Data Conversion
in Residue Number System

Omar Abdelfattah

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

January 2011

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Engineering.

Abstract

This thesis tackles the problem of data conversion in the Residue Number System (RNS). The RNS has been considered as an interesting theoretical topic for researchers in recent years. Its importance stems from the absence of carry propagation between its arithmetic units. This facilitates the realization of high-speed, low-power arithmetic. This advantage is of paramount importance in embedded processors, especially those found in portable devices, for which power consumption is the most critical aspect of the design. However, the overhead introduced by the data conversion circuits discourages the use of RNS at the applications. In this thesis, we aim at developing efficient schemes for the conversion from the conventional representation to the RNS representation and vice versa. The conventional representation can be in the form of an analog continuous-time signal or a digital signal represented in binary format. We present some of the currently available algorithms and schemes of conversion when the signal is in binary representation. As a contribution to this field of research, we propose three different schemes for direct conversion when interaction with the real analog world is required. We first develop two efficient schemes for direct analog-to-residue conversion. Another efficient scheme for direct residue-to-analog conversion is also proposed. The performance and the efficiency of theses converters are demonstrated and analyzed. The proposed schemes are aimed to encourage the utilization of RNS in various real-time and practical applications in the future.

Résumé

Cette thèse aborde le problème de la conversion de données dans le système numérique de résidus (*Residue Number System* - RNS). Le système RNS a été considéré comme un sujet intéressant par de nombreux chercheurs ces dernières années. Son importance découle de l'absence de la propagation de retenue entre ses unités de calcul. Ceci facilite la réalisation de circuits arithmétiques à grande vitesse et de faible puissance. Cet avantage est d'une importance primordiale dans les processeurs embarqués, en particulier ceux qu'on retrouve dans les appareils portables, pour lesquels la consummation d'énergie est l'aspect le plus critique de la conception. Cependant, le traitement supplémentaire introduit par les circuits de conversion de données décourage l'utilisation du RNS au niveau des applications. Dans cette thèse, nous cherchons des schémes efficaces pour la conversion de la représentation conventionnelle à la représentation RNS et vice-versa. La représentation conventionnelle peut être sous la forme d'un signal analogique en temps continu où d'un signal échantillonné numérique représenté en format binaire. Nous présentons quelques algorithmes actuellement disponibles et les systèmes de conversion associés lorsque le signal est sous une représentation binaire. Dans notre contribution à ce domaine de recherche, nous proposons trois astuces différentes pour la conversion lorsqu'une interaction avec le monde analogique réel est nécessaire. Nous développons deux systèmes efficaces pour la conversion directe du domaine analogique à RNS. Un autre système efficace pour la conversion directe de RNS à analogique est également proposé. La performance et l'efficacité de ces convertisseurs sont mises en évidence et analysées. Les schémas proposés sont destinés à encourager l'utilisation du RNS dans diverses applications dans l'avenir.

Acknowledgements

I would like to express my gratitude to the following people who supported and encouraged me during this work. First, I am grateful to my supervisors, Zeljko Zilic and Andraws Swidan, for giving me full independence and trust till I reached to this research topic and then for their unlimited assistance throughout my research toward my Master degree. Second, I would like to thank all my talented friends in Integrated Microsystems Laboratory (IML) and Microelectronics And Computer Systems (MACS) Laboratory for their help and guidance and for providing the friendly atmosphere that encouraged me in my daily progress. I would like also to thank all the professors who taught me in my undergraduate study in Kuwait University and in my graduate career in McGill University. Special thanks go to my parents, the reason that I exist, and to my sister who offered me all help and support during writing this thesis. I cannot adequately express my gratitude to all those people who made this thesis possible.

Contents

List of Figures

List of Tables

List of Acronyms

RNS	Residue Number System
CRT	Chinese Remainder Theorem
MRC	Mixed-Radix Conversion
ADC	Analog-to-Digital Converter
DAC	Digital-to-Analog Converter
B/R	Binary-to-Residue
R/B	Residue-to-Binary
A/R	Analog-to-Residue
R/A	Residue-to-Analog
ROM	Read Only Memory
LUT	Look-Up Table

Chapter 1

Introduction

A riddle posted in a book authored by a Chinese scholar called *Sun Tzu* in the first century was the first documented manifestation of *Residue Number System (RNS)* representation [1,2]. The riddle is described by the following statement:

We have things of which we do not know the number:

If we count them by threes, the remainder is 2.

If we count them by fives, the remainder is 3.

If we count them by sevens, the remainder is 2.

How many things are there?

The answer is 23.

The mathematical procedure of obtaining the answer 23 in this example from the set of integers 2, 3, and 2 is what was later called the *Chinese Remainder Theorem (CRT)*. The CRT provides an algorithmic solution of decoding the residue encoded number back into its conventional representation. This theorem is considered the cornerstone in realizing RNSs.

Encoding a large number into a group of small numbers results in significant speed up of the overall data processing. This fact encourages the implementation of RNS in some applications where intensive processing is inevitable.

In this chapter, we present the clear motivation of this thesis along with the main contributions. We also provide an introduction to RNS representation, properties, advantages, drawbacks, and applications.

1.1 Thesis Motivation

A general structure of a typical RNS processor is shown in Figure 1.1. The RNS represented data is processed in parallel with no dependence or carry propagation between the processing units. The process of encoding the input data into RNS representation is called *Forward Conversion*, and the process of converting back the output data from RNS to conventional representation is called *Reverse Conversion*.
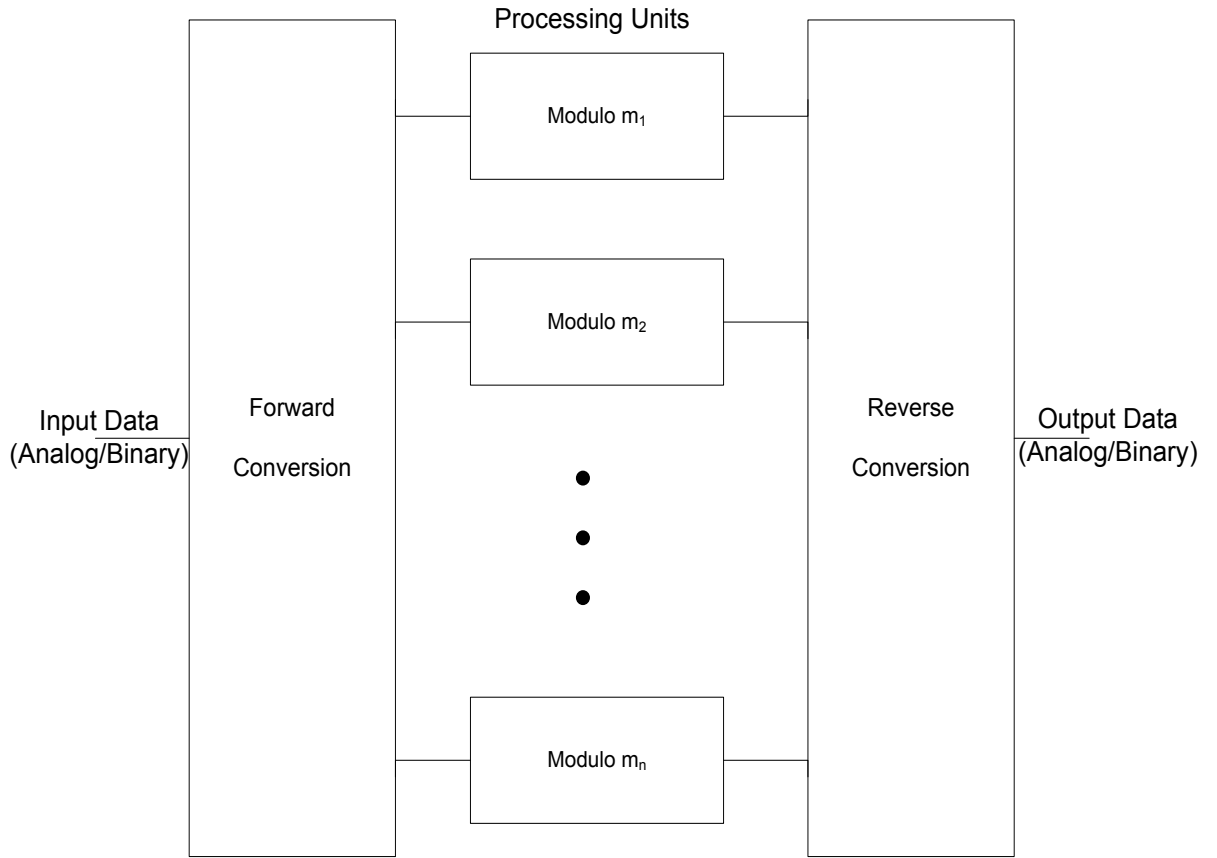
Figure 1.1. General structure of an RNS-based processor

The conversion stages are very critical in the evaluation of the performance of the overall RNS. Conversion circuitry can be very complex and may introduce latency that offsets the speed gained by the RNS processors. For a full RNS based system, the interaction with the analog world requires conversion from analog to residue and vice versa. Usually, this is done in two steps where conversion to binary is an intermediate stage. This makes the conversion stage inefficient due to their increased latency and complexity. To build an RNS

processor that can replace the digital processor in a certain application; we need to develop conversion circuits that perform as efficient as the analog-to-digital converter (ADC) and the digital-to-analog converter (DAC) in the digital binary-based systems. The reverse conversion process is based on the Chinese Remainder Theorem (CRT) or Mixed-Radix Conversion (MRC) techniques. Investigating new conversion schemes can lead to overcoming some obstacles in the RNS implementation of different applications. Thus, an analog-to-residue (A/R) converter and a residue-to-analog (R/A) converter are sought to eliminate the intermediate binary stage.

1.2   Main Contributions of This Work

The main contributions of this work are summarized as follows:

1. Two architectures for direct analog-to-residue conversion are proposed. The first proposed architecture is based on the two-stage flash conversion principle, while the second architecture is based on the successive approximation principle. The two architectures obviate the need of an intermediate binary stage and expedite the conversion process.

2. One architecture for direct residue-to-analog conversion is proposed. The proposed architecture is based on the CRT. The need for an intermediate binary stage is eliminated.

Overall, the proposed architectures facilitate the implementation of RNS based processors by reducing the latency and complexity introduced by the binary stage. This makes it more possible and more practical to build effective RNS based processors.

1.3   RNS Representation

An RNS is defined by a set of relatively prime integers called the *moduli*. The moduli-set is denoted as $\{m_1, m_2, \ldots, m_n\}$ where m_i is the ith *modulus*. Each integer X can be represented as a set of smaller integers called the *residues*. The residue-set is denoted as $\{r_1, r_2, \ldots, r_n\}$ where r_i is the ith residue. The residue r_i is defined as the least positive remainder when X is divided by the modulus m_i. This relation can be notationally written based on the congruence:

$$X \bmod m_i = r_i \tag{1.1}$$

The same congruence can be written in an alternative notation as:

$$|X|_{m_i} = r_i \tag{1.2}$$

The two notations will be used interchangeably throughout this thesis.

The RNS is capable of uniquely representing all integers X that lie in its *dynamic range*. The dynamic range is determined by the moduli-set $\{m_1, m_2, \ldots, m_n\}$ and denoted as M where:

$$M = \prod_{i=1}^{n} m_i \tag{1.3}$$

The RNS provides unique representation for all integers in the range between 0 and $M - 1$. If the integer X is greater than $M - 1$, the RNS representation repeats itself. Therefore, more than one integer might have the same residue representation.

It is important to emphasize that the moduli have to be relatively prime to be able to exploit the full dynamic range M.

To illustrate the preceding principles, we present a numerical example.

Example 1.1.

Consider two different residue number systems defined by the two moduli-sets $\{2, 3, 5\}$ and $\{2, 3, 4\}$. The representation of the numbers in residue format is shown in Table 1.1. for the two systems.

Table 1.1. RNS representation for two different moduli-sets

X	$\{2, 3, 5\}$			$\{2, 3, 4\}$		
	2	3	5	2	3	4
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	0	2	2	0	2	2
3	1	0	3	1	0	3
4	0	1	4	0	1	0
5	1	2	0	1	2	1
6	0	0	1	0	0	2
7	1	1	2	1	1	3
8	0	2	3	0	2	0
9	1	0	4	1	0	1
10	0	1	0	0	1	2
11	1	2	1	1	2	3
12	0	0	2	**0**	**0**	**0**
13	1	1	3	1	1	1

14	0	2	4	0	2	2
15	1	0	0	1	0	3
16	0	1	1	0	1	0
17	1	2	2	1	2	1
18	0	0	3	0	0	2
19	1	1	4	1	1	3
20	0	2	0	0	2	0
21	1	0	1	1	0	1
22	0	1	2	0	1	2
23	1	2	3	1	2	3
24	0	0	4	0	0	0
25	1	1	0	1	1	1
26	0	2	1	0	2	2
27	1	0	2	1	0	3
28	0	1	3	0	1	0
29	1	2	4	1	2	1
30	**0**	**0**	**0**	0	0	2

In the first RNS, the moduli in the moduli-set $\{2, 3, 5\}$ are relatively prime. The RNS representation is unique for all numbers in the range from 0 to 29. Beyond that range, the RNS representation repeats itself. For example, the RNS representation of 30 is the same as that of 0. In the second RNS, the moduli in the moduli-set $\{2, 3, 4\}$ are not relatively prime, since 2 and 4 have a common divisor of 2. We notice that the RNS representation repeats itself at 12 preventing the dynamic range from being fully exploited. Therefore, choosing relatively prime moduli for the RNS is necessary to ensure unique representation within the dynamic range.

In the preceding discussion on RNS, we assumed dealing with unsigned numbers. However, some applications require representing negative numbers. To achieve that, we can partition the full range $[0: M - 1]$ into two approximately equal halves: the upper half represents the positive numbers, and the lower half represents the negative numbers. The numbers X that can be represented using the new convention have to satisfy the following relations [4]:

$$-\frac{M-1}{2} \leq X \leq \frac{M-1}{2} \quad \text{if } M \text{ is odd} \tag{1.4}$$

$$-\frac{M}{2} \leq X \leq \frac{M}{2} - 1 \quad \text{if } M \text{ is even} \tag{1.5}$$

If $X = \{r_1, r_2, \ldots, r_n\}$ represents a positive number in the appropriate range, then $-X$ can be represented as $\{\overline{r_1}, \overline{r_2}, \ldots, \overline{r_n}\}$ where $\overline{r_i}$ is the m_i's complement of r_i, i.e. $\overline{r_i}$ satisfies the relation $(r_i + \overline{r_i}) mod\ m_i = 0$. In our discussion, we will assume that the numbers are unsigned unless otherwise it is mentioned.

Example 1.2.

Consider an RNS with the moduli-set $\{3, 4, 5\}$. The number 18 is represented as $\{0, 2, 3\}$ while the number -18 is represented as $\{0, 2, 2\}$.

The justification for that is as follows:

$$(0 +\ 0) mod\ 3 = 0$$
$$(2 +\ 2) mod\ 4 = 0$$
$$(3 +\ 2) mod\ 5 = 0$$

Therefore, the positive numbers are represented in the upper half of the dynamic range and the conversion to residue representation is straightforward, while the negative numbers are represented in the lower half of the dynamic range and the conversion to residue representation is interpreted as the conversion of the compliments of the residues with respect to the corresponding moduli.

1.4　　Mathematical Fundamentals

In this section, we introduce the fundamentals of the RNS representation. The congruences are explained in details with their properties. These properties form a solid background to understand the process of conversion between the conventional system and the RNS. More advanced results and mathematical relations can be found in the subsequent chapters. Basic algebra related to RNS is introduced here. This includes finding the additive and the multiplicative inverses, and some properties of division and scaling which are not easy operations in RNS.

1.4.1　Basic Definitions and Congruences

Residue of a number

The basic relationship between numbers in conventional representation and RNS representation is the following congruence:

$$X \bmod m_i = r_i \qquad (1.6)$$

where m_i is the *modulus*, and r_i is the *residue*. The residue is defined as the least positive remainder when the number X is divided by the modulus m_i.

Example 1.3.

For $X = 57$, $m_1 = 4$, and $m_2 = 5$, we find the residues r_1 and r_2 with respect to the moduli m_1 and m_2, respectively as follows:

$$57 \bmod 4 = 1 \quad since \quad 57 = 4 \times 14 + 1$$
$$57 \bmod 5 = 2 \quad since \quad 57 = 5 \times 11 + 2$$

Definition of the base values

With respect to modulus m_i, any number X can be represented as a combination of a *base value* B_i and a *residue* r_i:

$$X = B_i + r_i \qquad (1.7)$$
$$and \qquad B_i = k \times m_i \qquad (1.8)$$

where k is an integer that satisfies Equations (1.7) and (1.8).

The definition of the base value will be exploited in Chapter 4 where these values will be generated to directly convert from analog to RNS representation.

1.4.2   Basic Algebraic Operations

Addition (or subtraction)

We can add (or subtract) different numbers in the RNS representation by individually adding (or subtracting) the residues with respect to the corresponding moduli.

Consider the moduli-set $S = \{m_1, m_2, \ldots, m_n\}$, and the numbers X and Y are given in RNS representation:

$$X = \{x_1, x_2, \ldots, x_n\} \quad and \quad Y = \{y_1, y_2, \ldots, y_n\}$$

Then,

$$Z = X + Y = \{z_1, z_2, \ldots, z_n\} \qquad (1.9)$$
$$where \ z_i = (x_i + y_i) \bmod m_i$$

This property can be applied to subtraction as well, where subtraction of Y from X is considered as the addition of $\overline{Y}$.

The modulo operation is distributive over addition (and subtraction):

$$|X \mp Y|_m = ||X|_m \mp |Y|_m|_m \tag{1.10}$$

Multiplication

In a similar way to addition, multiplication in RNS can be carried out by multiplying the individual residues with respect to the corresponding moduli. Consider the moduli-set $S = \{m_1, m_2, ..., m_n\}$, and the numbers X and Y are given in RNS representation:

$$X = \{x_1, x_2, ..., x_n\} \quad \text{and} \quad Y = \{y_1, y_2, ..., y_n\}$$

Then,

$$Z = X \times Y = \{z_1, z_2, ..., z_n\} \tag{1.11}$$
$$\text{where } z_i = (x_i \times y_i) \bmod m_i$$

The modulo operation is distributive over multiplication:

$$|X \times Y|_m = ||X|_m \times |Y|_m|_m \tag{1.12}$$

Additive Inverse

The relation between the residue r_i and its additive inverse $\overline{r_i}$ is defined by the congruence:

$$(r_i + \overline{r_i}) \bmod m_i = 0 \tag{1.13}$$

The additive inverse $\overline{r_i}$ can be obtained using the following operation:

$$\overline{r_i} = (m_i - r_i) \bmod m_i \tag{1.14}$$

Subtraction is one application of this property, where subtraction is regarded as the addition of the additive inverse.

Example 1.4.

Given the moduli-set $\{2, 3, 5\}$, the dynamic range is $M = 30$. The RNS can uniquely represent all numbers in the range $[0:29]$. Let $X = 28 \triangleq \{0, 1, 3\}$ and $Y = 24 \triangleq \{0, 0, 4\}$. To find $-Y$, we need first to obtain $\overline{Y}$, and then find $X + \overline{Y}$. First,

$$\overline{Y} = \{(2 - 0) \bmod 2, (3 - 0) \bmod 3, (5 - 4) \bmod 5\} = \{0,0,1\}$$

Then, $X - Y = X + \bar{Y} = \begin{cases} (0+0) \bmod 2, (1+0) \bmod 3, \\ (3+1) \bmod 5 \end{cases} = \{0,1,4\}$ which is the RNS

representation of 4.

Multiplicative Inverse

The multiplicative inverse r_i^{-1} of the residue r_i is defined by the congruence:

$$(r_i \times r_i^{-1}) \bmod m_i = 1 \tag{1.15}$$

where r_i^{-1} exists only if r_i and m_i are relatively prime.

Example 1.5.

For the modulus $m = 5$, we find the multiplicative inverse r^{-1} of the residue $r = 3$ by applying Equation (1.15):

$$(3 \times r^{-1}) \bmod 5 = 1$$

We notice that the modulo multiplication of 3 and 2 with respect to 5 results in 1.

Thus, $\qquad\qquad\qquad\qquad\qquad r^{-1} = 2$

As illustrated in Example 1.5., there is no general method of obtaining the multiplicative inverse. The multiplicative inverse is usually obtained by brute-force search. Only when m is prime, we can utilize Fermat's Theorem which can be useful in determining the multiplicative inverse. This topic is out of the scope of this thesis. Reference [4] provides more details about the theorem and its application in RNS.

Example 1.6.

This example shows that the multiplicative inverse exists only if r and m are relatively prime. In Table 1.2., the multiplicative inverse r^{-1} is obtained, if exists, with respect to the modulus m. In the first column, $m = 7$ is always prime with respect to any integer. In the second column, $m = 8$ is not prime with respect to 2, 4, and 6. We notice that 2, 4, and 6 have no multiplicative inverse with respect to modulus 8.

Table 1.2. Multiplicative inverses with respect to two different moduli

r	$m = 7$	$m = 8$
	r^{-1}	r^{-1}
1	1	1
2	4	-
3	5	3
4	2	-
5	3	5
6	6	-
7		7

Division

Division is one of the main obstacles that discourage the use of RNS. In RNS representation, division is not a simple operation. The analogy between division in conventional representation and RNS representation does not hold.

In conventional representation, we represent division as follows:

$$\frac{x}{y} = q \tag{1.16}$$

which can be rewritten as:

$$y \times q = x \tag{1.17}$$

where q is the quotient.

In RNS, the analogous congruence is:

$$y \times q = x \bmod m \tag{1.18}$$

Multiplying both sides by the multiplicative inverse of y, we can write:

$$q = x \times y^{-1} \bmod m \tag{1.19}$$

In Equation (1.19), q is equivalent to the quotient obtained from Equation (1.16) only if it has an integer value. Otherwise, multiplying by the multiplicative inverse in RNS representation will not be equivalent to division in conventional representation.

Example 1.7.

Consider an RNS with $m = 7$, we want to compute the following quotients:

a) $\frac{6}{2}$ b) $\frac{6}{4}$

a) In the first case:

$$\frac{6}{2} = q$$
$$2q = 6 \bmod 7$$
$$q = 6 \times 2^{-1} \bmod 7$$
$$q = 6 \times 4 \bmod 7$$
$$q = 3$$

which is equivalent to division in conventional representation.

a) In the second case:

We know that the quotient in conventional representation is 1, and the result of the division is a non-integer value.

$$\frac{6}{4} = q$$
$$4q = 6 \bmod 7$$
$$q = 6 \times 4^{-1} \bmod 7$$
$$q = 6 \times 2 \bmod 7$$
$$q = 5$$

We notice in part (b) of Example 1.7. that division in RNS is not equivalent to that in conventional representation when the quotient is a non-integer value. Due to this fact, division in RNS is usually done by converting the residues to conventional representation, performing the division, and then converting back to RNS representation. Tedious and complex conversion steps result in undesired overhead. This is one of the main drawbacks of RNS representation.

1.5 Conversion between Conventional Representation and RNS Representation

To utilize the properties of the RNS and carry out the processing in the residue domain, we need to be able to convert smoothly between the conventional (binary or analog) representation

and the RNS representation. The process of conversion from conventional representation to RNS representation is called *Forward Conversion*. Conceptually, this process can be done by dividing the given conventional number by all the moduli and finding the remainders of the divisions. This is the most direct way that can be applied to any general moduli-set. However, we show in Chapter 2 that for some special moduli-sets this process can be further simplified. The simplification arises from the fact that division by a number, that is a power of two, is equivalent to shifting the digits to the right. This property can be utilized to expedite and simplify the forward conversion. The process of conversion from RNS representation to conventional representation is called *Reverse Conversion*. The reverse conversion process is more difficult and introduces more overhead in terms of speed and complexity. The algorithms of reverse conversion are based on Chinese Remainder Theorem (CRT) or Mixed-Radix Conversion (MRC). The use of the CRT allows parallelism in the conversion process implementation. The MRC is an inherently sequential approach. In general, the realization of a VLSI implementation of a reverse converter is complex and costly. More details about CRT and MRC are given in Chapter 2.

