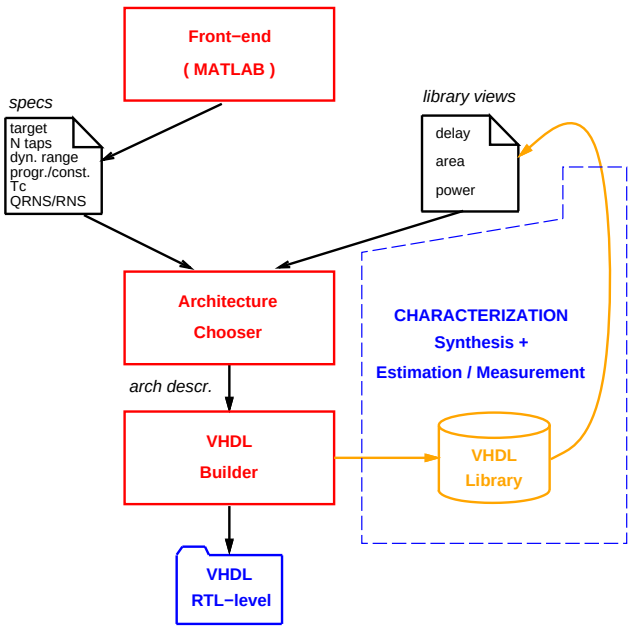


Figure 4.2: Structure of the tool.

4.2 Architecture Chooser

The Architecture Chooser (AC), according to the project constraints and the library characteristics (timing, area and power), determines the set of RNS moduli to be used, the architectures used in each RNS path, and the overall filter organization (RNS paths plus converters [47]). Currently, AC supports a number of optimization directives such as (T_C is the clock period):

- Given T_C, design filter with minimum area.
- Given T_C, design filter with lower power dissipation.
- Design the filter that can sustain the maximum throughput (minimum T_C).

Moreover, The AC instructs the VHDL Builder, to place pipeline registers, if necessary.

4.3 Characterization

The characterization of the basic blocks composing the filter is the key for an accurate estimate of the cost function. Therefore, the outcome of AC depends on an accurate characterization. For each RNS modulus (currently we consider a set of about 30) all basic blocks must be characterized in terms of timing, area and

dyn	N	T_C $[ns]$	estimated	actual	est./act.
20	40	5.0	37208	41368	0.90
20	40	4.5	39511	42938	0.92
20	40	4.0	42250	44828	0.94
16	40	4.5	29861	32705	0.91
24	40	4.5	53419	58537	0.91
16	40	4.0	32103	34039	0.94
16	80	4.0	64206	69384	0.93
16	120	4.0	96309	104462	0.92

Table 4.1: Area estimation for generated filters.

power. Furthermore, because we provide RTL-level descriptions to the synthesizer, the area (and the power dissipation) largely depends on timing constraints. As a consequence, the area and power characterization of a block must be done for different clock rates, increasing the complexity of the cost function to minimize. For the characterization of the power dissipation, we assume random activity at the blocks input. To make the tool as independent as possible of the library, as an option in AC, we can use standard units for (delay, area, and power)[1]. In this way, when changing technology, the characterization process is reduced to a few runs. We use the VHDL Builder to generate all blocks to be characterized (Figure 4.2).

4.4 Examples and Results

To test the accuracy of the estimates and, consequently, the soundness of the architectural choices made, we generated with the tool a number of FIR filters varying the dynamic range (dyn), the filter order (N), and the timing constraints (T_C). The target technology is the STM 0.35 μm standard cell library. In Table 4.1, we report the area estimates (as NAND2 equiv.) and the corresponding actual values determined after synthesis performed by Synopsys Design Compiler. From the last column of the table, the estimate error is less than 10% on the average.

[1]The delay of a NOT gate with fan-out=4, the area of a NAND2 gate, and the power dissipated by a NOT with fan-out=1 at 100 MHz.

Chapter 5

Conclusions and Future Work

In this chapter, a brief analysis of the importance of the use of RNS with respect to the evolving nanometric technologies and to the new available FPGA architectures, is given. These considerations are also linked to the trends of embedded electronics. The following topics will be briefly analyzed

- DSP functions of interest
- Hardware platforms of interest
- Candidate applications
- Proposal for further developments

5.1 DSP Functions of Interest and Applications

As stated in the report and illustrated in the case study (FIR filters), the most straightforward use of the RNS is in operations that can be categorized as linear combinations of integer numbers. For example the linear combination operator is fundamental in many signal processing algorithms such as filtering or transforms. Consequently, formulas of the following type are suitable for RNS implementations

$$Z = \sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \cdots \sum_{i_P=1}^{N_P} a_{i_1,\ i_2,\cdots,i_p} \cdot X_{i_1,\ i_2,\cdots,i_p} \tag{5.1}$$

where $a_{i_1,\ i_2,\cdots,\ i_p}$ and $X_{i_1,\ i_2,\cdots,i_p}$ can be either real or complex numbers. The RNS deals with real numbers, but it can be extended to complex computations with QRNS.