1.6 Advantages of RNS Representation

Implementing an algorithm using parallel distributed arithmetic with no dependence between the arithmetic blocks simplifies the overall design and reduces the complexity of the individual building blocks. The advantages of RNS representation can be summarized as follows [4,5,6]:

High Speed: The absence of carry propagation between the arithmetic blocks results in high speed processing. In conventional digital processors, the critical path is associated with the propagation of the carry signal to the last bit (MSB) of the arithmetic unit. Using RNS representation, large words are encoded into small words, which results in critical path minimization.

Reduced Power: Using small arithmetic units in realizing the RNS processor reduces the switching activities in each channel [7]. This results in reduction in the dynamic power, since the dynamic power is directly proportional to switching activities.

Reduced Complexity: Because the RNS representation encodes large numbers into small residues, the complexity of the arithmetic units in each modulo channel is reduced. This facilitates and simplifies the overall design.

Error Detection and Correction: The RNS is a non-positional system with no dependence between its channels. Thus, an error in one channel does not propagate to other channels. Therefore, isolation of the faulty residues allows fault tolerance and facilitates error detection and correction. In fact, the RNS has some embedded error detection and correction features described in [8].

1.7 Drawbacks of RNS Representation

We mentioned that RNS architectures result in great advantages, especially in terms of speed and power. This makes it very suitable to implement RNS in different applications. However, in spite of their great advantages, RNS processors did not find wide use but remained as an interesting theoretical topic. There are two main reasons behind the limited use of RNS in applications:

First, although the RNS representation simplifies and expedites addition and multiplication compared to the conventional binary system, other operations such as division, square-root, sign detection, and comparison are difficult and costly operations in the residue domain. Thus, building an RNS based ALU that is capable of performing the basic arithmetic is not an easy job.

Second, conversion circuitry can be complex and can introduce latency that offsets the speed gained by the RNS processor. Hence, the design of efficient conversion circuits is considered the bottleneck of a successful RNS.

Nevertheless, RNS architectures are considered an interesting theoretical topic for researchers. Some applications that are computationally intensive and require mainly recursive addition and multiplication operations, such as FFT, FIR filters, and public-key cryptography are appealing to be implemented using RNS. Therefore, investigating new conversion schemes can lead to overcoming some obstacles in the RNS implementation of different applications by reducing the overhead of the conversion stages.

1.8 Applications

As discussed in the last section, RNS is suitable for applications in which addition and multiplication are the predominant arithmetic operations. Due to its carry-free property, RNS has good potential in applications where speed and/or power consumption is very critical. In addition, the isolation between the modulo channels facilitates error detection and correction. Examples of these applications are digital signal processing (DSP) [9], digital image processing [10], RSA algorithms [11], communication receivers [12], and fault tolerance [8,13]. In most of these applications, intensive multiply-and-accumulate (MAC) operations are required.

One possible application of RNS in DSP is the design of digital filters. Digital filters have different uses such as interpolation, decimation, equalization, noise reduction, and band splitting [4]. There are two basic types of digital filers: Finite Impulse Response (FIR) filters and Infinite Impulse Response (IIR) filters. Carrying out the required multiplication and addition operations in the residue domain results in speeding up the system and reducing the power consumption [14,15]. Another possible application of RNS in DSP is the Discrete Fourier Transform (DFT) which is a very common transform in various engineering applications. Again, the main operations involved here are addition and multiplication. Using RNS in implementing DFT algorithms results in faster operations due to the parallelism in the processing. In addition, the carry-free property of the RNS makes it potentially very useful in fault tolerant applications. Nowadays, the integrated circuits are very dense, and full testing will no longer be possible. The RNS has no weight information. Therefore, any error in one of the residues does not affect the other modulo channels. Moreover, since ordering is not important in RNS representation, the faulty residues can be discarded and corrected separately. In summary, RNS seems to be good for many applications that are important in modern computing algorithms.

Chapter 2

Conversion between
Binary and RNS Representations

In this chapter, we discuss the conversion between binary and RNS representations. To be able to process the data in RNS, the data has to be first converted to RNS representation. The process of converting the data from conventional representation (analog or binary) to RNS representation is called *Forward Conversion*. Meanwhile, we shall assume that the initial inputs are available in binary representation. We need to utilize efficient algorithms and schemes for the forward conversion process. The forward converter has to be efficient in terms of area, speed, and power. After the data is processed through the modulo processing units of the RNS, they have to be converted back into the conventional representation. The process of converting the data from RNS representation to conventional representation is called *Reverse Conversion*. We present the basic theoretical foundations for the methods of reverse residue-to-binary (R/B) conversion. In addition, we present some architectures for the implementation of these methods. The overhead of the reverse conversion circuitry is the main impediment to build an efficient RNS processor. Particularly, the design of the reverse converter is more important and constitutes the bottleneck of any successful RNS. Therefore, developing efficient algorithms and architectures for reverse conversion is a great challenge and it has received a considerable deal of interest among researchers in the past few decades. In this chapter, we focus on the methods of reverse conversion where the output is in binary representation. However, direct conversion from RNS to analog representation is also based on the same methods. More details about direct residue-to-analog conversion are provided in Chapter 4.

2.1 Forward Conversion from Binary to RNS Representation

The forward conversion stage is of paramount importance as it is considered as an overhead in the overall RNS. Choosing the most appropriate scheme depends heavily on the used moduli-set. Forward converters are usually classified based on the used moduli into two categories. The first category includes forward converters based on arbitrary moduli-sets. These converters are usually built using look-up tables. The second category includes forward converters based on special moduli-sets. The use of special moduli-sets simplifies the forward conversion algorithms and architectures. The special moduli-set converters are usually realized using pure combinational logic.

We present here some of the available architectures for forward conversion from binary to RNS representation. First, we present forward converters based on arbitrary moduli-sets. Then, we present forward conversion based on the special moduli-set $\{2^n - 1, 2^n, 2^n + 1\}$. We show how the complexity of the overall design is minimized which reduces the overhead introduced by the forward converter. Finally, we provide some architectures for implementing the modulo addition that are used in the realization of all forward converters.

2.1.1 Arbitrary Moduli-Set Forward Converters

We present here some architectures for forward conversion from binary to RNS representation using any arbitrary moduli-set. We mentioned earlier that using special moduli-sets, such as $\{2^n - 1, 2^n, 2^n + 1\}$, makes the forward conversion process fast and simple. In general, forward converters based on special moduli-sets are the most efficient available converters. However, some applications require a very large dynamic range which cannot be achieved efficiently using the special moduli-sets. For example, most of the employed moduli-sets consist of three or four moduli. When the required dynamic range is very large, these moduli have to be large, which results in lower performance of the arithmetic units in each modulo channel. In that case, the best solution is to use many small moduli (five or more) to represent the large dynamic range efficiently. The research on representing large dynamic ranges has two main approaches. The first approach is to develop efficient algorithms and schemes for arbitrary moduli-set forward converters. The second approach is to develop new special moduli-sets with a large number of moduli to represent the large dynamic range efficiently. In this approach, a special five-moduli-set $\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1, 2^{n-1} - 1\}$

with its conversion circuits was proposed in [16]. The proposed moduli-set has a dynamic range that can represent $(5n - 1)$ bits while keeping the moduli small enough and the converters efficient. Nevertheless, it is important and useful to keep the research open for both approaches. Therefore, developing efficient schemes for forward conversion from binary to RNS representation using arbitrary moduli-sets is also of great importance.

The implementation of arbitrary moduli-set forward conversion algorithms is either based on look-up tables (typically ROMs), pure combinational logic, or a combination of both. Implementation of these converters using combinational logic is tedious and requires complex processing units. The all ROM implementation is preferred in this case. However, for a large dynamic range, the ROM size grows dramatically and makes the overall conversion process inefficient. A trade-off between the two implementations can be utilized using a combination of ROM and combinational logic [17].

In this section, we provide some basic architectures for arbitrary moduli-set forward converters. We aim at presenting the basic principle of each architecture. More advanced algorithms and architectures are available in [4]. As the look-up table implementation is preferred in the case of the arbitrary moduli-set, we shall focus on this implementation approach and show different techniques to realize it.

The main idea in the look-up table implementation of forward converters is to store all the residues and recall them based on the value of the binary input [18]. The binary input acts as an address decoder input that points at the appropriate value in the look-up table.

To find the residue of a binary number X with respect to a certain modulus m, we utilize the mathematical property of Equation (1.10) to obtain the residues of all required powers of two with respect to modulus m. To illustrate that, assume that X is a binary number:

$$X = x_{n-1}x_{n-2} \dots x_1 x_0 = \sum_{j=0}^{n-1} x_j 2^j \tag{2.1}$$

The residue of X is represented as:

$$|X|_m = \left| \sum_{j=0}^{n-1} x_j 2^j \right|_m \tag{2.2}$$

Using Equation (1.10), we can write:

$$|X|_m = \left| \sum_{j=0}^{n-1} \left| x_j 2^j \right|_m \right|_m \tag{2.3}$$

where x_j is either 0 or 1.

Serial Conversion

A direct implementation of Equation (2.3) is to store all the values $\left|2^j\right|_m$ in a look-up table. The values are activated or deactivated (set to 0) based on whether x_j is 0 or 1, respectively. A modulo-m adder with an accumulator is required to obtain the modulo addition of all activated values in the table. A direct implementation of Equation (2.3) is shown in Figure 2.1.
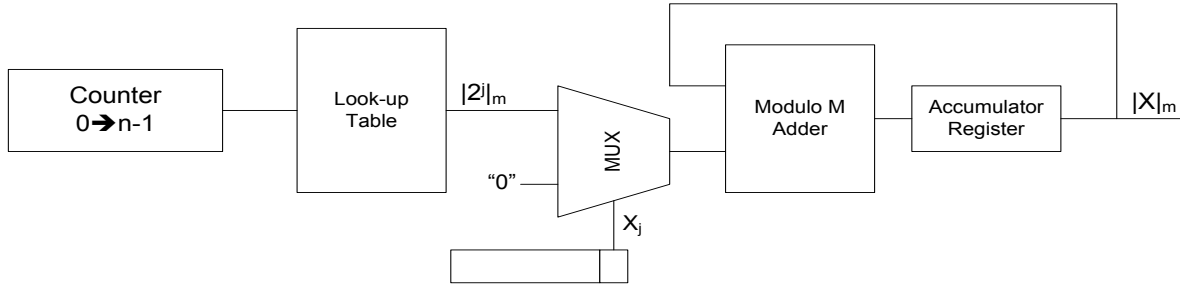
Figure 2.1. Serial forward converter

Initially the accumulator is set to zero. The conversion process requires n clock cycles, where n is the number of bits when X is represented in binary. The value of each bit x_j (either 0 or 1) instructs the multiplexer to accumulate the value $\left|2^j\right|_m$ or a zero. The counter counts from 0 to $n-1$ to address the look-up table. The look-up table is typically implemented as a ROM of size $(n \times \log_2 m)$ bits. The overall design is simple and only few components are required for the implementation. However, the algorithm is completely sequential. This makes it slow and inefficient for large dynamic range applications. Some modifications can be applied on the structure to improve its efficiency. As shown in [4], processing the two values $\left|x_j 2^j\right|_m$ and $\left|x_{j+1} 2^{j+1}\right|_m$ in each cycle doubles the conversion speed. The modified structure is shown in Figure 2.2. Pipelining is also possible in these architectures to increase the throughput.
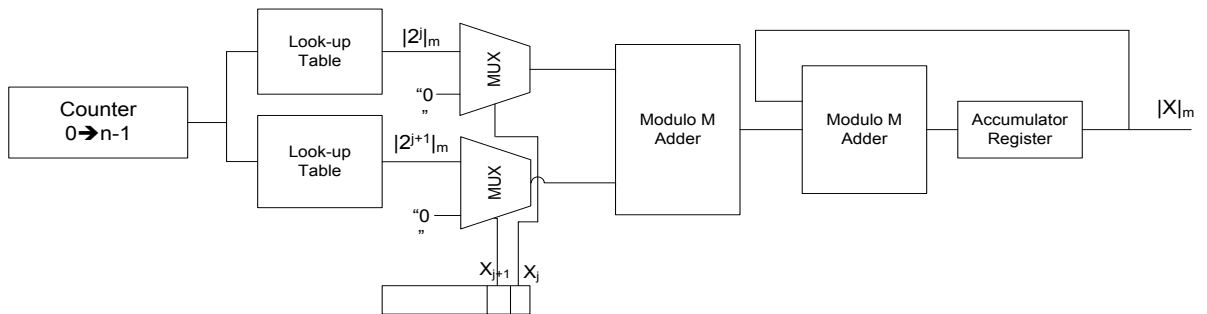
Figure 2.2. Modified structure for serial forward converter

Parallel Conversion

Another architecture for forward conversion from binary to RNS representation can be obtained by manipulating Equation (2.3). Suppose X is partitioned into k blocks, each of p-bits [19]. Let X be partitioned into the blocks $B_{k-1}B_{k-2} \dots B_1B_0$, then:

$$X = \sum_{j=0}^{k-1} 2^{jB} B_j \tag{2.4}$$

$$|X|_m = \left|\sum_{j=0}^{k-1} 2^{jB} B_j\right|_m = \left|\sum_{j=0}^{k-1} \left|2^{jB} B_j\right|_m\right|_m \tag{2.5}$$

Example 2.1.

Consider $X = 2456$ and $m = 19$. We want to find $|X|_m$ by partitioning X into four 3-bit blocks.

First, X is a 12-bit number that has the binary representation: 100110011000.

The four blocks are: 100, 110, 011, and 000. By applying Equation (2.5):

$$|2456|_{19} = ||2^0 \times 0|_{19} + |2^3 \times 3|_{19} + |2^6 \times 6|_{19} + |2^9 \times 4|_{19}|_{19}$$

$$= |0 + 5 + 4 + 15|_{19}$$

$$= 5$$

Equation (2.5) can be directly implemented by storing the values $\left|2^{jB} B_j\right|_m$ in k look-up tables, where k is the number of partitioning blocks. The values of B_j are used to address the values $\left|2^{jB} B_j\right|_m$ in the look-up table (LUT). These values are then added using a multi-operand modulo adder. A typical implementation of Equation (2.5) is shown in Figure 2.3.
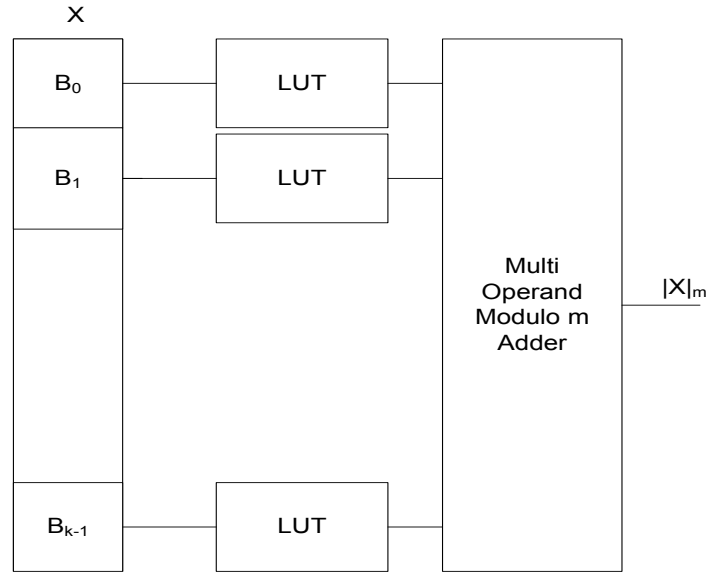
Figure 2.3. Parallel forward converter

Each look-up table (LUT) is a ROM cell that has a size of $(p \times \log_2 m)$ bits, where p is the number of bits in each block, and m is the modulus. Compared to serial forward converters, the parallel forward converters are faster and more adequate for high speed applications. However, the parallel converters require k look-up tables and a modulo adder that adds k operands with respect to modulus m.

In order to reduce the size of each look-up table and therefore enhance the performance of the overall converter, a technique called *periodic partitioning* is utilized [20]. We know from Equation (2.3) that obtaining $|X|_m$ requires storing all the residues $\left|2^j\right|_m$. Careful investigation of the residues of 2^n with respect to modulus m shows that these residues repeat themselves in a period l less than $m-1$ for some moduli. We refer to m-1 as the *basic period*, and to l as the *short period* [4]. The periodicity of the residues $|2^n|_m$ with respect to different moduli is shown in Table 2.1.

Table 2.1. Periodicity of $|2^n|_m$ for different moduli

m	$|2^n|_m$	$m-1$	l	Saving (%)
3	1,2,1,2,1, …	2	2	0 %
5	1,2,4,3,1,2, …	4	4	0 %
6	1,2,4,1,2,…	5	3	40 %
7	1,2,4,1,2,…	6	3	50 %
9	1,2,4,8,7,5,1,2, …	8	6	25 %
10	1,2,4,8,6,2,4,8, …	9	5	44.4 %
11	1,2,4,8,5,10,9,7,3,6,1,2, …	10	10	0 %
12	1,2,4,8,2,4,8,2, …	11	4	63.3 %
13	1,2,4,8,3,6,12,11, …	12	12	0 %
14	1,2,4,8,2,4,8, …	13	4	69.2 %
15	1,2,4,8,1,2,4, …	14	4	71.4 %
17	1,2,4,8,16,15,13,9, …	16	8	50 %
18	1,2,4,8,16,14,10,2,4,8, …	17	7	58.9 %
19	1,2,4,8,16,13,7,14,9,18, …	18	18	0 %
21	1,2,4,8,16,11,1,2,4, …	20	6	70 %

Table 2.1. shows the great saving when we design look-up tables for some values of m. For example, for $m = 15$, we need to store only 4 values. These values can be used for higher indices because of the periodicity of the residues. This results in saving of 71.4 % in the memory size.

2.1.2 Special Moduli-Set Forward Converters

Choosing a special moduli-set is the preferred choice to facilitate and expedite the conversion stages. The special moduli-set forward converters are the most efficient available converters in terms of speed, area, and power. Usually, the special moduli-sets are referred to as *low-cost moduli-sets*. In this section, we will focus on the special moduli-set $\{2^n - 1, 2^n, 2^n + 1\}$ as it is the most commonly used moduli-set.

In contrast to arbitrary moduli-set forward converters, the special moduli-set converters are usually implemented using pure combinational logic. To compute the residue of a number X (in binary representation) with respect to modulus m, we utilize the same principle of Equation (2.3), i.e. evaluate the values $\left|2^j\right|_m$. The only difference here is that m is restricted to 2^n, $2^n - 1$, and $2^n + 1$. We shall derive simple formulas that facilitate the algorithm used to obtain the residues. We show how the residues with respect to the special moduli can be obtained with reduced complexity algorithms and architectures.

Modulus 2^n

Obtaining the residue of X with respect to modulus 2^n is the easiest operation. To understand that, recall that the basic principle in residue computation is division. When the divisor is a power of two (2^n), the division is further simplified to n-bit right shifting. Thus, the residue of X with respect to 2^n is simply the first n least significant bits of the binary representation of X.

Example 2.2.

Let $X = 2456$ which has the 12-bit binary representation: 100110011000. We want to find the residue of X with respect to modulus $m = 2^4 = 16$.

The residue is simply the first four least significant bits of X:

$$|X|_{16} = (1000)_2 = 8$$

Modulus $2^n - 1$

The computation of the residue with respect to modulus $2^n - 1$ is also easy to implement. The only extra overhead is the need for adding an end-around carry in some cases. Many architectures are available to compute the residue with respect to $2^n - 1$ [4,5].

In order to understand the operation of evaluating $|X|_{2^n-1}$, we notice that:

$$|2^n|_{2^n-1} = |(2^n - 1) + 1|_{2^n-1} = ||2^n - 1|_{2^n-1} + |1|_{2^n-1}|_{2^n-1} = |0 + 1|_{2^n-1} = 1 \quad (2.6)$$

where $n > 1$

The same concept can be applied to $|2^{qn}|_{2^n-1}$ where q is an integer:

$$|2^{qn}|_{2^n-1} = \left|\prod_{n=1}^{q}|2^n|_{2^n-1}\right|_{2^n-1} = 1 \quad (2.7)$$

Thus, for $m \neq n$, the residue of 2^m with respect to $2^n - 1$ can be determined as follows:

$$|2^m|_{2^n-1} = |2^{qn+r}|_{2^n-1} = ||2^{qn}|_{2^n-1} + |2^r|_{2^n-1}|_{2^n-1} = |2^r|_{2^n-1} \quad (2.8)$$

where r is the remainder from the division of m by n.

Example 2.3.

Consider $X = 2^9$, and $m = 2^4 - 1$. We want to find the residue of X with respect to m.

Here: $n = 4, m = 9, q = 2$, and $r = 1$.

$$|2^9|_{2^4-1} = ||2^{2\times4}|_{2^4-1} + |2^1|_{2^4-1}|_{2^4-1} = 2$$

Modulus $2^n + 1$

In a similar procedure to modulus $2^n - 1$, we obtain the residue of X with respect to modulus $2^n + 1$ as follows:

First, we notice that:

$$|2^n|_{2^n+1} = |(2^n + 1) - 1|_{2^n+1} = ||2^n + 1|_{2^n+1} - |1|_{2^n-1}|_{2^n+1} = -1 \quad (2.9)$$

Equation (2.9) can be extended for $m \neq n$ and $m = qn + r$, where q is an integer, and r is the remainder from the division of m by n:

$$|2^{qn+r}|_{2^n+1} = ||2^{qn}|_{2^n+1} + |2^r|_{2^n+1}|_{2^n+1} = \begin{cases} 2^r & : q \text{ is even} \\ 2^n + 1 - 2^r & : q \text{ is odd} \end{cases} \quad (2.10)$$

The need for adding $2^n + 1$ where q is odd comes from the fact that $|2^{qn}|_{2^n+1} = -1$ for odd values of q. Therefore, to make the residue positive, we need to add $2^n + 1$.

Example 2.4.

Consider $X = 2^9$, and $m = 2^4 + 1$. We want to find the residue of X with respect to m.

Here: $n = 4$, $m = 9$, $q = 2$ (even), and $r = 1$.

$$|2^9|_{2^4+1} = ||2^{2\times4}|_{2^4+1} + |2^1|_{2^4+1}|_{2^4-1} = 2$$

Example 2.5.

Let $X = 2^{13}$, and $m = 2^4 + 1$. We want to find the residue of X with respect to m.

Here: $n = 4$, $m = 13$, $q = 3$ (odd), and $r = 1$.

$$|2^{13}|_{2^4+1} = ||2^{3\times4}|_{2^4+1} + |2^1|_{2^4+1}|_{2^4+1} = 2^4 + 1 - 2 = 15$$

The Special Moduli-Set $\{2^n - 1, 2^n, 2^n + 1\}$

By making use of the mathematical principles explained above, a general algorithm is presented to convert X (in binary representation) into RNS representation with respect to the special moduli-set $\{2^n - 1, 2^n, 2^n + 1\}$ [4,21,22]. We first partition X into 3 blocks, each of n bits: B_1, B_2, and B_3, where these blocks can be represented as follows:

$$B_1 = \sum_{j=2n}^{3n-1} x_j 2^{j-2n} \tag{2.11}$$

$$B_2 = \sum_{j=n}^{2n-1} x_j 2^{j-n} \tag{2.12}$$

$$B_3 = \sum_{j=0}^{n-1} x_j 2^j \tag{2.13}$$

Thus,

$$X = B_1 2^{2n} + B_2 2^n + B_3 \tag{2.14}$$

The residue r_2 is simply the first n least significant bits, and can be obtained by right shifting X by n-bits.

The residue r_3 is obtained as follows:

$$r_3 = |X|_{2^n+1} = |B_1 2^{2n} + B_2 2^n + B_3|_{2^n+1}$$
$$= ||B_1 2^{2n}|_{2^n+1} + |B_2 2^n|_{2^n+1} + |B_3|_{2^n+1}|_{2^n+1} \tag{2.15}$$

We notice that:

$$|B_1 2^{2n}|_{2^n+1} = ||B_1|_{2^n+1} \times |2^{2n}|_{2^n+1}|_{2^n+1} \tag{2.16}$$

$$|B_2 2^n|_{2^n+1} = ||B_2|_{2^n+1} \times |2^n|_{2^n+1}|_{2^n+1} \tag{2.17}$$

B_1 and B_2 are n-bit numbers. Therefore B_1 and B_2 are always less than $2^n + 1$. The values

$|X|_{2^n+1}$ are obtained as follows:

$$|2^{2n}|_{2^n+1} = ||2^n + 1 - 1|_{2^n+1} \times |2^n + 1 - 1|_{2^n+1}|_{2^n+1} = -1 \times -1 = 1 \qquad (2.18)$$

The value $|2^{2n}|_{2^n+1}$ is obtained as follows:

$$|2^n|_{2^n+1} = |(2^n + 1) - 1|_{2^n+1} = ||2^n + 1|_{2^n+1} - |1|_{2^n+1}|_{2^n+1} = |0 - 1|_{2^n+1} = -1 \quad (2.19)$$

Thus,

$$r_3 = |B_1 - B_2 + B_3|_{2^n+1} \qquad (2.20)$$

In a similar way, the residue r_1 is obtained as follows:

$$r_1 = |X|_{2^n-1} = |B_1 2^{2n} + B_2 2^n + B_3|_{2^n-1}$$
$$= ||B_1 2^{2n}|_{2^n-1} + |B_2 2^n|_{2^n-1} + |B_3|_{2^n-1}|_{2^n-1} \qquad (2.21)$$

We notice that:

$$|B_1 2^{2n}|_{2^n-1} = ||B_1|_{2^n-1} \times |2^{2n}|_{2^n-1}|_{2^n-1} \qquad (2.22)$$

$$|B_2 2^n|_{2^n-1} = ||B_2|_{2^n-1} \times |2^n|_{2^n-1}|_{2^n-1} \qquad (2.23)$$

The values $|X|_{2^n-1}$ are obtained as follows:

$$|2^{2n}|_{2^n-1} = ||2^n - 1 + 1|_{2^n-1} \times |2^n - 1 + 1|_{2^n-1}|_{2^n-1} = 1 \times 1 = 1 \qquad (2.24)$$

The value $|2^{2n}|_{2^n-1}$ is obtained as follows:

$$|2^n|_{2^n-1} = |(2^n - 1) + 1|_{2^n-1} = ||2^n - 1|_{2^n-1} + |1|_{2^n-1}|_{2^n-1} = |0 + 1|_{2^n-1} = 1 \quad (2.25)$$

Thus,

$$r_1 = |B_1 + B_2 + B_3|_{2^n-1} \qquad (2.26)$$

Example 2.6.

Consider the moduli-set $\{15, 16, 17\}$, and $X = 2456 = (100110011000)_2$. We want to find the residues r_1, r_2, and r_3.

First, we need to obtain the blocks B_1, B_2, and B_3 as follows:

$$B_1 = (1001)_2 = 9$$
$$B_2 = (1001)_2 = 9$$
$$B_3 = (1000)_2 = 8$$

Then, we obtain the residues as follows:

$$r_2 = B_3 = (1000)_2 = 8$$
$$r_1 = |9 + 9 + 8|_{15} = 11$$
$$r_3 = |9 - 9 + 8|_{17} = 8$$

Therefore, the RNS representation of $X = 2456$ with respect to the moduli-set $\{15, 16, 17\}$ is $\{11, 8, 8\}$.

A typical architecture for the implementation of a forward converter from binary to RNS representation for the special moduli-set $\{2^n - 1, 2^n, 2^n + 1\}$ is shown in Figure 2.4. The design of modulo adders is briefly described in the next section.
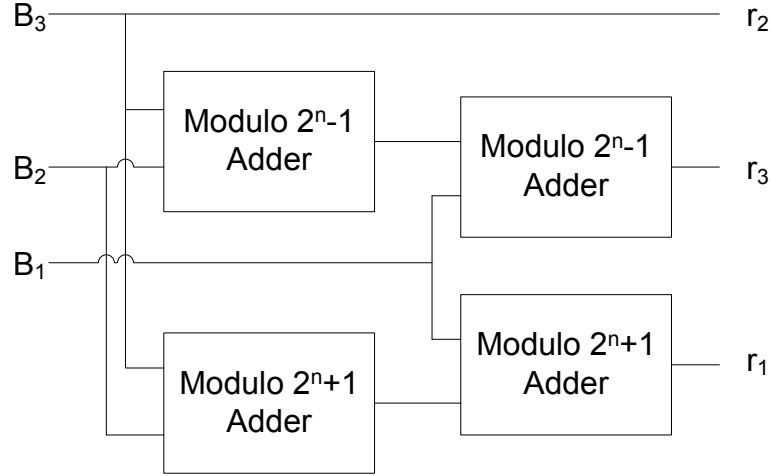
Figure 2.4. $\{2^n - 1, 2^n, 2^n + 1\}$ forward converter

2.1.3 Modulo Addition

In Sections 2.1 and 2.2, we presented some available architectures for the implementation of forward converters from binary to RNS representation. All these architectures, whether they are based on arbitrary moduli or special moduli, require modulo addition in the conversion process. The modulo adder is one of the basic arithmetic units in RNS operations and converters. The performance of the modulo adder is very critical in the design of forward converters from binary to RNS representation. In this section, we provide a brief introduction to the modulo addition operation. We focus on the high-level design of modulo adders. However, the design of the underlying adder is very important in determining the overall performance of the modulo adder. The underlying adder is a conventional binary adder that can have different forms such as ripple-carry adder (RCA), carry-save adder (CSA), carry-lookahead adder (CLA), parallel prefix adder, and so on. Different modulo adders based on different conventional adder topologies are explained in [4] for more advanced details. Here, we restrict ourselves to the basic architectures.

Modulo Adder for an Arbitrary Modulus

For the same word length, a modulo adder is, in general, slower and less efficient than a conventional adder. The basic idea of modulo addition of any two numbers X and Y with respect to an arbitrary modulus m is based on the following relation:

$$|X + Y|_m = \begin{cases} X + Y & : X + Y < m \\ X + Y - m & : X + Y \geq m \end{cases} \tag{2.27}$$

where $0 \leq X, Y < m$.

A typical straightforward implementation of Equation (2.27) is shown in Figure 2.5. The addition of X and Y is performed using a conventional adder. This results in an intermediate value S. Another intermediate value $S - m$ is computed using another conventional adder. Subtracting m is performed easily by adding m's compliment ($\overline{m}$). In binary representation, $\overline{m}$ also represents the value $2^n - m$. If $X + Y < m$, then $X + Y + \overline{m} < 2^n$, and the carry-out ($C_{out}$) is equal to 0. If $X + Y \geq m$, then $X + Y + \overline{m} = (X + Y - m) + 2^n$, and since $X + Y - m \geq 0$, a carry-out propagates in this case. The value of C_{out} instructs the multiplexer (MUX) to select the proper value between S and $S - m$.
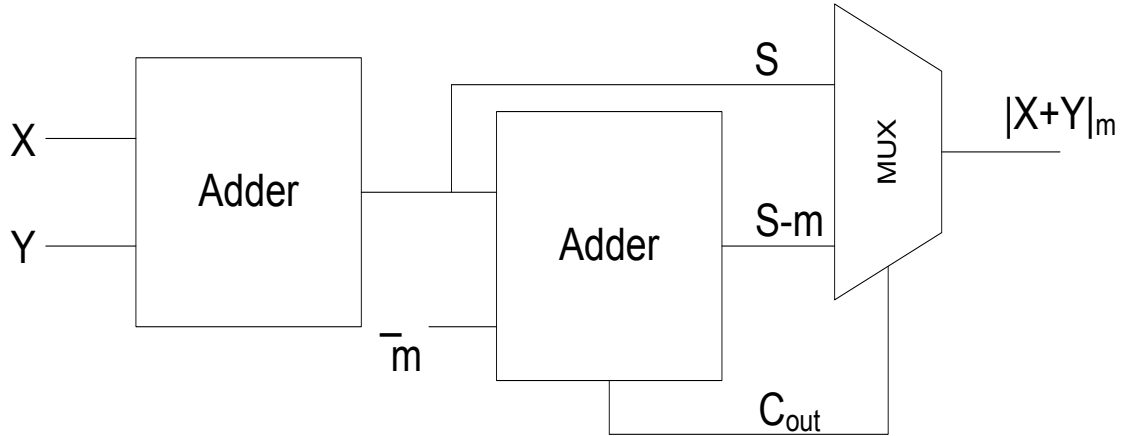

Figure 2.5. Modulo-m adder

Modulo Adder for Special Moduli

The use of some special moduli instead of arbitrary moduli simplifies the design of the modulo adder and makes it more efficient. Here, we present the modulo addition operation for the special moduli: 2^n, $2^n - 1$, and $2^n + 1$. We show some available architectures in the literature for the special moduli modulo adders.

Modulo 2^n Adder

Modulo 2^n addition is the easiest modulo addition operation in the residue domain because it does not require any extra overhead compared to the conventional addition. Modulo 2^n addition of any two numbers X and Y, each of n bits, is done by adding the two numbers using a conventional adder. The result is an $n + 1$ bit output, where the most significant bit is the carry-out. The residue is the first n lowest significant bits, and the final carry-out is neglected. Therefore, modulo 2^n addition is the most efficient modulo addition operation in the residue domain.

Example 2.7.

We want to compute the following modulo additions:

a) $(3 + 4)mod\ (8)$

b) $(5 + 6)mod\ (8)$

Since $8 = 2^3$, the result is simply the least three significant bits of the conventional addition, and the final carry-out is neglected.

a) $(3 + 4)mod\ (8)$ is computed as follows:

$$
\begin{array}{cccc}
 & 0 & 1 & 1 \\
 & 1 & 0 & 0 & + \\
\hline
0 \,|\, 1 & 1 & 1 & = 7
\end{array}
$$

b) $(5 + 6)mod\ (8)$ is computed as follows:

$$
\begin{array}{cccc}
 & 1 & 0 & 1 \\
 & 1 & 1 & 0 & + \\
\hline
1 \,|\, 0 & 1 & 1 & = 3
\end{array}
$$

Modulo $2^n - 1$ Adder

The modulo $2^n - 1$ adder is an important arithmetic unit in RNS because $2^n - 1$ is a commonly used modulus in most special moduli-sets, e.g. $\{2^n - 1, 2^n, 2^n + 1\}$. Some architectures to implement the $2^n - 1$ modulo addition are available in the literature. Here, we shall present the basic idea behind these algorithms and architectures.

To understand the operation of modulo $2^n - 1$ addition of any two numbers X and Y, where

$0 \leq X, Y < m$, we need to distinguish between three different cases:

a) $0 \leq X + Y < 2^n - 1$

b) $X + Y = 2^n - 1$

c) $2^n - 1 < X + Y < 2^{n+1} - 2$

In the first case, the result of the conventional addition is less than the upper limit $2^n - 1$ and no carry-out (C_{out}) is generated at the most significant bit. In this case, the modulo addition of X and Y is equivalent to the conventional addition. In the second case, the result is equal to $2^n - 1$ (i.e. all 1's in binary representation). However, from RNS definition, the result has to be less than $2^n - 1$. In this case, the result should be zero. This case can be detected when all bits of the resulting number are ones (i.e. all $P_i = x_i \oplus y_i$ are ones). Correction is done simply in this case by adding a one and neglecting the carry-out. In the third case, the result of the conventional addition exceeds $2^n - 1$ and a carry-out is generated at the most significant bit. This case is easily detected by the carry-out. Correction is done by ignoring the carry-out (equivalent to subtracting 2^n) and adding 1 to produce the correct result.

Example 2.8.

We want to find the following modulo $2^n - 1$ addition operations. Let $n = 5$, and so the modulus is 31.

a) $(7 + 12) mod (31)$

b) $(15 + 16) mod (31)$

c) $(15 + 18) mod (31)$

In part (a): $7 + 12 = 19 < 31$, therefore no correction needed, and the residue is obtained as follows:

$$
\begin{array}{ccccccc}
 & 0 & 0 & 1 & 1 & 1 & \\
 & 0 & 1 & 1 & 0 & 0 & + \\
\hline
0 & 1 & 0 & 0 & 1 & 1 & = 19
\end{array}
$$

In part (b): $15 + 16 = 31$, then:

$$
\begin{array}{ccccccc}
 & 0 & 1 & 1 & 1 & 1 & \\
 & 1 & 0 & 0 & 0 & 0 & + \\
\hline
0 & 1 & 1 & 1 & 1 & 1 & = 31
\end{array}
$$

Since $P_i = x_i \oplus y_i = 1$ for all all i's, we need to add 1 to the answer and ignore the final carry-out to obtain the desired value.

$$
\begin{array}{ccccccc}
& 1 & 1 & 1 & 1 & 1 & \\
& 0 & 0 & 0 & 0 & 1 & + \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & = 0
\end{array}
$$

In part (c): $15 + 18 = 33 > 31$, then:

$$
\begin{array}{ccccccc}
& 0 & 1 & 1 & 1 & 1 & \\
& 1 & 0 & 0 & 1 & 0 & + \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & = 33
\end{array}
$$

A carry-out is generated which indicates that the result exceeds 31. To correct the result, we ignore the final carry-out and add 1 to the result.

$$
\begin{array}{ccccccc}
& 0 & 0 & 0 & 0 & 1 & \\
& 0 & 0 & 0 & 0 & 1 & + \\
\hline
0 & 0 & 0 & 0 & 1 & 0 & = 2
\end{array}
$$

A possible implementation of modulo $2^n - 1$ adder using ripple-carry adder (RCA) principle is shown in Figure 2.6. Correction is done by feeding 1 into the carry-in (C_{in}) of the first full-adder (FA) if one of the following two cases is detected:

a) $P_i = x_i \oplus y_i = 1$ for all all i's
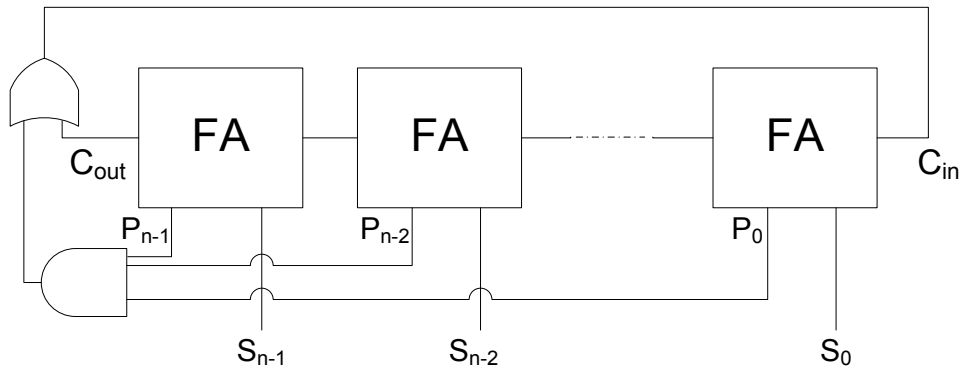b) $C_{out} = 1$

Figure 2.6. Modulo $2^n - 1$ adder

In practice, the architecture in Figure 2.6. suffers from race condition because of the feedback. To avoid that, the operation can be done in two cycles where the intermediate output is latched in the first cycle.

Modulo $2^n + 1$ Adder

The modulo $2^n + 1$ adder is the bottleneck of the design of a forward converter from binary to RNS representation for the special moduli-set $\{2^n - 1, 2^n, 2^n + 1\}$. Its importance arises from the fact that designing an efficient modulo $2^n + 1$ adder is more difficult than that of the other two moduli. This is due to difficulties in detecting when the result is equal to $2^n + 1$ and when it exceeds $2^n + 1$.

In a similar way to that used in modulo $2^n - 1$ addition, three cases have to be distinguished [4]. First, we define Z as follows:

$$Z = X + Y - (2^n + 1) \tag{2.28}$$

Then, we define the three cases as follows:

a) $X + Y \geq 2^n + 1 \ (i.e. Z \geq 0)$

b) $X + Y = 2^n \ (i.e. Z = -1)$

c) $X + Y < 2^n + 1 \ and \ X + Y \neq 2^n \ (i.e. Z < 0 \ but \ Z \neq -1)$

In the first case, $(X + Y)mod(2^n + 1)$ is simply equal to Z. In the second case, $(X + Y)mod(2^n + 1)$ is obtained from Z by setting the most significant bit of Z to 1 and adding 1 to the result. In the third case, Z is negative, and $(X + Y)mod(2^n + 1)$ is obtained from Z by setting the most significant bit to 0 and adding 1 to the result. In summary:

$$|X + Y|_{2^n+1} = \begin{cases} Z & : Z \geq 0 \\ 2^n + |Z + 1|_{2^n} & : Z = -1 \\ |Z + 1|_{2^n} & : otherwise \end{cases} \tag{2.29}$$

Example 2.9.

We want to compute the following modulo $2^n + 1$ addition operations. Let $n = 4$ and so the modulus is $2^n + 1 = 17$.

a) $(7 + 14)mod(17)$

b) $(10 + 6)mod(17)$

c) $(7 + 4)mod(17)$

In part (a): $Z = 7 + 14 - 17 = 4 \geq 0$, then $\qquad$ $(7 + 14) mod(17) = 4$

In part (b): $Z = 10 + 6 - 17 = -1 = (11111)_2$

We set the most significant bit to 1, and add 1 to the result:

$$
\begin{array}{ccccc}
1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 \quad + \\
\hline
\end{array}
$$

$1 \; | \; 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad = 0$

In part (c): $Z = 7 + 4 - 16 = -6 = (11010)_2, \; Z < 0 \; and \; Z \neq -1$

We set the most significant bit to 0: $(01010)_2$, and add 1 to the result:

$$
\begin{array}{ccccc}
0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 \quad + \\
\hline
\end{array}
$$

$0 \; | \; 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad = 11$

A possible architecture for implementing a modulo $2^n + 1$ adder is proposed in [4]. The architecture is shown in Figure 2.7. A carry-save adder (CSA) reduces the three inputs X, Y, and $-(2^n + 1)$ to two: partial sum $(\check{S})$ and partial carry $(\check{C})$. The two values $\check{S}$ and $\check{C}$ are then processed using a parallel-prefix adder. Case (b) is detected if $P_0^n = P_0 P_1 \ldots P_n = 1$. Then, the correction is done by adding P_0^n as an end-around carry and setting $S_n = P_0^n$. Case (c) is detected if C_{n-1} and therefore C_n is 0. The correction is done in this case by adding the inverse of the end-around carry $\overline{C_n}$ and setting S_n to zero.
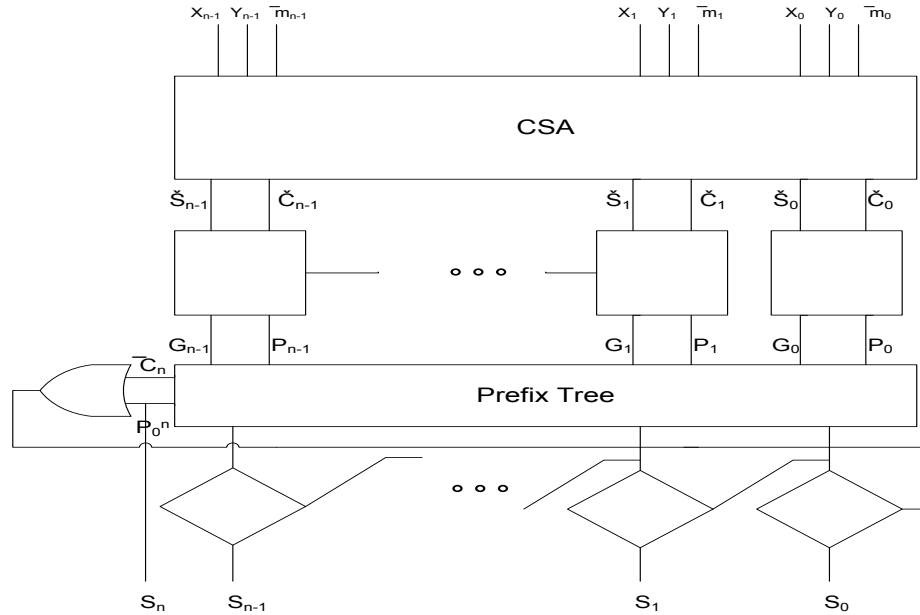

Figure 2.7. Modulo$2^n + 1$ adder

2.2 Reverse Conversion from RNS to Binary Representation

Reverse conversion algorithms in the literature are all based on either Chinese Remainder Theorem (CRT) or Mixed-Radix Conversion (MRC). The MRC is an inherently sequential approach. On the other hand, the CRT can be implemented in parallel. The main drawback of the CRT based R/B reverse converter, is the need of a large modulo adder in the last stage. All the converters proposed in the literature have this problem. The reverse conversion is one of the most difficult RNS operations and has been a major, if not the major, limiting factor to a wider use of RNS [4]. In general, the realization of a VLSI implementation of R/B converters is still complex and costly. Here, we derive the mathematical foundations of the CRT and the MRC, and then we present possible implementations of these methods in reverse conversion.

2.2.1 Chinese Remainder Theorem

The statement of the Chinese Remainder Theorem (CRT) is as follows [4]:

Given a set of pair-wise relatively prime moduli $\{m_1, m_2, \ldots, m_n\}$ and a residue representation $\{r_1, r_2, \ldots, r_n\}$ in that system of some number X, i.e. $r_i = |X|_{m_i}$, that number and its residues are related by the equation:

$$|X|_M = \left| \sum_{i=1}^{n} r_i \left| M_i^{-1} \right|_{m_i} M_i \right|_M \tag{2.30}$$

where M is the product of the m_i's, and $M_i = M/m_i$. If the values involved are constrained so that the final value of X is within the dynamic range, then the modular reduction on the left-hand side can be omitted.

To understand the formulation of Equation (2.30), we rewrite X as:

$$X \triangleq \{r_1, r_2, \ldots, r_n\}$$
$$\triangleq \{r_1, 0, \ldots, 0\} + \{0, r_2, \ldots, 0\} + \ldots + \{0, 0, \ldots, r_n\}$$
$$\triangleq X_1 + X_2 + \ldots + X_n$$

Hence, the reverse conversion process requires finding X_i's. The operation of obtaining each X_i is a reverse conversion process by itself. However, it is much easier than obtaining X.

Consider now that we want to obtain X_i from $\{0, 0, \ldots, r_i, \ldots, 0, 0\}$. Since the residues of X_i are zeros except for r_i. This dictates that X_i is a multiple of m_j where $j \neq i$. Therefore, X_i can be expressed as:

$$X_i \triangleq r_i \times \{0, 0, \ldots, 1, \ldots, 0, 0\} \triangleq r_i \times \widetilde{X_i}$$

where $\widetilde{X}_\iota$ is found such that $\left|\widetilde{X}_\iota\right|_{m_i} = 1$. We recall from Equation (1.15) that the relation between the number r_i and its inverse r_i^{-1} is as follows:

$$(r_i \times r_i^{-1}) \bmod m_i = 1$$

We define M_i as M/m_i, where $M = \prod_{i=1}^{n} m_i$. Then:

$$\left|\left|M_i^{-1}\right|_{m_i} M_i\right|_{m_i} = 1$$

Since all m_i,'s are relatively prime, the inverses exist:

$$\widetilde{X}_\iota = \left|M_i^{-1}\right|_{m_i} M_i$$

and

$$X_i = r_i \widetilde{X}_\iota = r_i \left|M_i^{-1}\right|_{m_i} M_i$$

$$X = \sum_{i=1}^{n} X_i = \sum_{i=1}^{n} r_i \left|M_i^{-1}\right|_{m_i} M_i$$

To ensure that the final value is within the dynamic range, modulo reduction has to be added to both sides of the equation. The result is Equation (2.30).

Example 4.1.

Consider the moduli-set $\{3,4,5\}$. To find the conventional representation of the residue-set $\{2,3,1\}$ with respect to the given moduli-set using the CRT, we first determine M_i's:

$$M_1 = \frac{M}{m_1}$$

$$= \frac{3 \times 4 \times 5}{3}$$

$$= 20$$

$$M_2 = \frac{M}{m_2}$$

$$= \frac{3 \times 4 \times 5}{4}$$

$$= 15$$

$$M_3 = \frac{M}{m_3}$$

$$= \frac{3 \times 4 \times 5}{5}$$

$$= 12$$

and their inverses:

$$|M_1 \times M_1^{-1}|_3 = 1$$
$$|20 \times M_1^{-1}|_3 = 1$$
$$M_1^{-1} = 2$$

Similarly:

$$|M_2 \times M_2^{-1}|_4 = 1$$
$$|15 \times M_2^{-1}|_4 = 1$$
$$M_2^{-1} = 3$$

and:

$$|M_3 \times M_3^{-1}|_5 = 1$$
$$|12 \times M_3^{-1}|_5 = 1$$
$$M_3^{-1} = 3$$

Using Equation (2.30):

$$X = \left| \sum_{i=1}^{3} r_i \left| M_i^{-1} \right|_{m_i} M_i \right|_{60}$$
$$= |2 \times 20 \times 2 + 3 \times 15 \times 3 + 1 \times 12 \times 3|_{60}$$
$$= 11$$

We notice from Equation (2.30) that implementing the CRT requires three main steps:

- Obtaining M_i's and their inverses M_i^{-1}'s.
- Multiply-and-Accumulate operations
- Modular reduction

Since there is no general method to obtain M_i^{-1} using Equation (1.15), the best way to implement it is to save the constants $X_i = \left| M_i^{-1} \right|_{m_i} M_i$ in a ROM. These constants are then multiplied with the residues (r_i) and added using a modulo M adder. This is a straightforward implementation of Equation (2.30). The resulting architecture has two main drawbacks when the dynamic range is large: one, large or many multipliers are required to multiply the constants X_i by the residues; two, a large modulo M adder is required at the final stage. One possible

remedy to obviate the delay and the cost of large or many multipliers is to replace them with ROMs (look-up tables). All possible values of $r_i X_i$ are stored in the ROMs. This solves one of the drawbacks mentioned above. However, the need for a multi-operand modulo M adder at the final stage is inevitable.