Referring to (5.1) the RNS is particularly useful when

1. the input data data to the computational unit are characterized by a large dynamic range

2. the order of the linear combination is high (i.e. the number of summations and the summation indices are high)

3. the final result must be accurate in the sense that the application is critical in terms of quantization errors (take into account that RNS offers error free computations)

In practice when the number of Multiply and Accumulate (MAC) operations to compute the result is very high RNS is convenient. Many applications are critical in this sense:

1. High order FIR filters (both real and complex filters). Often in real application we need to implement parallel filters with hundreds of taps in order to obtain a high frequency selectivity. In situations in which a very precise frequency mask is needed errors derived by

   - rounding of the filter coefficients (coefficient characterized by a large number of bits)
   - truncation in the internal filter computations (it is necessary to avoid truncation at the multipliers output)

   must be avoided or limited.

2. Multidimensional FIR filtering. There are applications in which is needed to elaborate multidimensional signals (medical imaging).

3. High resolution and high speed implementation for the DFT. There is an extending field of applications for such transforms. In Electronic Warfare (EW) for example, the most important apparatus is a high dynamic range and very large bandwidth radio receiver. The real time analysis of the spectrum is often based on an high resolution FFT.

4. High speed implementation for wavelet transforms.

5. Adaptive signal processing when the adaptation rate is high.

Another important factor that drives to faster DSP units and consequently to a number representation that is intrinsically parallel such as the RNS, is the very fast growth in the performance of Analog to Digital (ADC) and Digital to Analog (DAC) converters. At present it is easy to see on the market ADCs and DACs characterized by 10/12 bits of resolution at conversion speeds up to 2 GSPS.

Other applications of the RNS are in the following fields

1. **Cryptography.** In this relative new field for RNS, the designer deals with very high dynamic range integer numbers that can take advantage from the RNS as shown in the recent work in the literature [52].

2. **Very high resolution Direct Digital Frequency Synthesis (DDFS).** An increment in the resolution of a DDFS is obtained by implementing an accumulator characterized by a very high number of bits. If low power consumption is required, at the same speed, the RNS is convenient (see the patents section).

3. **Low power implementations.** FPGAs are really important in these days due to the flexibility they add in the design chain. FPGAs can be reprogrammed and a small number of different parts can be used in the product design. The limitation of the FPGAs capability to penetrate important markets, such as consumer portable multimedia and space, is related to the power hungry technology. To reduce the power dissipation in FPGAs, special design techniques must be used. In particular, it has been proved in literature that saving of the order of 30% can enable FPGAs to these markets.

5.2 Proposal for Further Developments

To extend the field of application of the RNS, the following topics should be investigated

1. Efficient scaling techniques. Efficient algorithms for the scaling operation permits to optimize the use of the RNS in adaptive signal processing and IIR filtering. In fact, in the case of IIR filters, due to the feedback, scaling is needed to avoid the growth in wordlength.

2. Polyphase Filter banks. In [53] a polyphase filter bank in the Quadratic Residue Number System (QRNS) has been implemented and then compared, in terms of performance, area, and power dissipation to the implementation of a polyphase filter bank in the traditional two's complement system (TCS). The resulting implementations, designed to have the same clock rates, show that the QRNS filter is smaller and consumes less power than the TCS one. More investigations especially in the case in which an FPGA is chosen has the target architecture are important due to the complexity of this architectures.

3. Improvement in the tool for the automatic generation of RNS based Intellectual Property (IP) blocks by a more accurate characterization of the library blocks.

4. More accurate power consumption evaluation for the new FPGAs families.

5. Exploration of the advantages obtainable in the synthesis and mapping of RNS architectures to the new families of FPGAs based on 6-input LUTs.

6. Introduction of fault tolerant techniques based on RNS in DSP blocks, to increment the reliability due to losses such as the radiation effects on nanometric technologies.

7. Evaluation of power consumption earnings that can be obtained by translating software procedures into RNS. In this way the microprocessor multiply unit will not be used and only the internal memory access and modular additions will be performed.

8. Use of the RNS in high dynamic range operations to be executed on low cost microcontrollers characterized by the absence of the multiplier.

Acknowledgments

The stay of Prof. Marco Re at DTU was supported by the Otto Mønsteds Fond in the context of a Visiting Professor grant for the years 2007-2008.