The modulo M adder can be realized using ROMs [23], pure combinational logic, or a combination of both. When the dynamic range is large, the speed and the complexity of the multi-operand modulo M adder becomes the bottleneck of the design of the R/B converter. Most of the available CRT based R/B converters have the general high-level block diagram shown in Figure 2.8.
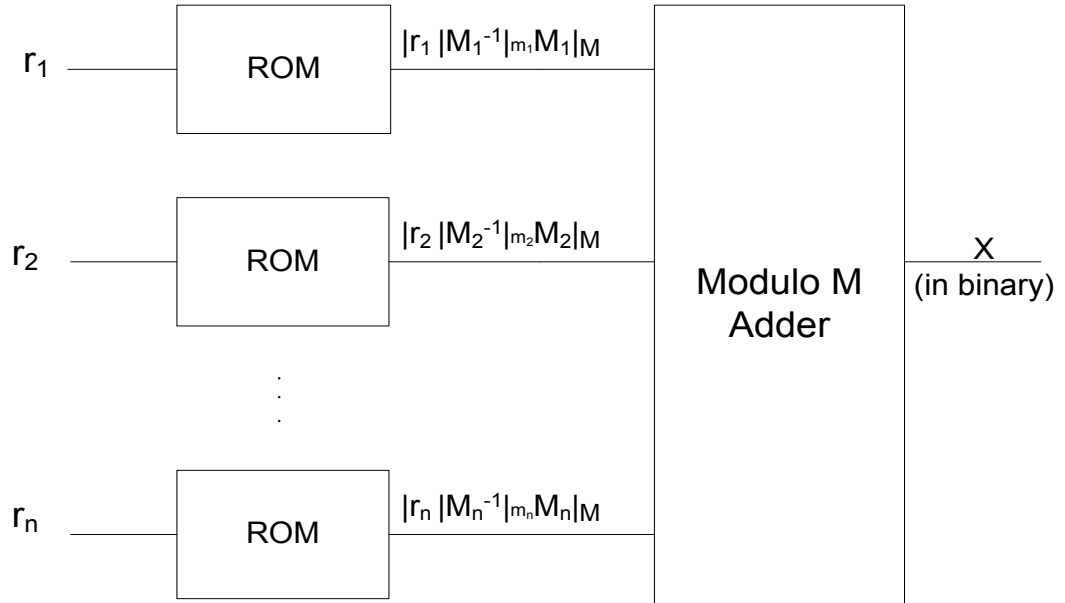
Figure 2.8. CRT based R/B converter

2.2.2 Mixed-Radix Conversion

Given a set of pair-wise relatively prime moduli $\{m_1, m_2, \ldots, m_n\}$ and a residue representation $\{r_1, r_2, \ldots, r_n\}$ in that system of some number X, i.e. $r_i = |X|_{m_i}$, that number X can be uniquely represented in mixed-radix form as [4,24]:

$$X = \{z_1, z_2, \ldots, z_n\}$$

where

$$X = z_1 + z_2 m_1 + z_3 m_2 m_1 + \ldots + z_n m_{n-1} m_{n-2} \ldots m_1 \tag{2.31}$$

and $0 \leq z_i < r_i$.

The Mixed-Radix Conversion (MRC) establishes an association between the unweighted, non-positional RNS and a weighted, positional mixed-radix system. All what is required to perform the reverse conversion is to obtain the values z_i.

The first value z_1 is obtained by applying modulo m_1 reduction on both sides of Equation (2.31):

$$|X|_{m_1} = z_1 = r_1$$

The value z_2 is obtained by rewriting Equation (2.31) as follows:

$$X - z_1 = z_2 m_1 + \ldots + z_n m_{n-1} m_{n-2} \ldots m_1$$

and then applying modulo m_2 reduction on both sides:

$$|X - z_1|_{m_2} = |z_2 m_1|_{m_2}$$

Multiplying both sides by $|m_1^{-1}|_{m_2}$ yields:

$$\left| |m_1^{-1}|_{m_2} (X - z_1) \right|_{m_2} = |z_2|_{m_2} = z_2$$

but:

$$|X - z_1|_{m_2} = \left| |X|_{m_2} - |z_1|_{m_2} \right|_{m_2} = |r_2 - z_1|_{m_2}$$

Therefore,

$$z_2 = \left| |m_1^{-1}|_{m_2} (r_2 - z_1) \right|_{m_2}$$

The value z_3 is obtained in a similar way:

$$z_3 = \left| |(m_2 m_1)^{-1}|_{m_3} (r_3 - (z_2 m_1 + z_1)) \right|_{m_3}$$

In general:

$$z_n = \left| |(m_n \ldots m_2 m_1)^{-1}|_{m_n} (r_n - (r_{n-1} m_{n-2} \ldots z_2 m_1 + z_1)) \right|_{m_n}$$

We notice from the above equations that the MRC is an inherently sequential approach, where obtaining z_i requires generating z_{i-1} first. This is the main drawback of the MRC approach. On the other hand, the CRT allows parallel computation of the partial sums X_i's which results in faster conversion.

Example 4.2.

Consider the moduli-set {3,4,5}. To find the conventional representation of the residue-set {2,3,1} with respect to the given moduli-set using MRC, we determine the required inverses:

First, we determine $|m_1^{-1}|_{m_2}$ as follows:

$$\left|\left|m_1^{-1}\right|_{m_2} m_1\right|_{m_2} = 1$$

$$\left|\left|m_1^{-1}\right|_{m_2} \times 3\right|_4 = 1$$

$$\left|m_1^{-1}\right|_{m_2} = 3$$

Similarly, we determine $\left|(m_2 m_1)^{-1}\right|_{m_3}$:

$$\left|\left|(m_2 m_1)^{-1}\right|_{m_3} (m_2 m_1)\right|_{m_3} = 1$$

$$\left|\left|(m_2 m_1)^{-1}\right|_{m_3} \times 12\right|_5 = 1$$

$$\left|(m_2 m_1)^{-1}\right|_{m_3} = 3$$

The values z_1, z_2, and z_3 are obtained as follows:

$$z_1 = r_1 = 2$$

$$z_2 = \left|\left|m_1^{-1}\right|_{m_2} (r_2 - z_1)\right|_{m_2}$$

$$= |3 \times (3 - 2)|_4$$

$$= 3$$

$$z_3 = \left|\left|(m_2 m_1)^{-1}\right|_{m_3} (r_3 - (z_2 m_1 + z_1))\right|_{m_3}$$

$$= |3 \times (1 - (3 \times 3 + 2))|_5$$

$$= 0$$

Therefore, the number X has the mixed-radix representation:

$$X \triangleq \{2,3,0\}$$

To obtain X in conventional form, we apply Equation (2.31):

$$X = z_1 + z_2 m_1 + z_3 m_2 m_1$$

$$= 2 + 3 \times 3 + 0 \times 4 \times 3$$

$$= 11$$

Figure 2.9. shows one possible implementation of an MRC based R/B converter [4]. Two types of ROMs are used in this realization. The sum addressable ROMs are used to generate the product of the differences and the inverses [4]. The ordinary ROMs are used to generate the products of the moduli and the z_i's. The summation in Equation (2.31) is implemented using carry-save adders (CSAs).
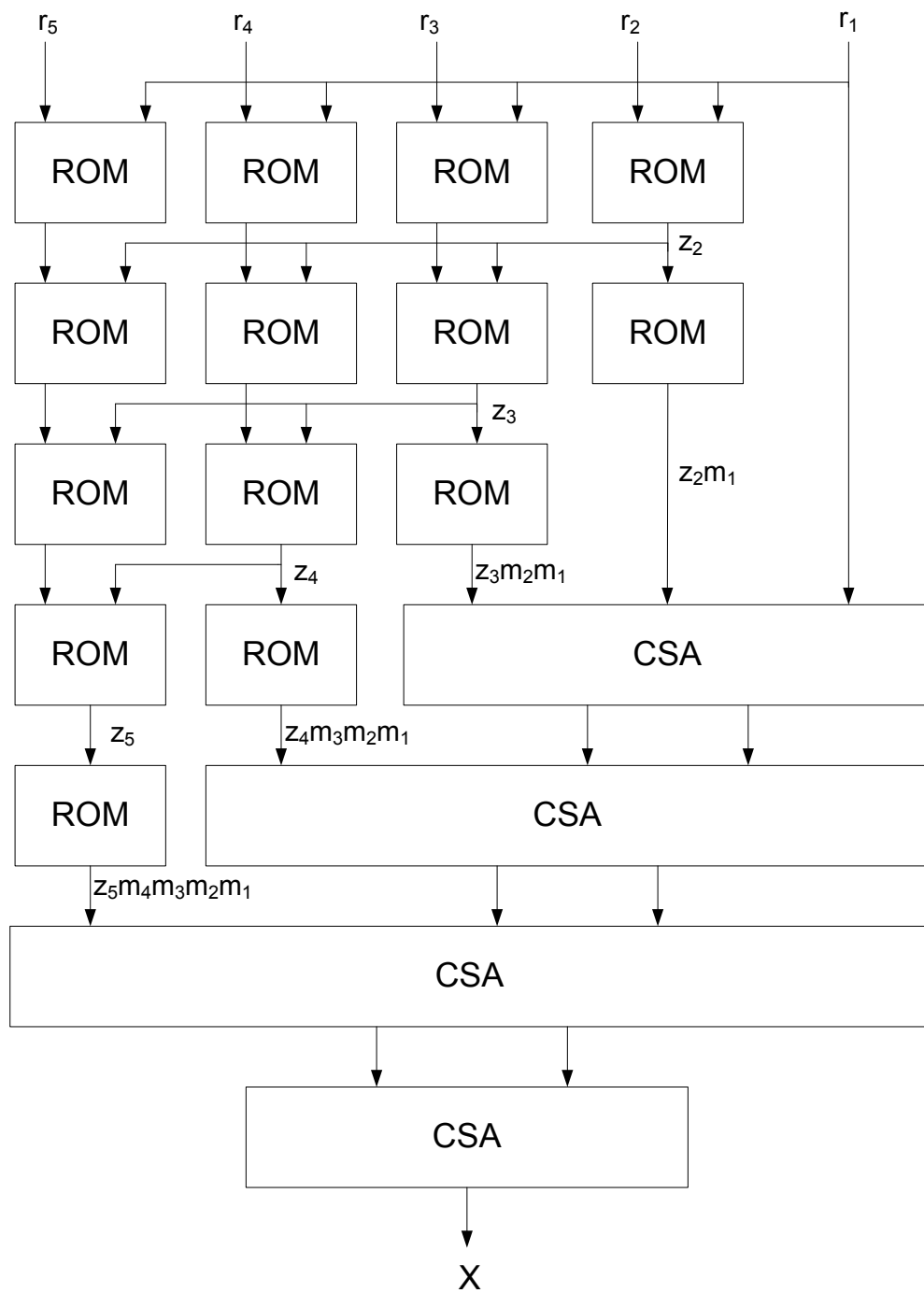
Figure 2.9. MRC based R/B converter (*n*=5)

Chapter 3

Conversion between Analog and Binary Representations

In a typical signal processing system, the analog signal is transformed into digital data represented in binary form. This is done by an analog-to-binary converter, or more often called *analog-to-digital converter (ADC)*. The binary represented data is then processed by the DSP core. The binary output data can be reconverted into analog form using a binary-to-analog converter, or more often called *digital-to-analog converter (DAC)*. To perform the same processing after replacing the DSP core in the system with an RNS based DSP core, we need first to convert the analog signal into binary form using an ADC, and then convert the binary data into RNS representation. In Chapter 4, we show various schemes that overcome this extra overhead and directly convert the analog signal into RNS representation. However, all these schemes adopt similar algorithms and schemes of the available ADCs. Therefore, it is very useful to understand the ADC techniques and architectures. In addition, the DAC is a basic element in the realization of direct reverse converters from RNS to analog representation as shown in Chapter 4. Also, it is used in some ADC architectures. A brief introduction to the available DAC architectures is presented.

Before proceeding to ADC architectures, it is useful to cover the essentials of sampling and quantization processes. A brief introduction to sample/hold (S/H) circuits and quantizers is presented in the next two sections. In the third section, we present some available architectures for real-life quantizers (ADCs). In the fourth section, some available architectures for the implementation of the DAC are presented.

3.1 Sampling

Sampling is the process of obtaining values from a continuous-time signal at fixed intervals. The concept of sampling is illustrated in Figure 3.1. A sample-and-hold (S/H) circuit is used to sample the analog input signal and hold it for quantization by a subsequent circuit. The switch shown turns on and off periodically in a very short time. When the switch is on, the output tracks the input, and when it turns off, the sampled input is stored in the output capacitor. The switch can be implemented as a MOS transmission gate. Practical issues that arise in the implementation of S/H circuits such as delay, glitches, and charge injection are out of the scope of this thesis.
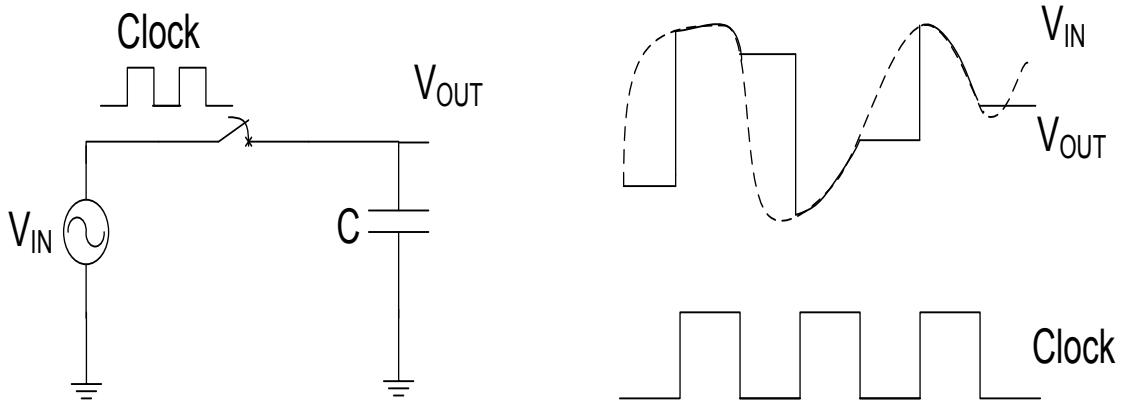
Figure 3.1. Periodic sampling process

The minimum sampling frequency f_s is determined by the Nyquist-Shannon sampling theorem [25]. The theorem states that the minimum sampling frequency required to perfectly reconstruct a bandlimited signal from its samples is $2f_{in,max}$, where $f_{in,max}$ is the highest frequency component in the spectrum of the bandlimited signal. If this condition is not satisfied, some information will be lost due to aliasing. In practice, most of ADCs operate at 3 to 20 times the input signal bandwidth to facilitate the realization of antialiasing and reconstruction fillers [26]. These ADCs are usually referred to as *Nyquist-rate ADCs*. The other category includes ADCs that operate much faster than the Nyquist-rate $2f_{in,max}$ (typically 20 to 512 times faster). These ADCs are referred to as *oversampling ADCs*. In our discussion, we will focus on Nyquist-rate ADCs since they can provide adequate speed for RNS applications compared to oversampling converters.

3.2 Quantization

Quantization is a non-linear process that transforms a continuous range of input samples into a finite set of digital code words. Conceptually, the process of analog-to-digital conversion comprises both sampling and quantization processes. A conventional ADC performs both sampling and quantization. However, the terms quantizer and ADC are often used interchangeably. A quantizer is fully described by its transfer function. The transfer function of a typical quantizer is shown in Figure 3.2. The horizontal axis includes the threshold levels with which the sampled input is compared. The vertical axis includes the digital code representation associated with each output state.
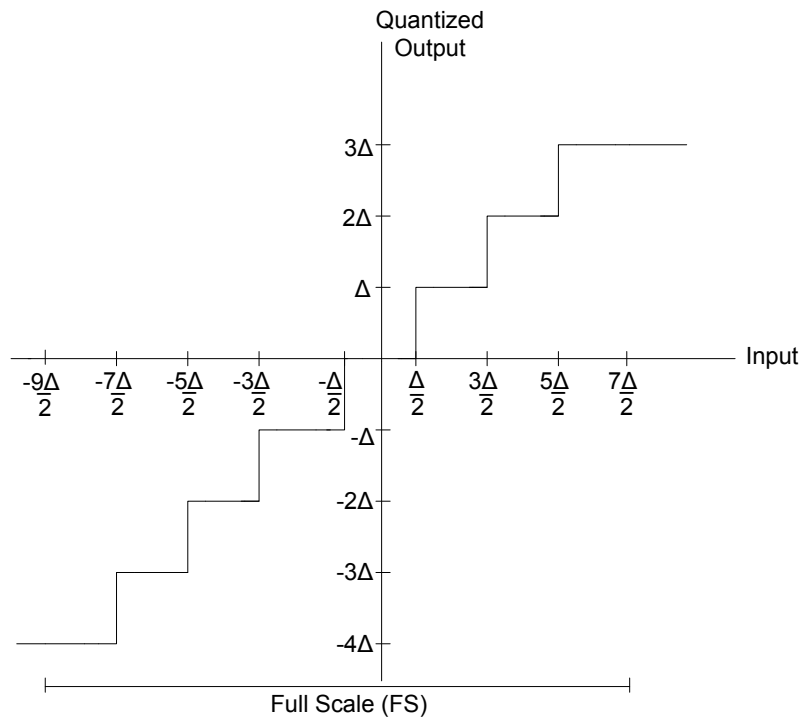

Figure 3.2. Transfer function of a typical quantizer

The analog input voltage has to be within the allowed range of voltages. The allowed voltage range is referred to as the *full scale* (*FS*). If the analog input exceeds the full scale, the quantizer goes into saturation. The difference between the threshold levels is called the *step size* (Δ) and it determines the resolution of the quantizer. The step size of the converter is related to the full scale (*FS*) and the number of representing bits (*n*) by the equation:

$$\Delta = \frac{FS}{2^n}$$

(3.1)

This means that the output digital code changes each time the analog input changes by Δ. The quantizer is a non-linear system. A straight line that represents the relationship between the input and the output in a linear system is replaced by a staircase-like transfer function. The quantizer shown in Figure 3.2 is classified as a midtread uniform quantizer. The quantizers can be divided into two categories based on the locations of the threshold levels: *uniform* and *non-uniform* (Figure 3.3). In uniform quantizers, the threshold levels are evenly distributed, while the thresholds in non-uniform quantizers are non-evenly distributed. Instead, they follow the probability density function (PDF) of the input signal. In our discussion, we will restrict ourselves to uniform quantizers. Based on the existence of an output zero level, the quantizers can be divided into two categories: *midtread* and *midrise* (Figure 3.4). Midtread quantizers include one zero output level. On the other hand, midrise quantizers do not include a zero output level. The transfer function of both midtread and midrise quantizers is odd symmetric about the vertical axis.

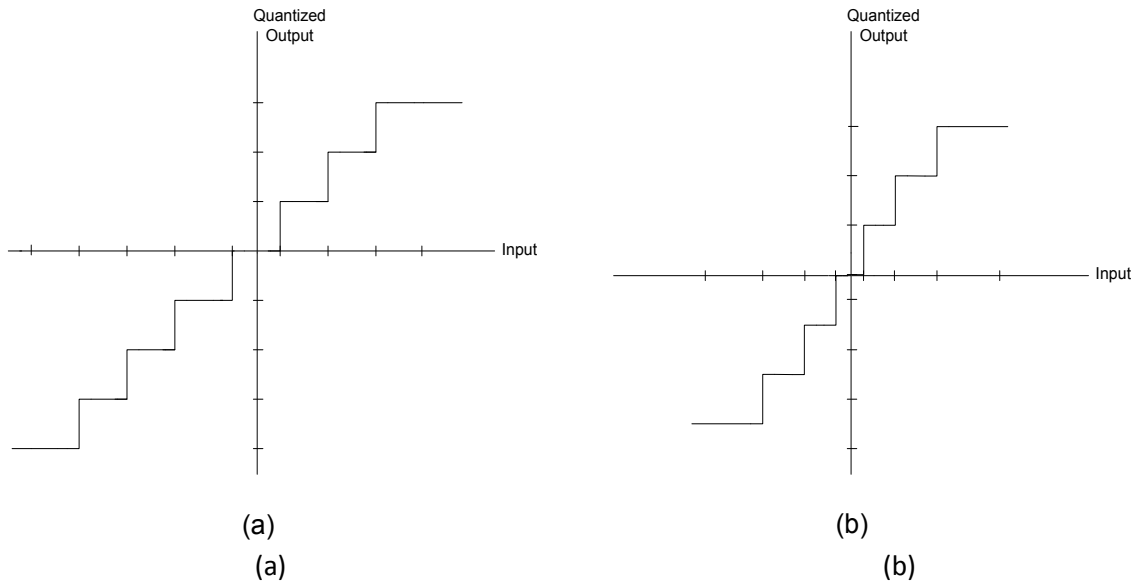

(a)   (b)

Figure 3.3. Quantizer transfer function: (a) uniform (b) non-uniform

All quantizers covered so far are assumed to be ideal. However, real quantizers deviate from the ideal transfer function because of the imperfections in the manufacturing process. These imperfections cause the threshold levels to deviate from their correct locations. In this context, we need to define the commonly used terms to describe some of the performance limitations:
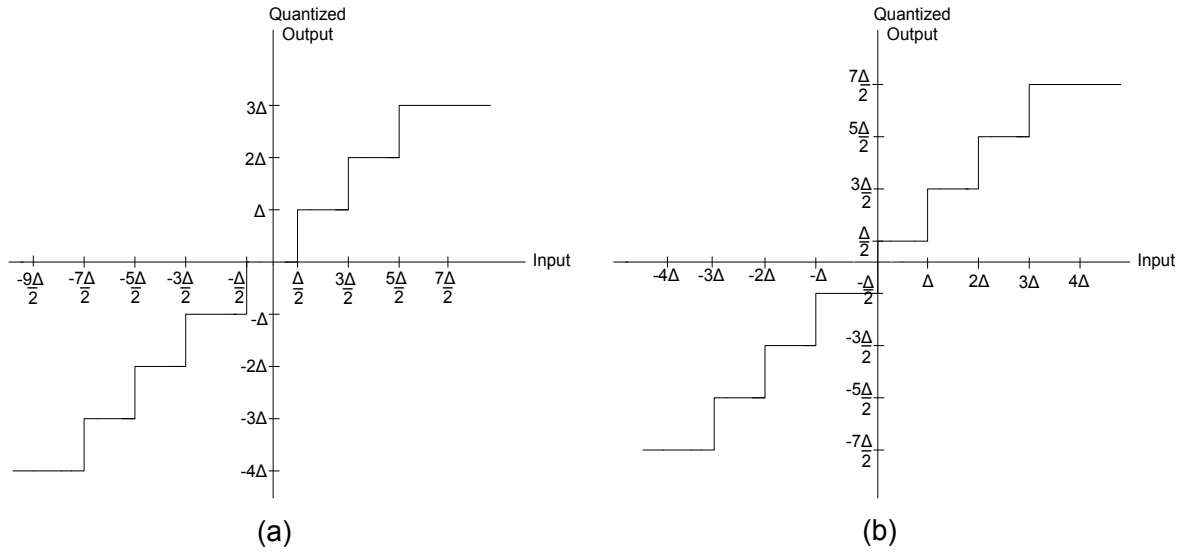
Figure 3.4. Quantizer transfer function: (a) midtread (b) midrise

Offset error

The error that causes all threshold levels to shift from their ideal positions by an equal amount is called an offset error (Figure 3.5). The offset error is the deviation of the actual analog voltage, that ideally corresponds to the level $00\dots01$, from $\frac{1}{2}\Delta$, or in units of Δ:

$$e_{offset} = \frac{V_{00\dots01}}{V_\Delta} \tag{3.2}$$
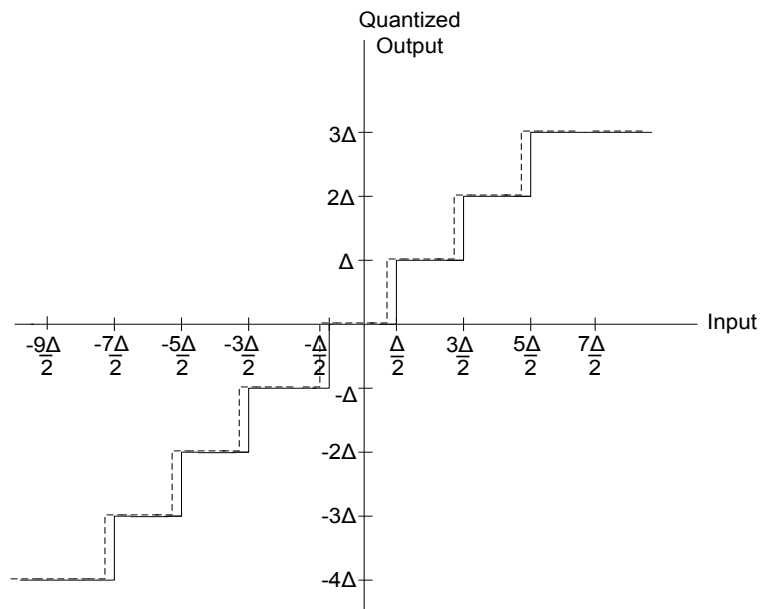

Figure 3.5. Effect of offset error on quantizer transfer function

Gain error

The gain error e_{gain} is the difference between the actual and the ideal quantizer transfer functions at the full scale after the offset error is removed (Figure 3.6). The gain error of an *n*-bit ADC in units of Δ is given by:

$$e_{gain} = \frac{V_{11...11} - V_{11...11}}{V_\Delta} - (2^n - 2) \tag{3.3}$$
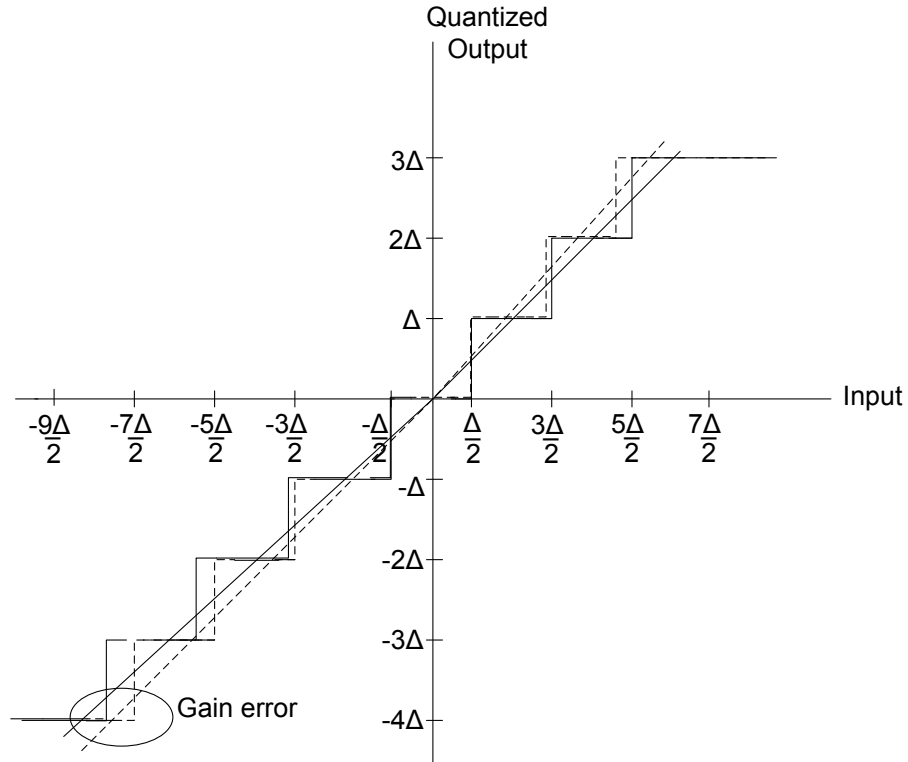

Figure 3.6. Effect of gain error on quantizer transfer function

Non-linearity errors

Non-linearity errors refer to the deviation of the actual transfer function from the straight line after the offset error and the gain error are removed. This is called the *integral non-linearity (INL) error*.

Another term that is used to characterize the non-linearity of the quantizer is the *differential non-linearity (DNL) error*. In an ideal quantizer, the threshold levels are exactly one step size (Δ) apart. DNL is the deviation of the analog step size away from the ideal value (Δ), after the offset error and the gain error are removed. Linearity error effect on the quantizer transfer function is depicted in Figure 3.7.

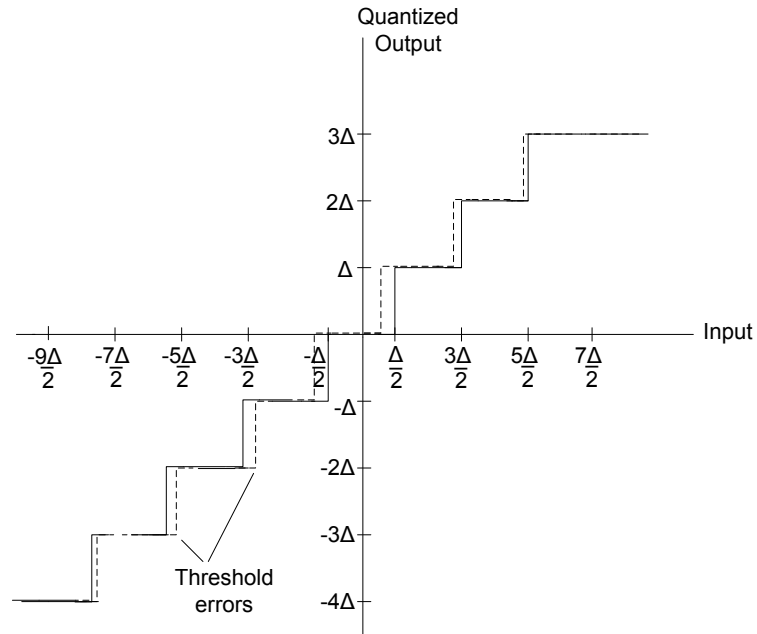Figure 3.7. Effect of linearity error on quantizer transfer function

Missing codes

Missing codes result when a valid output code never occurs because of excessive non-linearity errors. The phenomenon is graphically illustrated in Figure 3.8. An ADC is guaranteed to be missing code free if INL $< \frac{1}{2}\Delta$ or DNL $< \Delta$.
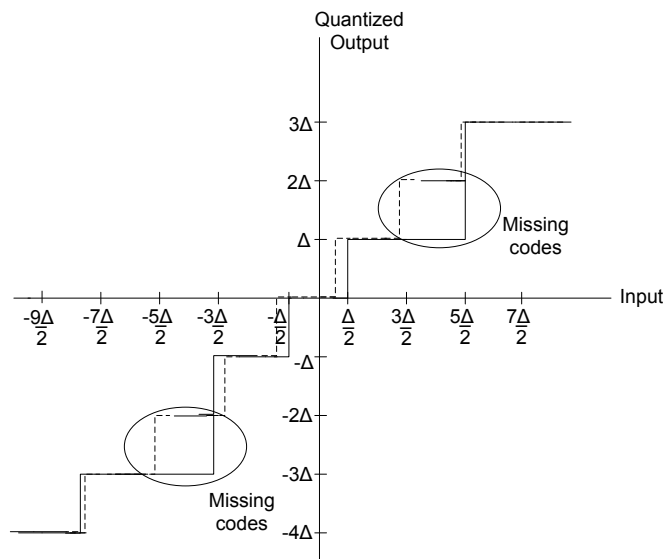

Figure 3.8. Effect of missing codes on quantizer transfer function

Quantization Noise

As mentioned earlier, the quantizer is a non-linear system that assigns a representation level to the sampled input based on its location within the threshold intervals. The *quantized value* $q(x)$ is, in general, different from the input value X. The difference between the two values is referred to as the *quantization error $e(x)$*, where:

$$e(x) = q(x) - x \tag{3.4}$$

If the analog input is guaranteed to be within the full scale, then the quantization error will be in the range:

$$-\frac{\Delta}{2} \le e(x) < \frac{\Delta}{2} \tag{3.5}$$

A simplified statistical model that approximates the quantization error as a random noise component is often used. The statistical representation is based on the following assumptions [25]:

- The quantization error is a sequence of a stationary random process.
- The quantization error is uncorrelated with the sampled input.
- The elements of the quantization error are uncorrelated (white noise process).
- The probability density function (PDF) is uniform over the range $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$.

X ——————→ ▦ ——————→ q(x)

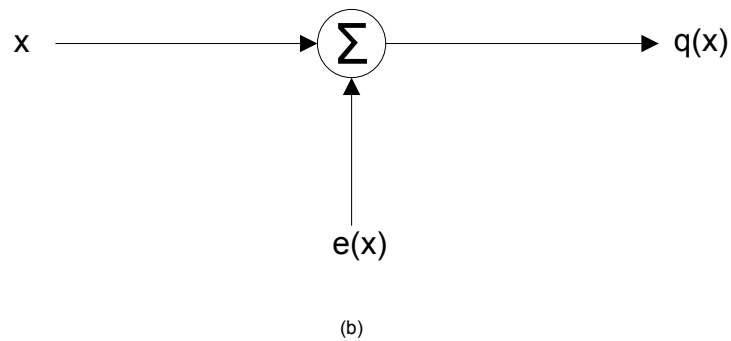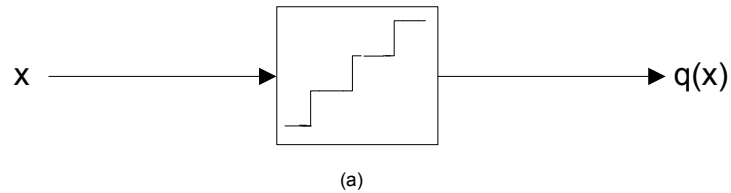(a)

X ——————→ Σ ——————→ q(x)

e(x)

(b)

Figure 3.9. Quantizer models: (a) non-linear (b) linear

These constraints lead to simple and effective analysis of the quantizer performance, where the quantization error is considered as an additive white noise source (Figure 3.9) with uniformly distributed PDF. The probability density function (PDF) of the quantization error based on the above mentioned assumptions is shown in Figure 3.10. The PDF is uniformly distributed over the range $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$. Therefore, the *output noise power* P_e is equivalent to the variance of (x) :

$$P_e = \sigma_e^2 = \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} e^2 \frac{1}{\Delta} \, de = \frac{\Delta^2}{12} \qquad (3.6)$$

The power of a full swing sinusoidal input $x(t) = \frac{FS}{2} \sin(2\pi f_{in} t)$ is given by:

$$P_s = \left(\frac{FS}{2\sqrt{2}}\right)^2 = \frac{FS^2}{8} = \frac{(2^n \Delta)^2}{8} \qquad (3.7)$$

The performance of the quantizer is usually characterized by the ratio between the *output signal power* (P_s) and the *output noise power* (P_e). This ratio is called *signal-to-noise-ratio* (SNR) and often evaluated in decibels. The quantizer SNR is therefore given by:

$$SNR = 10 \log\left(\frac{2^{2n}.\Delta^2/8}{\Delta^2/12}\right) = 6.02n + 1.76 \ dB \qquad (3.8)$$
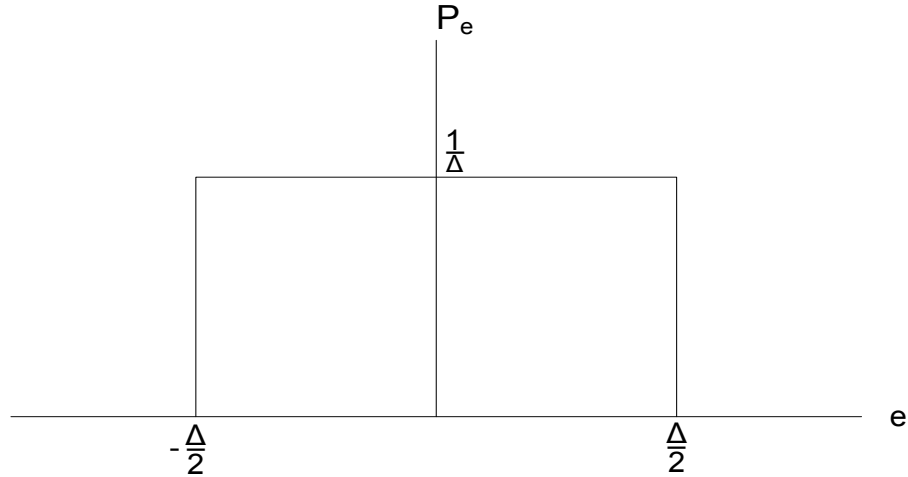

Figure 3.10. Quantizer PDF

The SNR obtained from Equation (3.8) is used to predict the performance of the quantizer. The obtained value represents the maximum SNR of an n-bit quantizer. Usually, the performance of the quantizer is compared to the ideal one by rewriting Equation (3.8). The maximum SNR is replaced by the actual SNR, and Equation (3.8) is solved for the equivalent

resolution (n). The result is called the *Equivalent-Number-Of-Bits* ($ENOB$) and it is commonly used as a figure of merit to evaluate the performance of the quantizer:

$$ENOB = \frac{SNR - 1.76}{6.02} \tag{3.9}$$

In practice, ADCs have different achievable maximum SNR when different inputs are applied in different conditions. It is important here to emphasize that the SNR is evaluated in the above calculations with respect to quantization noise, and other noise sources (such as thermal noise, jitter, etc.) are isolated.

3.3 Analog-to-Digital Converter Architectures

We need to develop efficient architectures for direct conversion from analog to RNS representation. However, we should keep in mind that the residue representation is in digital form. Therefore, investigating some conventional analog-to-binary, or more often called analog-to-digital, conversion schemes and extending their concepts to analog-to-residue conversion is a very interesting approach. In fact, most of the available architectures of direct analog-to-residue converters are based on similar ADC architectures.

In this section, we present a brief comparison among some ADC architectures that are suitable for RNS implementation. The RNS implementation requires high conversion speed and high enough resolution to be partitioned into small residues. Because speed is one of the main objectives of RNS implementation, we shall restrict ourselves to ADCs which have medium-to-high speed.

3.3.1 Flash (or parallel) ADC

Flash ADC is considered the fastest among all analog-to-digital converters. A general architecture of a flash ADC is shown in Figure 3.11. A typical n-bit flash ADC requires 2^n resistors, $2^n - 1$ comparators, and ($2^n - 1$ to n) encoder. Each comparator is connected with one input to the analog sampled input and with the other input to the resistor ladder. The comparators compare the analog input with the threshold levels. The voltages that correspond to the threshold levels are generated using a resistor ladder that contains 2^n resistors. The digital output of the comparator bank is called a *thermometer code*, because as the analog input increases, the comparators turn more outputs into one in a monotonic way (like a mercury-filled

thermometer). The thermometer-coded output is converted into digital binary code using a $(2^n - 1 \text{ to } n)$ digital encoder.

   As mentioned above, the flash ADC is the best choice when high conversion speed is required. The high speed of the flash ADC is due to the fact that it compares the input with all threshold levels simultaneously (in parallel) and produces a digital output. However, flash ADCs suffer from many practical limitations. First, the hardware complexity and, more importantly, the power consumption increase exponentially (2^n) with resolution. Every additional bit doubles both the area and the power consumption. The second drawback is that the large number of comparators connected to the input results in a large capacitive load at the input node [26]. Indeed, this limits the speed of the converter when the targeted resolution is high. The requirement of small comparator offset is another limitation that is difficult to achieve in modern technologies due to process variations. It is very important to maintain the comparator offset less than half a step size to ensure monotonicity of the converter. When high resolution is required, the step size is very small. Usually, an additional circuit is added before the encoder to detect bubble errors and ensure monotonicity. In general, flash ADCs are the optimum for very high speed applications, but the resolution cannot be very high as many limitations make the implementation impractical [27]. Flash topology is very effective for low resolution up to 8 bits [28].
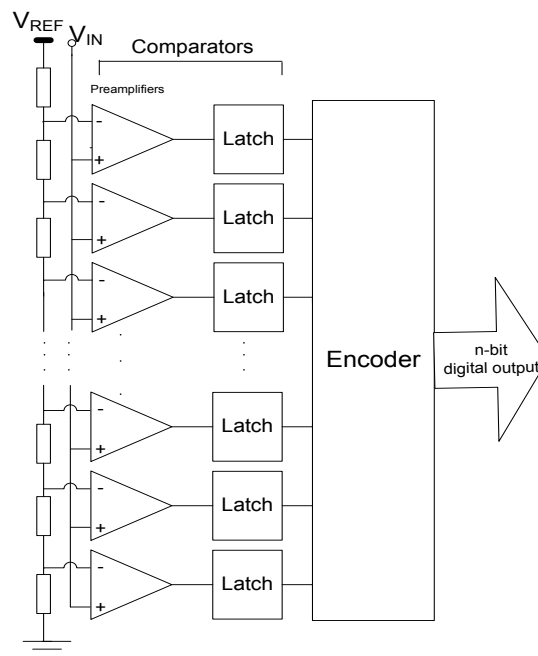

Figure 3.11. Flash ADC

3.3.2 Interpolating Flash ADC

Interpolation is a common technique used to reduce the hardware complexity and improve the performance of flash ADCs. Usually, this technique is used along with folding technique. However, interpolation was used effectively by itself [29]. A 3-bit interpolating flash ADC is shown in Figure 3.12. The main idea behind interpolation is to use the pre-amplifiers as linear amplifiers near the threshold voltages. The inputs to the latches are generated using a resistor ladder. The main advantage of interpolation is the reduction of the number of preamplifiers which results in reduction in the input capacitance. This solves a practical problem in flash ADCs, and slightly reduces the power consumption.
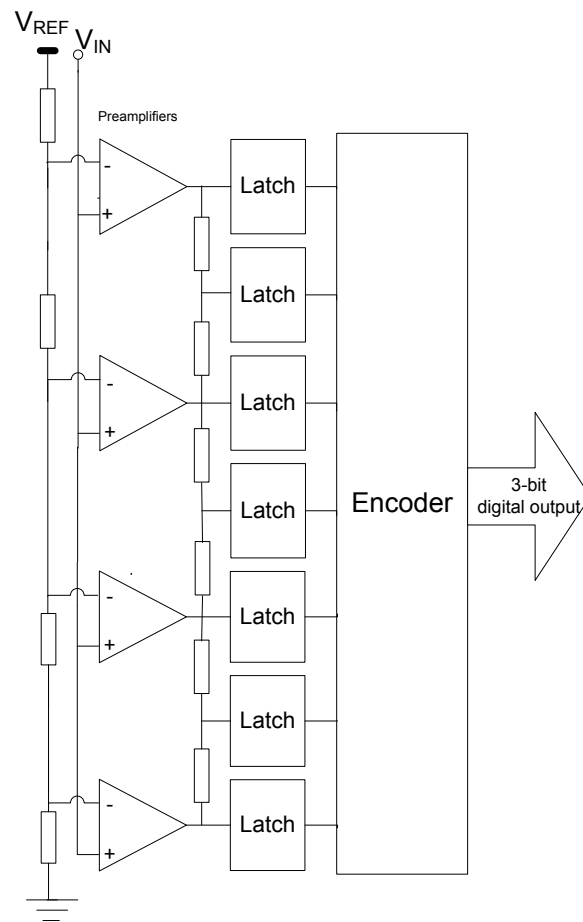

Figure 3.12. A 3-bit interpolating flash ADC

Interpolation can be realized using a resistive ladder as shown in Figure 3.11. Other techniques such as current mirrors [30] and capacitors [31] can be utilized to implement the interpolating technique.

3.3.3 Two-Stage Flash ADC

The two-stage architecture is one of the most popular approaches for high speed, medium resolution ADCs. Figure 3.13. shows the basic architecture of an n-bit two-stage ADC. The S/H circuit samples the analog input. The sampled input is fed to the first n_1-bit flash ADC to obtain the most significant bits (MSBs). The first flash ADC is usually referred to as the *coarse converter*. The output of the coarse converter is converted back into analog using an n_1-bit DAC. This value is subtracted from the sampled input. The residue obtained from subtraction is amplified and quantized using a second n_2-bit flash ADC to obtain the lowest significant bits (LSB). The second flash ADC is referred to as the *fine converter*. The S/H output is held until the completion of the conversion in the fine converter. The overall resolution is n bits where $n = n_1 + n_2$.
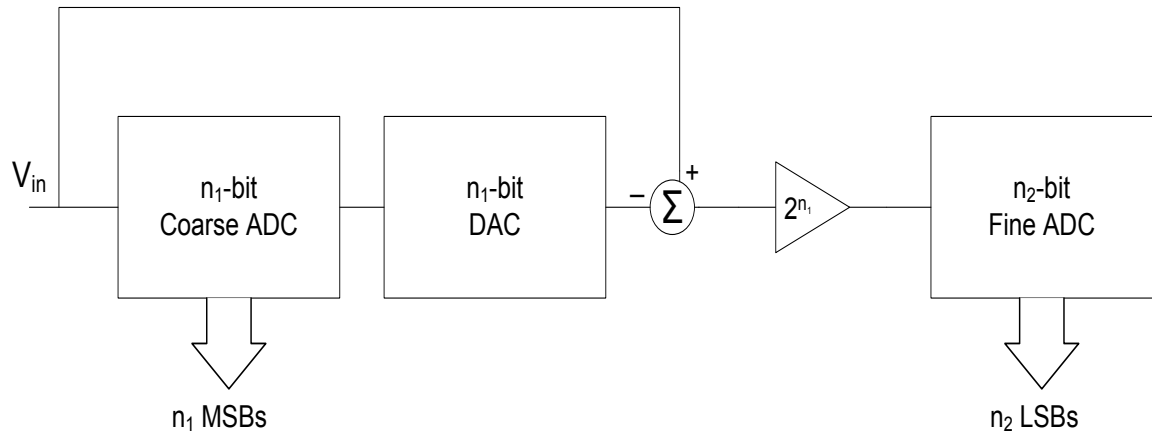

Figure 3.13. Two-stage flash ADC

The two-stage ADC reduces the number of comparators from $2^n - 1$ to $(2^{n_1} - 1) + (2^{n_2} - 1)$. For example, an 8-bit ($n_1 = n_2 = 4$) two-stage flash ADC requires 30 comparators, which is much less than 255 comparators required by an 8-bit full flash ADC. As a result, both the area and the power consumption are reduced. In addition, the input capacitive loading is reduced.

However, there are some drawbacks of the two-stage ADC architecture compared to the flash ADC. First, the two-stage ADC has larger delay due to the additional stage. The second drawback of the n-bit two-stage ADC is the requirement of a DAC whose linearity is better than n bits. In addition, a difference amplifier is required to obtain the residue and amplify it. This difference amplifier also adds to the overall latency of the two-stage ADC. In general, two-stage ADCs have good performance in terms of speed, and medium resolution (10-12 bits) [32, 33].

3.3.4 Multi-Stage Pipelined ADC

The concept of the two-stage ADC can be extended to multi–stage, where a single bit is obtained in each stage. Direct implementation of this concept will suffer from very large delay. Using a sample-and-hold (S/H) circuit in each stage allows pipelining and increases the throughput. A general structure of a pipelined ADC is illustrated in Figure 3.14. The pipelined ADC does not have to wait till the residues ripple through the entire converter. After each stage completes the conversion, it does not sit idle but immediately starts converting the next sample.