Prof. Marco Re wishes to thank all the people at DTU Informatics for their kind hospitality and in particular the Director of the Department Prof. Kaj Madsen who evaluated his application to be qualified for selection by the Otto Mønsteds Fond.

Marco particularly thanks Prof. Alberto Nannarelli who invited him for the visiting professor position and Prof. Flemming Stassen for the interesting historical and scientific discussions.

Bibliography

[1] A. Svoboda and M. Valach. Operational circuits. *Stroje Na Zpracovani Informaci*, 3, 1955.

[2] A. Svoboda and M. Valach. Rational residual system of residual classes. *Stroje Na Zpracovani Informaci*, 5:9–37, 1957.

[3] A. Svoboda and M. Valach. The numerical system of residual classes in mathematical machines. *Proc. Congr. Int. Automa*, 1958.

[4] H. H. Aiken and W. Semon. Advanced digital computer logic. *Tech. Rep. WADC TR*, pages 59–472, 1959.

[5] R. A. Bauught and E. C. Day. Electronic sign evaluation for residue number systems,. *RCA Tech. Rep. No. TR-60-597-32*, 1961.

[6] R. I. Tanaka. Modular arithmetic techniques. *Tech. Rep. No. 2-38-62-1A, ASTDR, Lockheed Missiles and Space Co.*, 1962.

[7] D.L. Slotnick. Modular arithmetic computing techniques. *Tech.Rep ASD -TDR-63-280, Westinghouse Electric Corporation, Air Arm Division*, 1963.

[8] R. C. Aitken. Nanometer technology effects on fault models for ics testing. *Computer*, pages 46–51, 1999.

[9] ITRS. *Nanometer Technology Effects on Fault Models for ICs Testing*, 2007. http://www.itrs.net/.

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

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

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

[13] P. V. Ananda Mohan. *Residue Number Systems: Algorithms and Architectures*. The Springer International Series in Engineering and Computer Science, 2002.

[14] Marco Re. *Metodologie di conversione ingresso-uscita in processori in aritmetica finita*. PhD thesis, University of Rome Tor Vergata, 1996.

[15] T. Stouraitis. Efficient convertors for residue and quadratic-residue number systems. *IEE PROCEEDINGS-G*, 139(6):626–634, 1992.

[16] T. V. Vu. Efficient implementation of the chinese remainder theorem for sign detection and residue decoding. *IEEE Trans. Circuits Systems-I*, 45:667–669, June 1985.

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

[18] G. Cardarilli, M. Re, and R. Lojacono. A residue to binary conversion algorithm for signed numbers. *European Conference on Circuit Theory and Design (ECCTD'97)*, 3:1456–1459, 1997.

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

[20] Z. D. Ulman and M. Czyzak. Highly parallel, fast scaling of numbers in nonredundant residue arithmetic. *IEEE Trans. Signal Processing*, 46:487–496, 1998.

[21] A. Garcia and A. Lloris. A look-up scheme for scaling in the RNS. *IEEE Trans. Comput.*, 48:748–751, 1999.

[22] M. A. P. Shenoy and R. Kumaresan. A fast and accurate RNS scaling technique for high speed signal processing. *IEEE Trans. Acoust., Speech, Signal Processing*, 37:929–937, 1989.

[23] F. Barsi and M. C. Pinotti. Fast base extension and precise scaling in RNS for look-up table implementations. *IEEE Trans. Signal Processing*, 43:2427–2430, 1995.

[24] G. C. Cardarilli, M. Re, R. Lojacono, and G. Ferri. A systolic architecture for high performance scaled residue to binary conversion. *IEEE Trans. Circuits Syst. I*, 47:1523–1526, 2000.

[25] N. Burgess. Scaled and unscaled residue number system to binary conversion techniques using the core function. *Proc. of 13th Symposium on Computer Arithmetic*, pages 250–257, 1997.

[26] M. Griffin, M. Sousa, and F. Taylor. Efficient scaling in the residue number system. *Proc. of Intl. Conf. on Acoustics, Speech, and Signal Processing*, pages 1075–1078, 1989.

[27] U. Mayer-Base and T. Stouraitis. New power-of-two scaling scheme for cell-based IC design. *IEEE Trans. VLSI Syst.*, 1:446–450, 2003.

[28] F. Barsi and P. Maestrini. Error Correction Properties of Redundant Residue Number System. *IEEE Transactions on Computers*, pages 370–375, March 1973.

[29] F. Barsi and P. Maestrini. Error Detection and Correction by Product Codes in Residue Number System. *IEEE Transactions on Computers*, pages 915–924, Sept. 1974.