The pipelining architecture severely reduces the complexity. The complexity is proportional to n instead of 2^n (as in flash ADC), because each stage needs only 1-bit ADC. However, the gain accuracy of the first residue amplifier becomes more stringent, because the accuracy of conversion for the remaining bits is dependent on the first residue. Nevertheless, very high resolution with high throughput can be achieved by adding a digital-error correction circuit and utilizing trimming or calibration. In summary, pipelined ADCs are very suitable for applications in which high resolution and high throughput are required, while the latency is not critical [34].
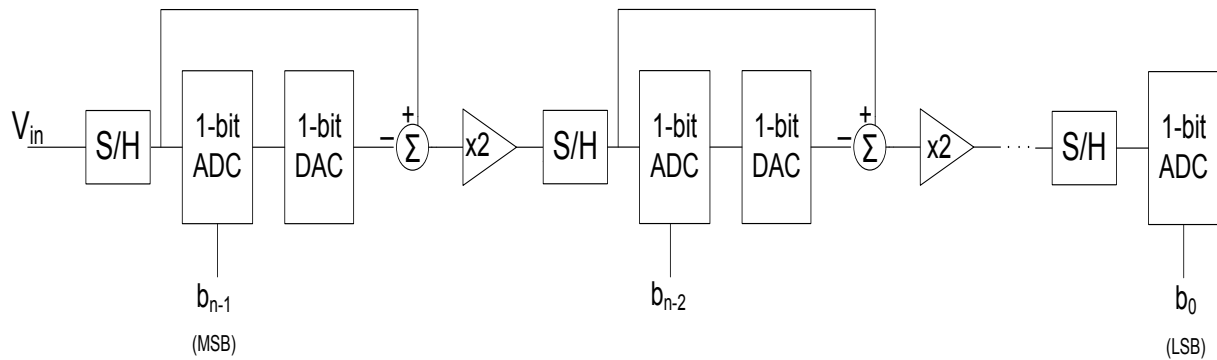
Figure 3.14. Pipelined ADC architecture

3.3.5 Time-Interleaved ADC

The main idea of time-interleaved ADCs is the utilization of parallelism in the conversion process. The general architecture of a three-channel time-interleaved $3n$-bit ADC is shown in Figure 3.15. The three ADCs operate in parallel fashion and time-interleaving manner [26]. The first channel samples the input while the other channels are evaluating the previous samples. The clock $\emptyset_0$ samples the input three times faster than $\emptyset_1, \emptyset_2,$ and $\emptyset_3$. In addition, $\emptyset_1, \emptyset_2,$ and $\emptyset_3$ are delayed with respect to each other by a period $\emptyset_0$. Theoretically, the speed of the overall

three-channel time-interleaved converter is three times the speed of the individual converters. This is a great advantage because the overall speed can be increased by increasing the number of channels, while the area and the power consumption will increase linearly. However, it is very difficult to achieve well matched delayed clocks in practice because of mismatches in the layout of the clock distribution as well as some other noise effects on the clock. Mismatches produce tones at $\frac{1}{N}$ the sampling frequency, where N is the number of channels. This distorts the output spectrum and degrades the performance of the converter.
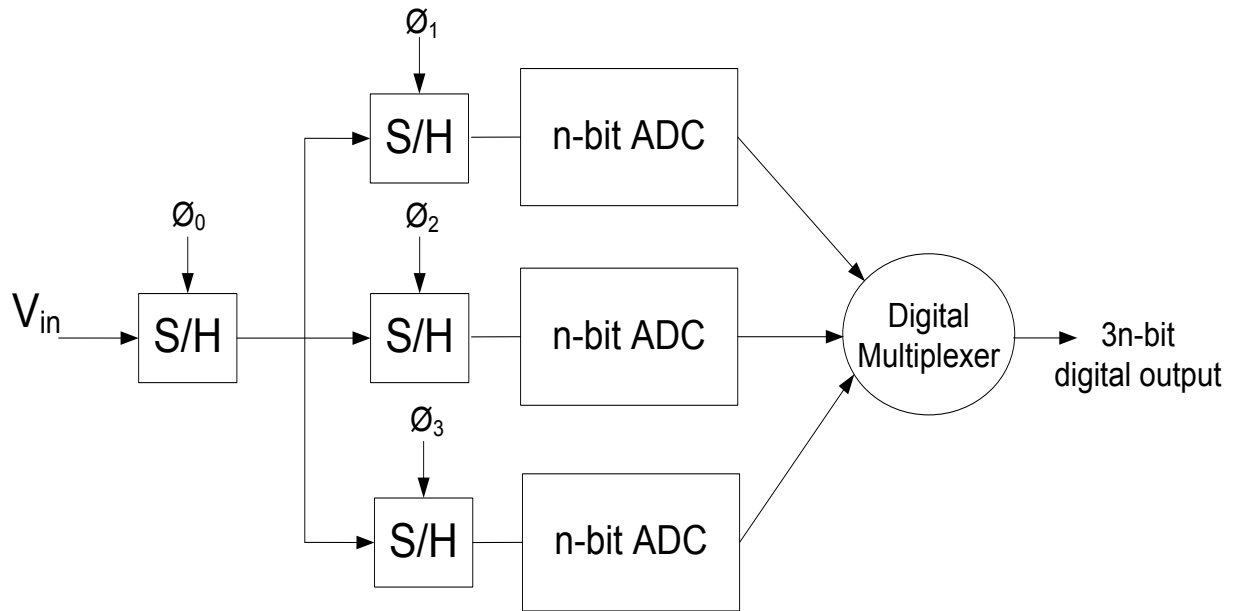
Figure 3.15. A $3n$-bit three-channel time-interleaved ADC architecture

3.3.6 Folding ADC

As mentioned earlier, interpolating is used to reduce the number of preamplifiers and consequently the input capacitive loading. This results in slight reduction in hardware complexity and power consumption. However, the number of latches and the size of the encoder are still proportional to $2^n,$ where n is the number of bits.

Folding is a technique that significantly reduces the number of latches and the size of the encoder along with the number of preamplifiers. Folding and interpolating are often used concurrently for significant reduction in the complexity of the overall architecture. Folding was first introduced by Arbel and Kurz [35] in 1975.

The operation of a folding ADC is similar to that of a two-stage ADC where a coarse converter is used to obtain the MSBs and the remaining LSBs are obtained after the signal is folded. The folding ADC obviates the need for an accurate DAC. Instead, the analog input has to undergo an analog preprocessing stage to fold the signal. A general structure that illustrates the concept of folding is shown in Figure 3.16. The architecture uses a $(\log_2 F)$-bit coarse converter and a $(N - \log_2 F)$-bit fine converter, where F is the *folding factor*. The folding factor F is defined as the number of folding segments in the transfer function of the folding block.
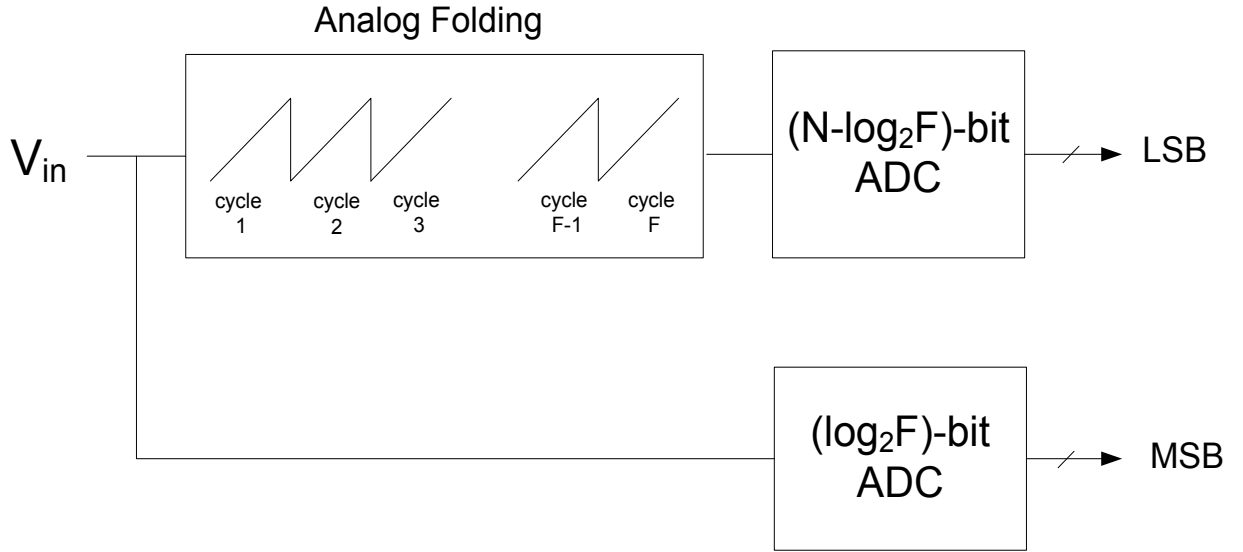
Analog Folding

Figure 3.16. Folding ADC architecture

The complexity of the folding ADC depends on F. The complexity of the coarse quantizer and the folding circuit is proportional to F, while the complexity of the fine quantizer is proportional to $2^n/F$.

3.3.7 Successive Approximation ADC

The successive approximation ADC is widely used in many applications. Its popularity stems from the good ratio of speed/power and the fact that the converter is very compact making it an inexpensive device [36]. The successive approximation ADC has reasonable speed with very good resolution and reduces the complexity and the power consumption of the circuit. A block diagram of a typical successive approximation ADC is shown in Figure 3.17. The basic idea behind the successive approximation is the binary search algorithm. The analog input X is

sampled. A timing control system known as the *successive approximation register (SAR)* is used to control the voltages generated by the DAC. The SAR sets the most significant bit (MSB) to 1 while the remaining bits are maintained at 0. The DAC converts the SAR value into analog and compares it to the analog input. If the analog input voltage is higher than the DAC output, the comparator output is set to 1; if not, the comparator output is set to 0. The SAR retains the MSB bit and proceeds with the next significant bit. The procedure is repeated until the output voltage of the DAC converges to the analog input within a specified accuracy determined by the size of the SAR. The algorithm requires one clock cycle to sample the analog input and one clock cycle to determine each bit. In total, the conversion requires $B + 1$ clock cycles where B is the size of the SAR which is equivalent to the required resolution. The main drawback of the successive approximation technique is its sequential nature which limits the speed when high resolution is also required.
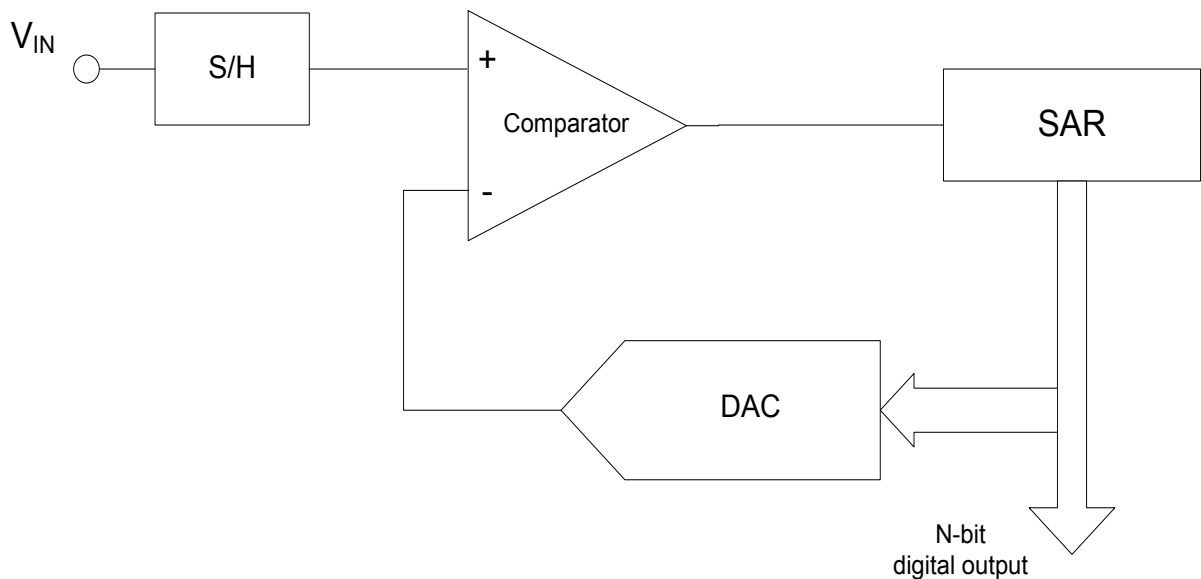

Figure 3.17. Successive approximation ADC architecture

3.3.8 Summary Comparison

Table 3.1. Comparison among the described ADC architectures

Architecture	Advantages	Disadvantages
Flash	- Very fast. - No DAC required.	- Low resolution (8 bits). - Complexity and power increase exponentially with resolution. - High input capacitive loading. - Small comparator offset required.
Interpolating	- Very fast. - No DAC required. - Reduced input capacitive loading.	- Low resolution. - Complexity and power increase exponentially with resolution.
Folding	- Fast. - No DAC required. - Reduced area, power, and input capacitive loading.	- Limited resolution. - Non-idealities in the folding circuits.
Two -Stage	- Reduced area, power, and input capacitive loading. - Possible error correction.	- Moderate speed. - Resolution limited to (10 $-$12 bits).
Pipelined	- High throughput. - Possible error correction. - Reduced input capacitive loading.	- Latency is proportional to the number of stages. - Multiple S/H circuits required.
Time-Interleaved	- High throughput.	- Clock mismatches degrade the performance.
Successive Approximation	- High resolution. - Low input capacitance. -Very low complexity and power consumption.	- Low speed (n cycles) compared to flash ADC. - Accurate DAC required.

3.4 Digital-to-Analog Converter Architectures

We present here some available architectures for the implementation of the DAC. In general, there are three main types of DACs: decoder-based, binary-weighted, and thermometer-code converters. Other hybrid architectures can be realized using any combination of these three types. However, we shall restrict our discussion to the three main types.

The general characteristic equation of an n-bit DAC is given by:

$$V_{out} = V_{ref}(b_1 2^{-1} + b_2 2^{-2} + \ . \ . \ . \ + b_n 2^{-n}) \tag{3.10}$$

3.4.1 Decoder-based DAC

The decoder-based DAC is a straightforward implementation of Equation (3.10). In this approach, an n-bit DAC has 2^n reference signals at its input. Depending on the digital input, only one low impedance path is created, and the corresponding signal passes to the output. Figure 3.18. shows a 3-bit decoder-based DAC. The switches can be implemented using pass transistors or transmission gates.
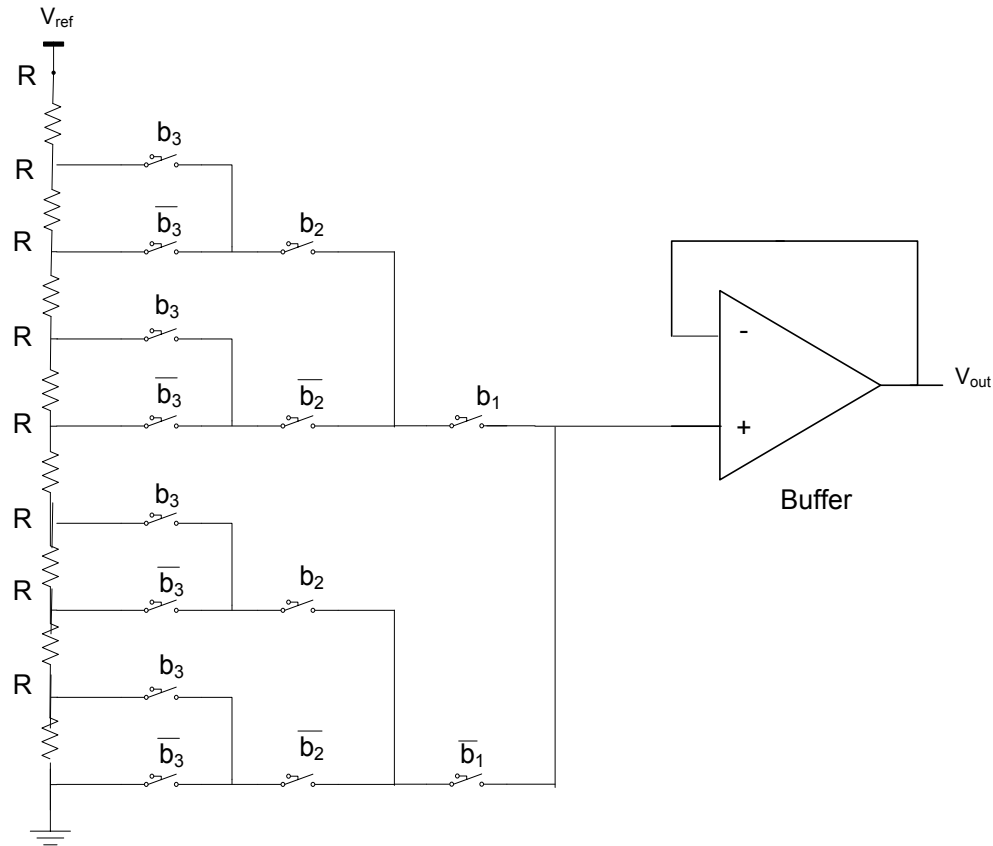

Figure 3.18. A 3-bit decoder-based DAC

Since the voltage at any intermediate node in the resistor ladder must have a lower voltage than all upper nodes, the DAC is guaranteed to be monotonic. Matching between the resistors in the ladder is the most important factor in determining the resolution of the resulting DAC. In terms of speed, the delay through the resistors is a major limitation of the speed. An alternative implementation of the decoder-based DAC for high speed applications is shown in Figure 3.19., where a digital decoder is used to provide a single bus at the input switches. However, the price is larger area and larger capacitive loading at the bus nodes [26].
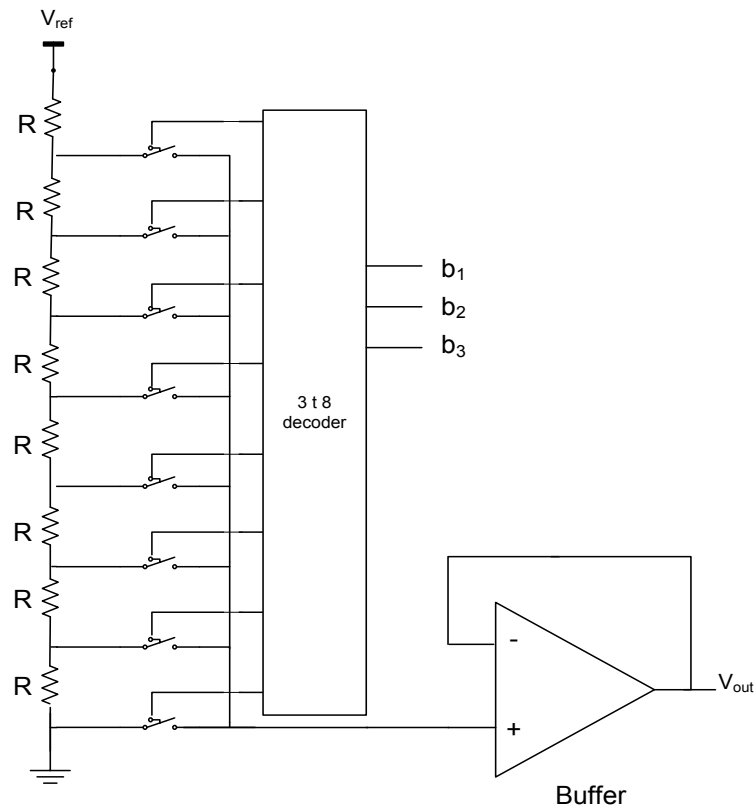
Figure 3.19. An alternative implementation of decoder based DAC

3.4.2 Binary-scaled DAC

The basic idea behind the binary-scaled DAC is to create an array of signals that are scaled in a binary fashion. One possible implementation is the architecture shown in Figure 3.20. where the resistors, and so the currents, are scaled by order of 2^n. However, when n is large, the resistors and the currents can be large which limits the performance. This architecture is not adequate for high speed applications because of the glitches at the output and non guaranteed monotonicity.
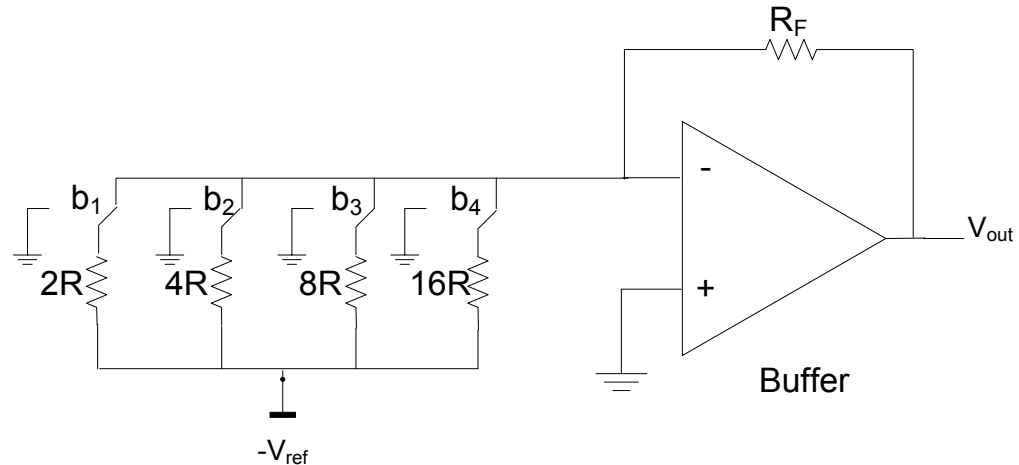
Figure 3.20. A 4-bit binary-weighted DAC

A better implementation of the binary-scaled DAC is the R-2R ladder architecture. The architecture is shown in Figure 3.21. In this architecture, the currents are scaled in a binary fashion while maintaining small values of the resistors. The resulting DAC has better accuracy and smaller size compared to the architecture shown in Figure 3.20. In addition, matching the resistors is easier in this case, since only two resistor values are required (R and 2R).
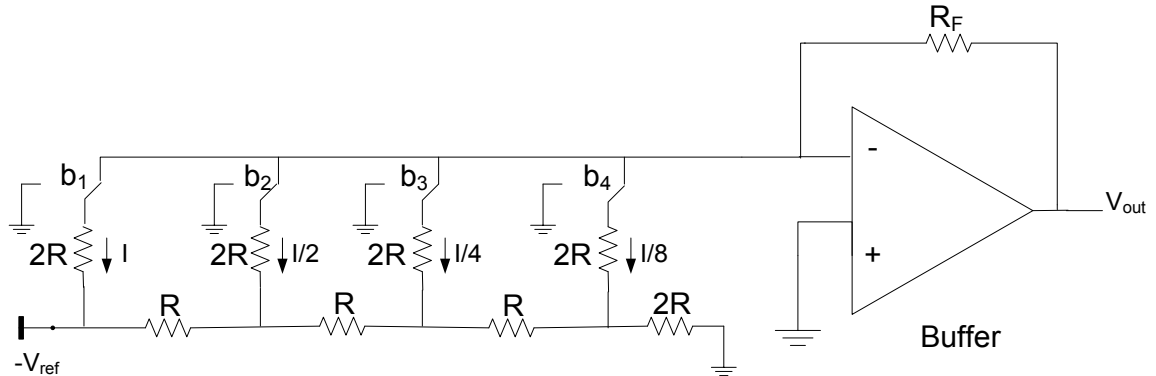
Figure 3.21. A 4-bit R-2R DAC

3.4.3 Thermometer-code DAC

An n-bit thermometer-code DAC requires 2^n input switches to fully represent the input data. This is done by encoding the binary input into thermometer-code using a conversion circuit. The area of the resulting circuit can be large. However, the thermometer-code DAC has better linearity performance and reduces the glitches at the output. A 3-bit thermometer-code DAC architecture is shown in Figure 3.22.
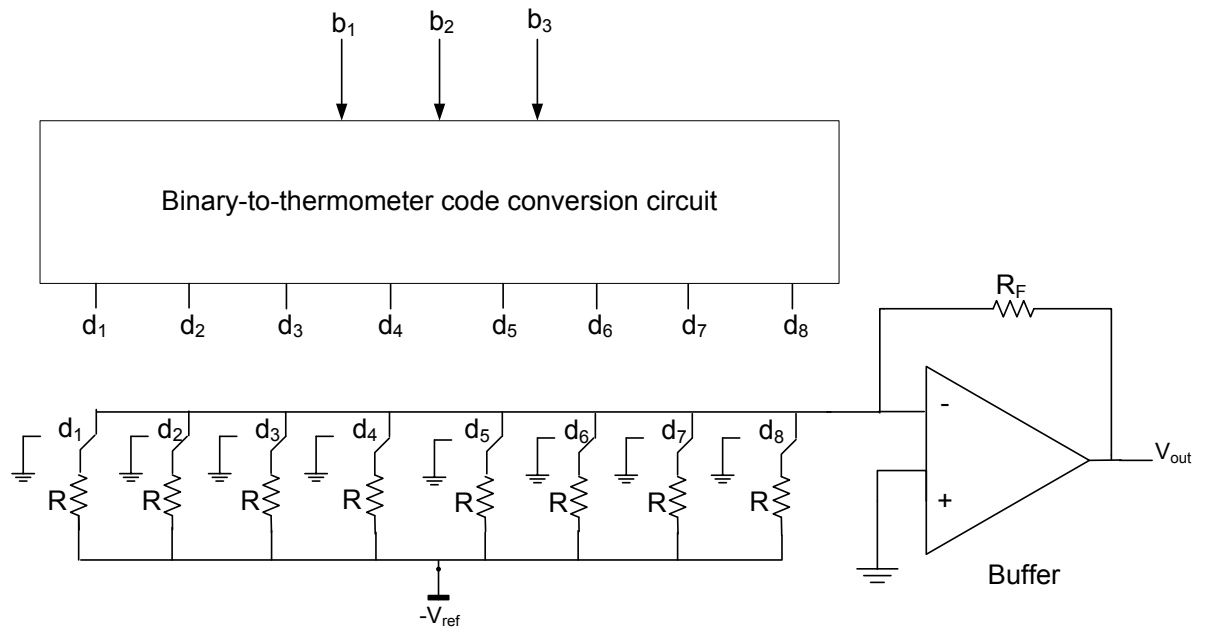
Figure 3.22. A 3-bit thermometer-code DAC

Chapter 4

Conversion between Analog and RNS Representations

In this chapter, we discuss the direct conversion between analog representation and RNS representation. In Chapter 2, we assumed that the available data is already sampled, quantized, and in binary format. However, in real-time applications, the interaction with the real analog world requires converting the continuous-time analog signal into residue representation and vice versa. Usually, this is done in two stages. For example, to convert an analog signal into residue form, the analog signal is first sampled and quantized using an ADC. Next, the binary represented data is converted into residue representation using one of the proposed B/R forward conversion schemes in Chapter 2. This makes the conversion inefficient due to the increased latency and complexity. In order to utilize an RNS-based processor (Figure 1.1.) in a certain application, we need to develop conversion circuits that perform as efficient as the analog-to-digital converter (ADC) and the digital-to-analog converter (DAC) in digital binary systems. Thus, direct conversion from analog-to-residue (A/R) and from residue-to-analog (A/R) is sought to eliminate the intermediate binary stage delay and improve the efficiency of the overall RNS.

This chapter consists of two main sections. In the first section, we discuss the process of direct A/R conversion and present some proposed schemes and architectures. In the second section, we discuss the process of direct R/A conversion and present possible implementations of the available algorithms.

4.1. Forward Conversion from Analog to RNS Representation

In this section, in addition to the literature review of some proposed direct analog-to-residue (A/R) conversion schemes, we present two schemes proposed by us. The A/R conversion is necessary when interaction with the real analog world is required. Some researchers have worked on that problem and proposed various architectures [37-42]. The proposed architectures extend the principles of the conventional ADCs to A/R converters. Therefore, we shall refer to Section 3.3. whenever further explanation is needed. Explanation and analysis to demonstrate the efficiency of the proposed schemes are provided. The proposed architectures are compared to both their analog-to-digital counterpart converters and to their similar A/R architectures proposed in previous work. The main A/R converters types are the following:

4.1.1 Flash A/R Converter

The flash principle described in Section 3.3. can be applied to A/R converters. The flash A/R converter described in [37] and [38] uses the same number of comparators and resistors. The proposed n-bit flash A/R converter requires $2^n - 1$ comparators and 2^n resistors. The only modification is that the thermometer code is converted into residue representation instead of binary representation. To do that, we invoke to the base value definition described in Section 1.2. The dynamic range M is partitioned into groups of m_r integer numbers, where m_r is the largest modulus. The $(2^n - 1$ to $n)$ encoder in an n-bit flash ADC is replaced with the encoder shown in Figure 4.1. The proposed encoder converts the thermometer-coded output of the comparator bank into residue form. The first step of the conversion process is to obtain the base value that corresponds to the sampled input X. The function E_i is defined as:

$$E_i(G_i) = G_i \oplus G_{i+1} \tag{4.1}$$

where

$$G_i = \begin{cases} 1 & : X \geq B_i \\ 0 & : otherwise \end{cases} \tag{4.2}$$

Equation (4.1) can be implemented using XOR gates. This array of XOR gates enables the buffer that corresponds to the base value (B_i). The residue is obtained using a set of M/m_r buffers. Since the XOR gate output will be zero for $X = 0$, a NAND gate is used for the first buffer enable. For any input $X \in M$, only one XOR output will be asserted and the function E_i will enable only the buffer that corresponds to the base value of X. Thus, the residue of X with

respect to m_r is the number of 1's beyond the base value, i.e. $|X|_{m_r} = X - B_r$. The output of the buffer that corresponds to the base value drives a PLA of size $m_r \times \log_2(m_r - 1)$ bits, whose output is the digital binary representation of $|X|_{m_r}$ [38].

To obtain all the residues with respect to different moduli, we notice that any number X has unique representation in its range. Therefore, the knowledge of the base value (B_r) along with the residue $|X|_{m_r}$ is sufficient to uniquely identify the other residues. The PLA size, in this case, has to be modified to $(m_i + l) \times \sum_{i=1}^{n}\lceil\log_2(m_i - 1)\rceil$ bits, where $l = \lceil\log_2(m_r - 1)\rceil$ represents the extra bits to be added [39].
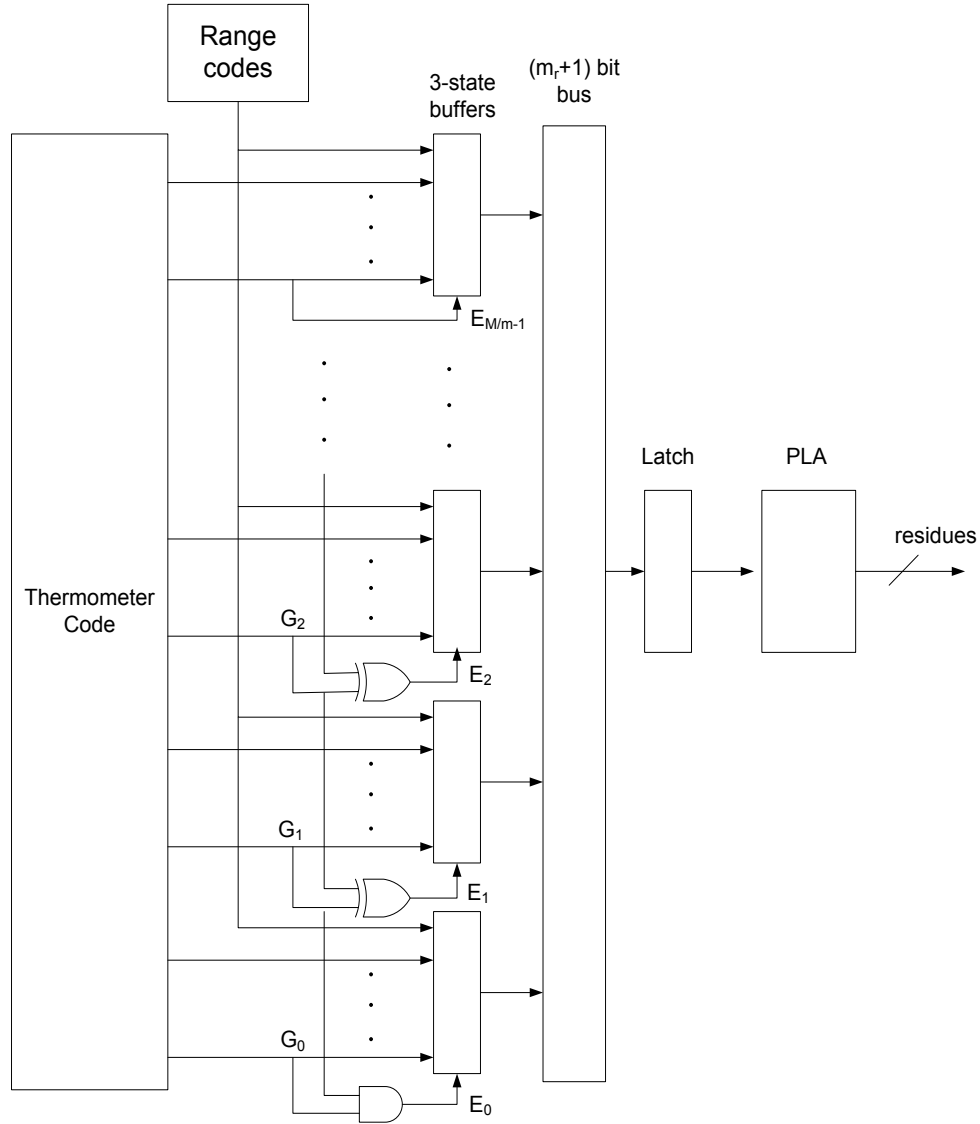

Figure 4.1. Conversion from thermometer code to residue

The latency of the proposed flash A/R converter is:

$$t_{A/R} = t_{comparator} + t_{XOR} + t_{buffer} + t_{PLA} \tag{4.3}$$

On the other hand, a similar flash ADC will have a latency given by:

$$t_{ADC} = t_{comparator} + t_{ROM} \tag{4.4}$$

where t_{ROM} is the delay of the ROM encoder used to convert the thermometer code into binary code in a flash ADC.

For large dynamic ranges: $t_{XOR} + t_{buffer} + t_{PLA} \ll t_{ROM}$. Therefore, a flash A/R converter can be even faster than its ADC counterpart. However, the proposed converter does not solve any of the practical limitations of flash converters discussed in Section 3.3. The resolution of the proposed converter is limited due to the exponential increase in hardware complexity and power consumption. In addition, the input capacitance is large for high resolution, and offset mismatches are inevitable.

An iterative technique applied to flash principle has been reported in [39] and [40] to reduce its hardware complexity. The proposed architecture consists of two stages, where the first stage generates the base value, and the second stage generates the residue. The proposed architecture is shown in Figure 4.2.
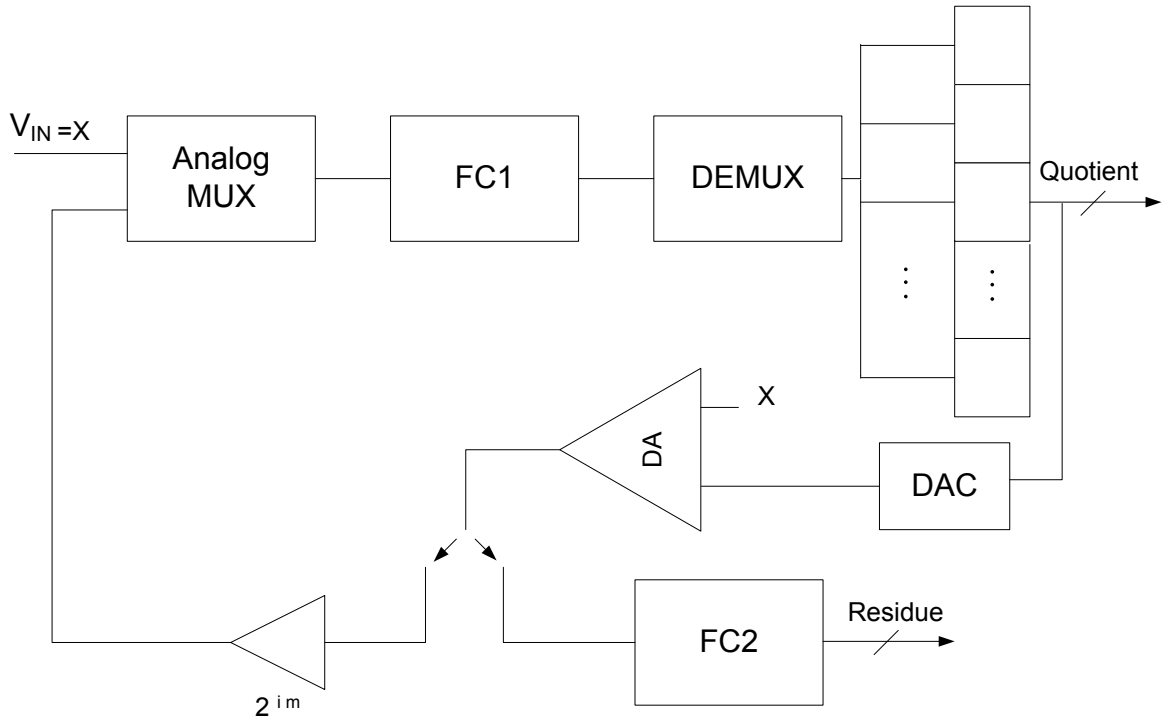

Figure 4.2. Iterative flash A/R converter

The analog input X is sampled and fed to the input of the first flash converter (FC1). The first converter obtains the m most significant bits (MSBs) of the quotient which will be stored in the (MSBs) of the first register (R1) using a digital demultiplexer (DEMUX). This value is converted back into analog using a DAC with gain m_r and fed to a difference amplifier (DA). The output of the DA is the remaining quotient and the residue. This value is amplified by a gain 2^{im}, where i is the number of iteration, and the next m bits of the quotient are obtained again by the first flash converter. The process is repeated for k cycles where:

$$k = \left\lceil \frac{\left\lceil \log_2\left(\frac{2^n - 1}{m_r}\right) \right\rceil}{m} \right\rceil \tag{4.5}$$

After k cycles, the quotient which represents the base value will be stored in R1. The output of the DA represents the residues. This value is fed to the second flash converter (FC2) to obtain the digital representation of the residue [39,40].

The proposed architecture reduces the hardware complexity and thereafter the power consumption. In addition, it reduces the capacitive loading at the input, and improves the performance of the overall A/R converter. Digital correction circuit can be added for further improvement as proposed in [39]. However, these advantages are at the price of increasing the latency, where $k + 1$ cycles are required to perform the conversion. Moreover, the circuit is not very simple and requires an accurate variable gain amplifier.

In our approach [41], we propose an A/R conversion scheme based on the same flash principle. The complexity is significantly reduced while preserving most of the advantages of the flash converter. The proposed architecture is shown in Figure 4.3. Consider the analog input X is in the dynamic range $[0 : M - 1]$. We compare the analog input with the base values of modulus m_i which are produced by the resistor ladder. This requires $\frac{M}{m_i} - 1$ comparators in the first stage instead of M comparators to compare with all levels as in [37] and [38]. The outputs of the comparators are converted from thermometer code to binary code using an encoder. The digital output is the binary representation of the base value with respect to modulus m_i. The base value is converted into analog and subtracted from the analog input using a difference amplifier (DA). The difference amplifier has a gain of $\frac{M}{m_i}$ to maintain the same V_{ref} for both ladders in the two stages. The output of the difference amplifier represents the residue. The

output residue is converted into digital format using another flash ADC with $m_i - 1$ comparators. By knowing the base value and the residue of the input X with respect to one of the moduli, we can determine the other residues with respect to their moduli as shown in [37] and [38].

The total number of comparators in the proposed architecture is $\left(\frac{M}{m_i} - 1\right) + (m_i - 1)$. This number can be used to estimate the overall area size since it is proportional to the number of latches and the size of the encoders. The power consumption is also directly proportional to the number of comparators. To illustrate the great saving in the proposed converter, consider the moduli-set $\{15, 16, 17\}$ where $M \approx 2^{12}$ represents a 12-bit converter. Using the architecture in [37], we need $M - 1 = 4079$ comparators. Using our proposed architecture, we need only 255 comparators.
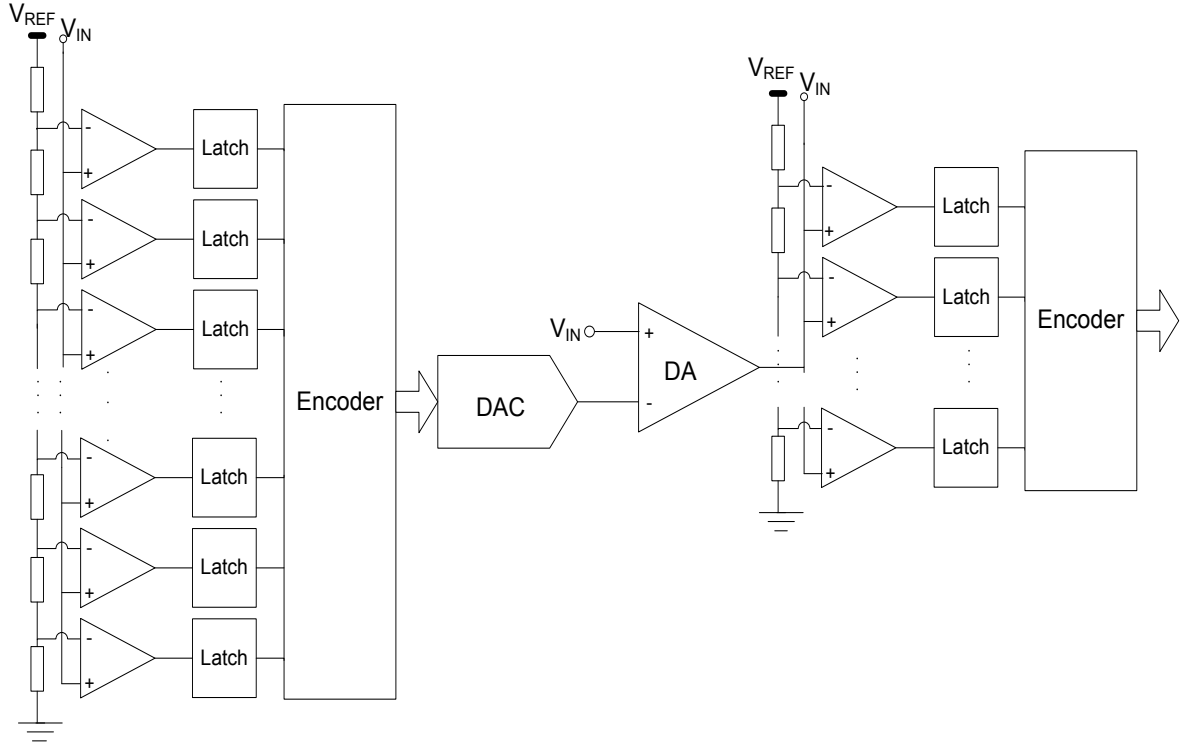

Figure 4.3. Modified flash A/R converter

For the widely used moduli-set $\{2^k - 1, 2^k, 2^k + 1\}$, the dynamic range is given by $2^{3k} - 2^k \approx 2^{3k}$ for $k \gg 3$, and the resolution is $n \approx 3k$ bits. Table 4.1. shows the total number of comparators required to implement the architecture proposed in [37] and the one proposed here

for different resolutions. Figure 4.4. shows the great advantage gained by reducing the complexity (in terms of number of comparators) versus k. The figure shows that the complexity of the scheme proposed in [37] grows exponentially with the resolution, while the proposed scheme allows using much higher resolution without highly increasing the complexity. Further reduction in the number of comparators can be achieved using interpolating and folding techniques.

Table 4.1. Number of comparators in [37] and in the proposed architecture

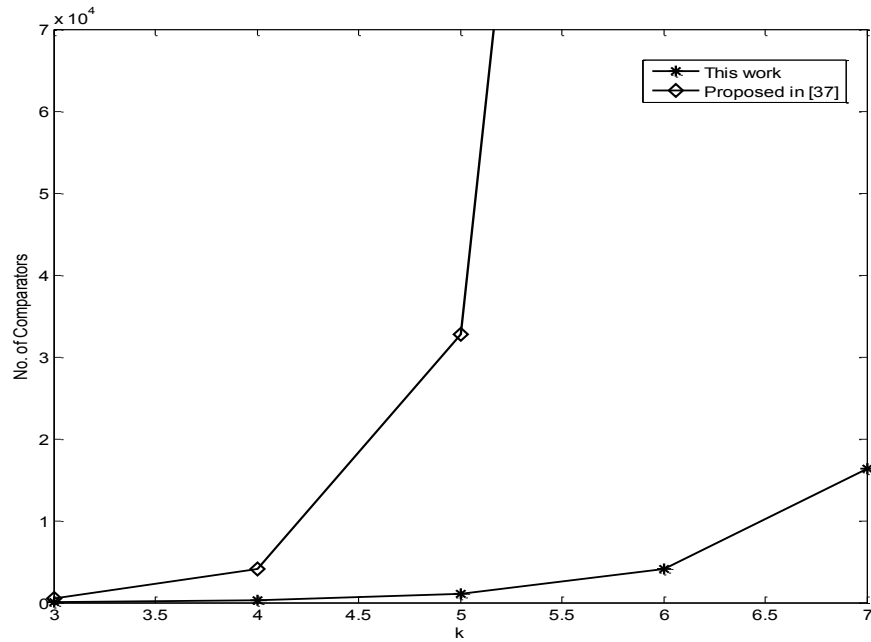| Resolution	Number of comparators	
	In [37]	This work
9	503	63
12	4,079	255
15	32,735	1023
18	262,079	4095
21	2,097,023	16,383

Figure 4.4. Complexity vs. k of the proposed scheme compared to [37]

Reducing the number of comparators in each stage relaxes the requirement of small comparator offsets, and improves the monotonicity of the conversion stages.

In terms of speed, the proposed architecture introduces an additional conversion stage and needs digital-to-analog conversion. At first glance, the proposed architecture seems to be much slower than the one proposed in [37]. However, reducing the number of comparators reduces the input capacitive loading which solves a practical problem in flash converters and reduces the delay. In addition, the two stage configuration allows pipelining the input data into two stages. The first stage is to obtain the base value, and the second is to obtain the residue. This pipelining increases the throughput by a factor of 2 approximately.