[30] R. Lojacono G. C. Cardarilli, M. Re. Efficient modulo extraction for crt based residue to binary converters. *IEEE International Symposium on Circuits and Systems*, 3:2036–2039, 1997.

[31] R. Conway and J. Nelson. Fast converter for 3 moduli rns using new property of crt. *IEEE Trans. on Computers*, 48(8):852–860, 1999.

[32] D. Gallaher, F. E. Petry, and P. Srinivasan. The digit parallel method for fast rns to weighted number system conversion for specific moduli ($2^k-1,\ 2^k,\ 2^k+1$). *IEEE Trans. Circuits Syst.-II*, 44(1):53–57, 1997.

[33] A. B. Premkumar. An rns to binary converter in 2n-1, 2n, 2n+1 moduli set. *IEEE Trans. Circuits Syst.-II*, 39(7):480–482, 1992.

[34] A. B. Premkumar, M. Bhardwaj, and T. Srikathan. High-speed and low-cost reverse converters for the $2n-1,\ 2n,\ 2n+1$ moduli set. *IEEE Trans. Circuits Syst.-II*, 45(7):903–908, 1998.

[35] K.M. Elleyth and M.A. Bayoumi. Fast and flexible architectures for rns arithmetic decoding. *IEEE Trans.Circuits Systems.-II Analog and Digital Signal Processing*, 39(4):226–235, 1992.

[36] F. Barsi. Mod m arithmetic in binary systems. *Information Processing Letters*, 40(6):303–309, 1991.

[37] A. P. Chandrakasan and R. W. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.

[38] STMicroelectronics. *90nm CMOS090 Design Platform*. http://www.st.com/stonline/prodpres/dedicate/soc/asic/90plat.htm.

[39] A. Del Re, A. Nannarelli, and M. Re. A tool for automatic generation of RTL-level VHDL description of RNS FIR filters. *Proc. of 2004 Design, Automation and Test in Europe Conference (DATE)*, 48:686–687, Feb. 2004.

[40] K. Usami and M. Horowitz. Clustered voltage scaling technique for low-power design. *Proc. of International Symposium on Low Power Design*, pages 3–8, Apr. 1995.

[41] N. H. E. Weste and K. Eshraghian. *Principles of CMOS VLSI Design*. Addison-Wesley Publishing Company, 2nd edition, 1993.

[42] Synopsys Inc. *Synopsys products: Design Compiler*. http://www.synopsys.com/products/logic/design_compiler.html.

[43] B. Chen and I. Nedelchev. Power compiler: A gate-level power optimization and synthesys system. *Proc. of International Conference on Computer Design (ICCD)*, pages 74–78, Oct. 1997.

[44] G. C. Cardarilli, A. Nannarelli, and M. Re. Reducing power dissipation in FIR filters using the residue number system. *to appear in Proc. of the 43rd IEEE Midwest Symposium on Circuits and Systems*, Aug. 2000. Available at http://dspvlsi.uniroma2.it/pubs/mwscas00/.

[45] P. S. Moharir. Extending the scope of golub's method beyond complex multiplication to binary converters. *IEEE Transactions on Computers*, C-34(5):484–487, 1985.

[46] *Xilinx Inc.* http://www.xilinx.com/.

[47] A. Nannarelli, M. Re, and G. C. Cardarilli. Tradeoffs between Residue Number System and Traditional FIR Filters. *Proc. of IEEE International Symposium on Circuits and Systems*, II:305–308, May 2001.

[48] U. Weiser, N. Magen, A. Kolodny, and N. Shamir. Interconnect-power dissipation in a microprocessor. *Proc. of SLIP'04*, February 2004.

[49] T. Stouraitis and V. Paliouras. Considering the alternatives in low-power design. *IEEE Circuits and Devices Magazine*, 17:22–29, July 2001.

[50] A. S. Kaviani, L. Shang, and K. Bathala. Dynamic power consumption in Virtex-II FPGA family. *Proc. of FPGA'02*, February 2002.

[51] D. Soudris et al. A methodology for implementing FIR filters and CAD tool development for designing RNS-based systems. *Proc. of IEEE Int.l Sym. on Circuits and Systems*, V:129–132, May 2003.

[52] J-C. Bajard and L. Imbert. A full RNS implementation of RSA. *IEEE Transaction on Computers*, pages 769–774, June 2004.

[53] G. C. Cardarilli, A. Del Re, A. Nannarelli, and M. Re. Low-power implementation of polyphase filtersnin quadratic residue number system. *IEEE International Symposium on Circuits and Systems, ISCAS 2004*, 2:725–728, 2004.