A Simulink model is built to simulate the behavior of the proposed A/R converter. The high-level block diagram of a 9-bit two-stage flash A/R converter is shown in Figure 4.5. The used moduli-set here is $\{7, 8, 9\}$. At first, the components are assumed to be ideal.
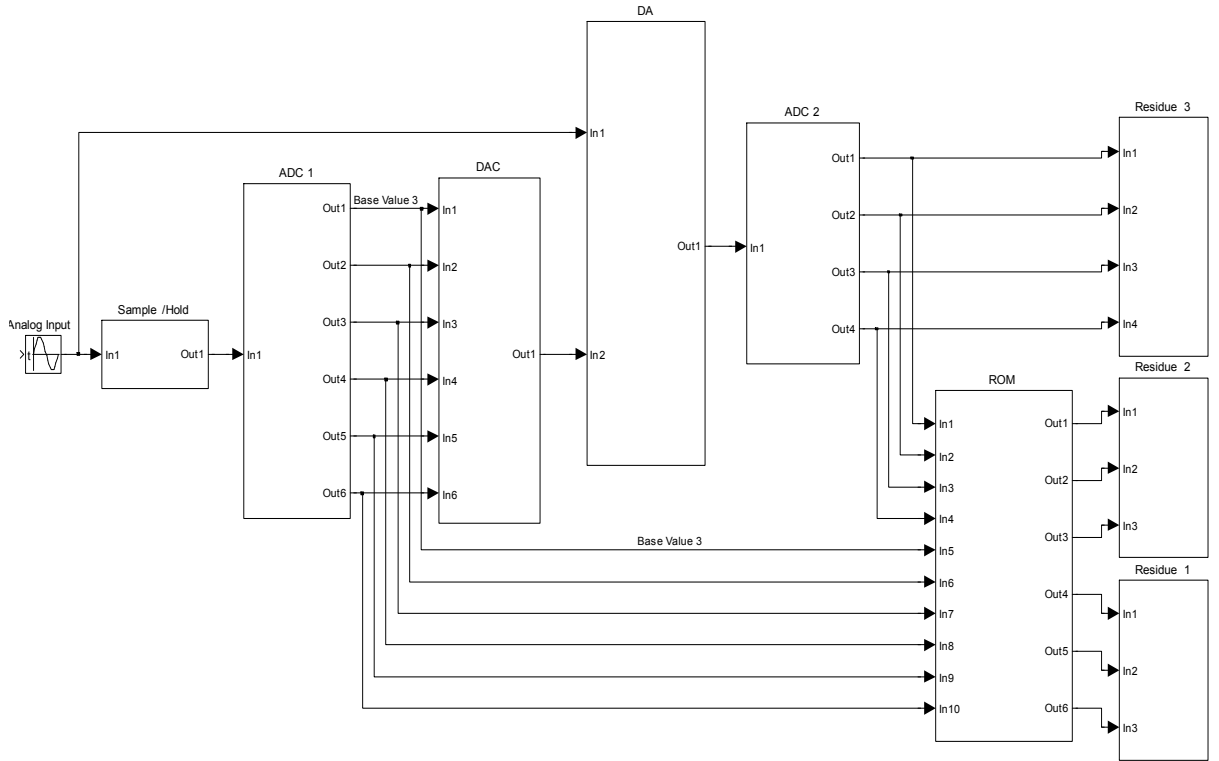
Figure 4.5. Simulink model of the two-stage flash A/R converter

A ramp is applied at the input, and the residue with respect to modulus $m_3 = 9$ is obtained at the output. The scaled input and output are shown in Figure 4.6. The other two residues can be obtained in parallel using two additional converters. However, this will increase both the area and the power consumption of the A/R converter by a factor of 3. Another way to obtain the remaining residues is to use a ROM look-up table. The ROM inputs are the base value and

the residue with respect to modulus m_3. Knowledge of these two values is sufficient to obtain the other two residues. This approach maintains the area and the power consumption at minimum, but extra delay through the ROM is inevitable.
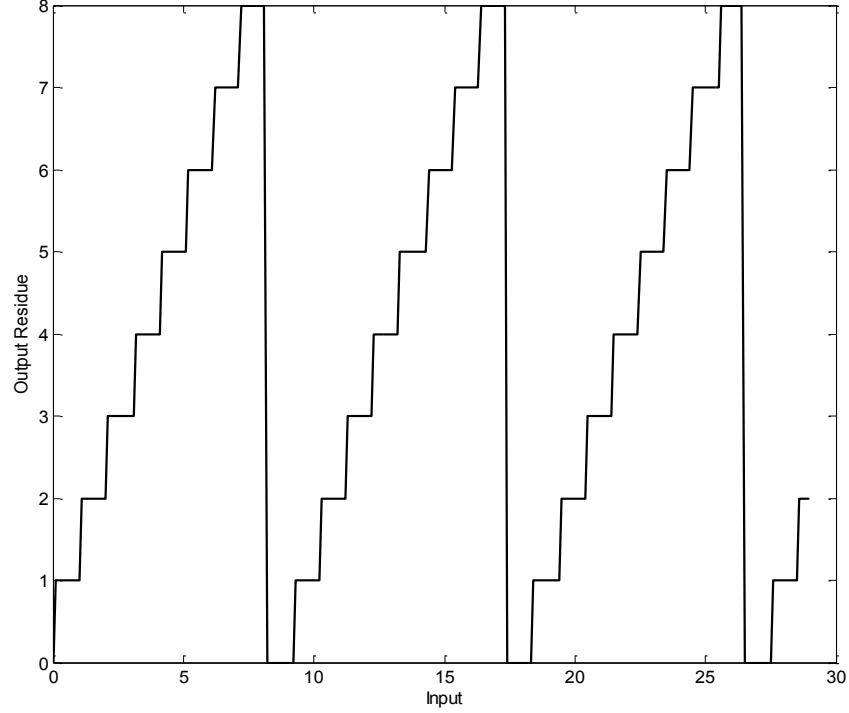
Figure 4.6. Output response to a ramp input

The maximum SNR is calculated by applying a single tone input within the specified range of frequency. In our case, the sampling frequency f_s is chosen at 1 GHz. Consequently, to satisfy Nyquist-Shannon theorem, the bandwidth has to be at maximum 500 MHz. The input frequency f_{in} is chosen at 519 KHz, and it satisfies the coherency requirement to avoid spectrum leakage:

$$f_{in} = \frac{M}{N} f_s \tag{4.6}$$

where M and N are relatively prime. In our case, $M = 17$ and $N = 2^{17}$. The quantized output has 504 quantization levels. This is equivalent to $\log_2 504 = 8.98$ bits.

The SNR is calculated from the output spectrum shown in Figure 4.7. The bin ($M + 1 = 18$) corresponds to f_{in}. The SNR is obtained over the range $0 - 500$ MHz. The obtained value from the simulation is 55.8 dB. This is consistent with the value obtained from Equation (3.8) where:

$$SNR = 6.02 \times 8.98 + 1.76 = 55.8 \; dB$$
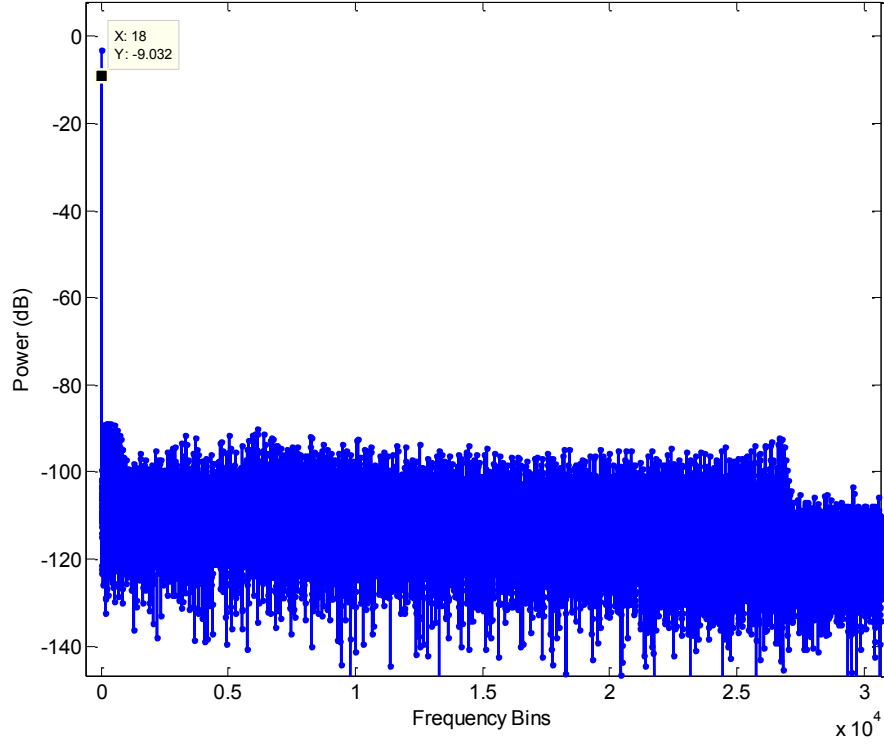
Figure 4.7. The quantized output spectrum

Non-idealities of the components degrade the performance of the converter and reduce the SNR. The S/H circuit suffers from two main non-ideality sources: thermal noise and clock jitter. The S/H circuit model is shown in Figure 4.8. The S/H circuit is mainly an RC circuit which has an RMS input referred noise voltage $v_{in,rms}$ given by [26]:

$$v_{in,rms} = \sqrt{\frac{kT}{C_s}} \qquad (4.7)$$

where k is Boltzmann constant, T is the absolute temperature, and C_s is the sampling capacitor.
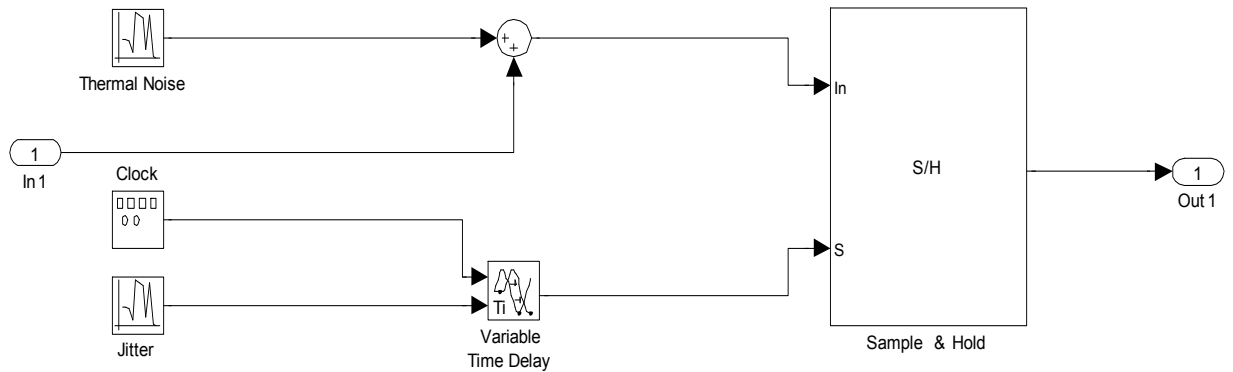
Figure 4.8. The S/H circuit model

The thermal noise is usually modeled as an additive white noise source with Gaussian distribution [43]. The RMS value is equivalent to the standard deviation of the noise distribution. Hence, the thermal noise can be modeled as a random variable generator added to the input signal with zero mean and standard deviation equals to the RMS value. It is worthy to notice that larger sampling capacitor C_s results in less thermal noise. However, this limits the speed of the converter as it reduces the maximum allowable sampling frequency f_s. Proper value of C_s has to be chosen to satisfy both requirements. The effect of the thermal noise of the S/H circuit on the SNR is shown in Figure 4.9.
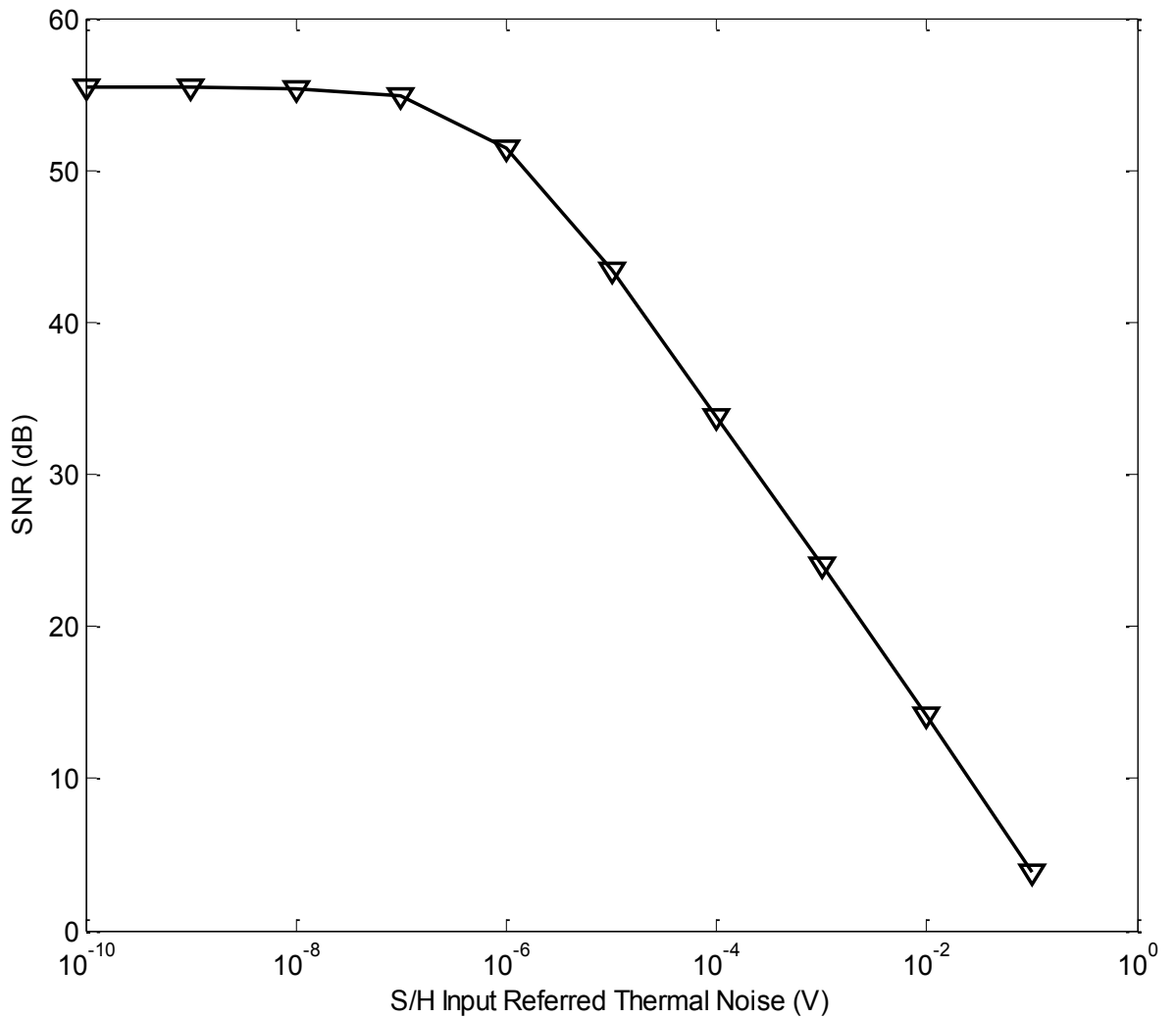
Figure 4.9. SNR vs. S/H input referred thermal noise

The second non-ideality source in the S/H circuit is the clock jitter. Clock jitter refers to the temporal variation of the clock period – that is, the clock period can reduce or expand on a cycle-by-cycle basis [44]. The jitter effect can be characterized as a zero mean random variable with standard deviation that causes variation in the clock sampling edge. The effect of the clock jitter on the SNR is shown in Figure 4.10.
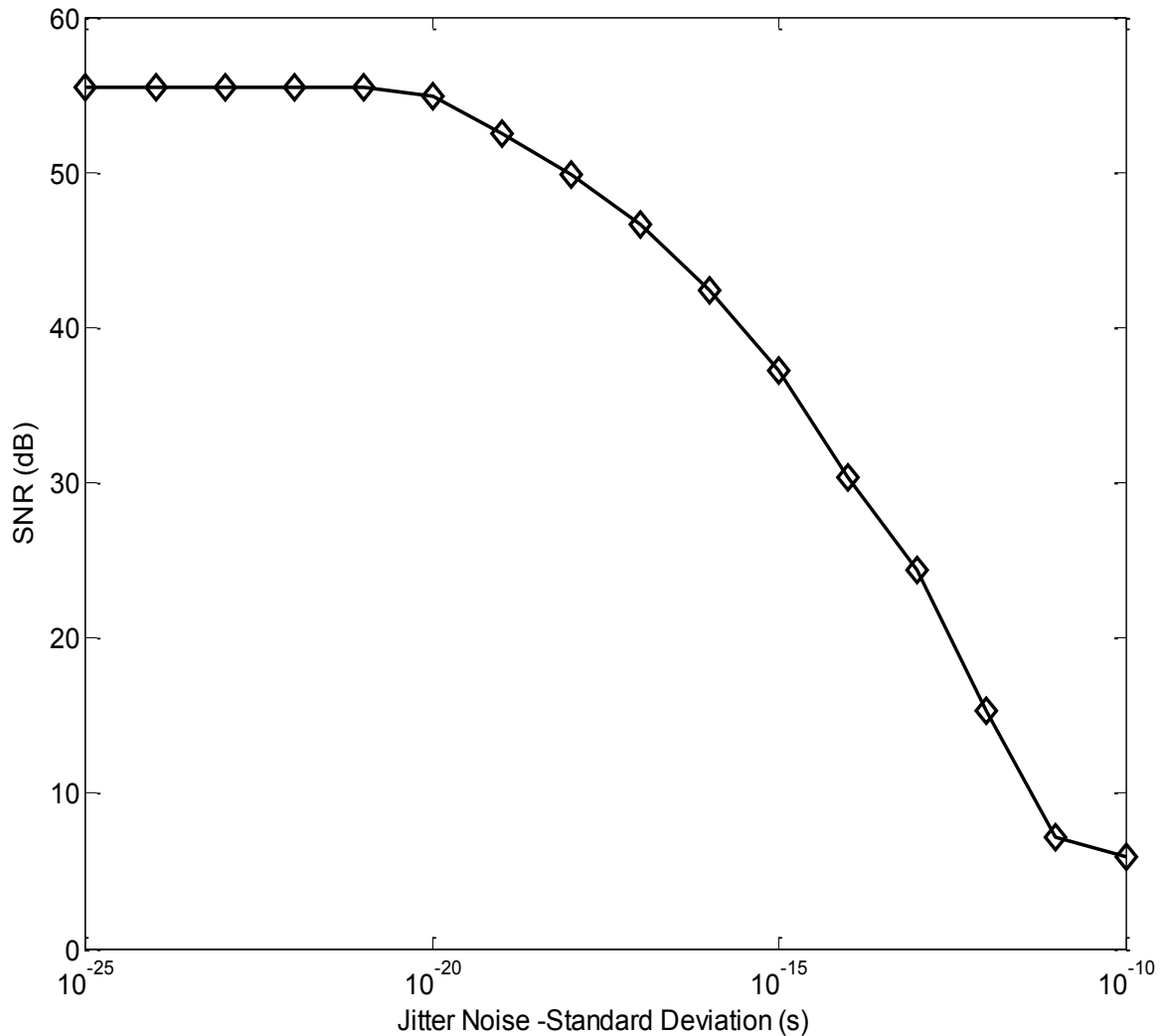
Figure 4.10. SNR vs. clock jitter

The simulated 9-bit two-stage flash A/R converter has two ADCs, one for each stage. The first ADC generates the base value, and consists of 55 comparators and a logic encoder. The second ADC generates the residue, and consists of 8 comparators and another logic encoder. The second ADC block diagram is shown in Figure 4.11.
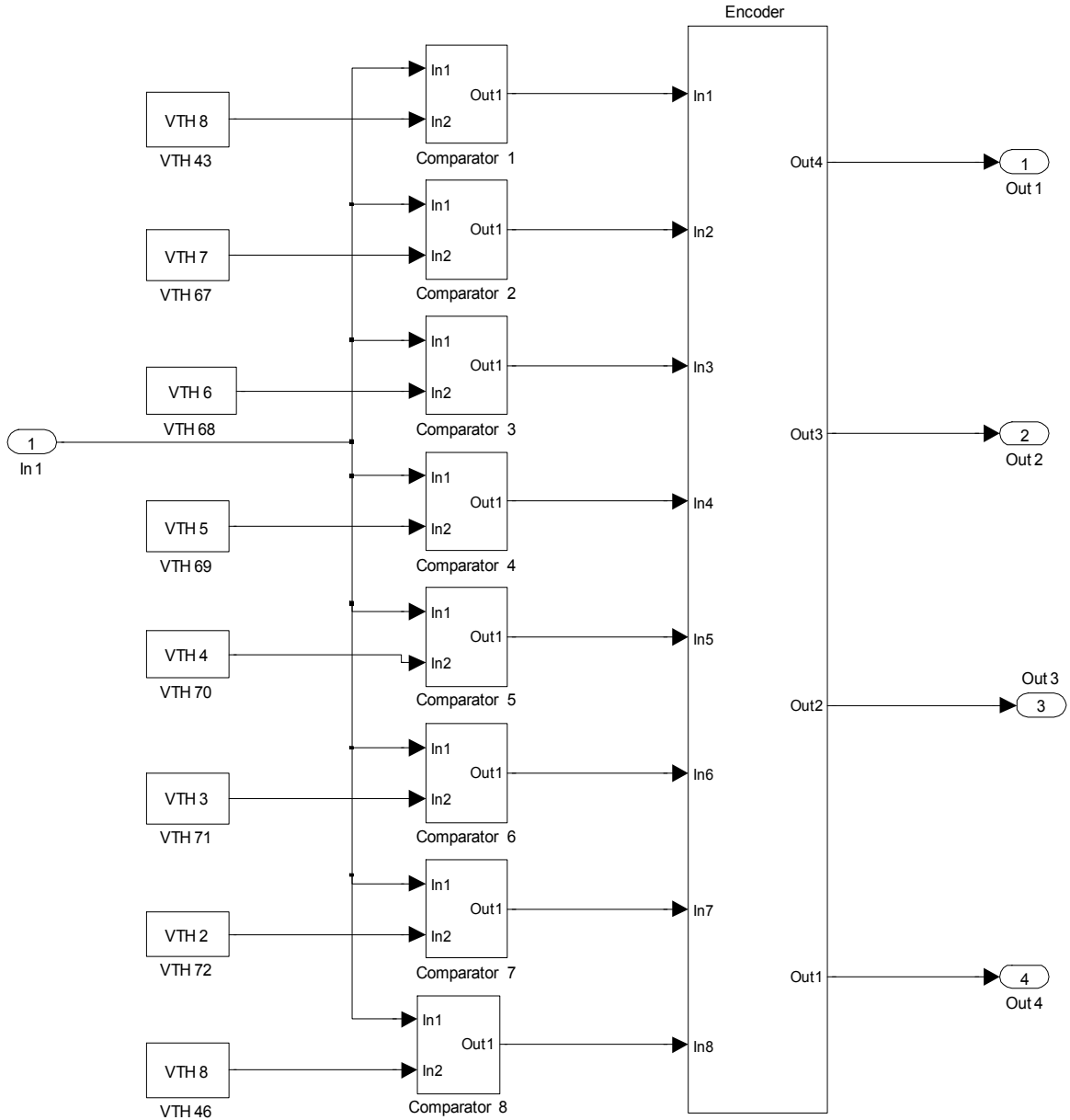
Figure 4.11. The second stage ADC block diagram

The logic encoder converts the thermometer code generated by the comparators into binary code. Usually, the thermometer code is converted into gray code, and then the gray code is converted into binary code. This approach alleviates the bubble errors since the adjacent gray codes will change only one bit in the code word [45]. For example, the second ADC requires a 4-bit thermometer code to binary code encoder. The conversion to gray code is shown in Table 4.2. The complete conversion to binary code can be easily implemented using pure Exclusive-OR operations as shown in Figure 4.12.

Table 4.2. Conversion from thermometer code to gray code

T_8	T_7	T_6	T_5	T_4	T_3	T_2	T_1	G_1	G_2	G_3	G_4
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	1	1
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	1	1	0
1	1	1	1	1	0	0	0	0	1	1	1
1	1	1	1	1	1	0	0	0	1	0	1
1	1	1	1	1	1	1	0	0	1	0	0
1	1	1	1	1	1	1	1	1	1	0	0
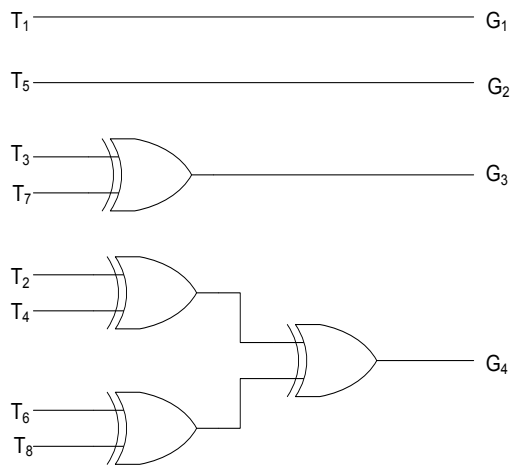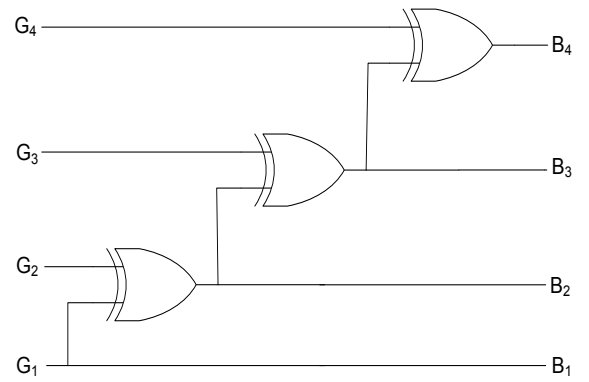
Figure 4.12. A 4-bit encoder: (a) thermometer to gray (b) gray to binary

The encoder shown in Figure 4.12 (a) can be represented by the following logic expressions:

$$G_1 = T_1 \tag{4.8a}$$

$$G_2 = T_5 \tag{4.8b}$$

$$G_3 = T_3 \oplus T_7 \tag{4.8c}$$

$$G_4 = T_2 \oplus T_4 \oplus T_6 \oplus T_8 \tag{4.8d}$$

Similarly, the encoder shown in Figure 4.12 (b) can be represented by the following logic expressions:

$$B_1 = G_1 \tag{4.9a}$$

$$B_2 = G_1 \oplus G_2 \tag{4.9b}$$

$$B_3 = G_1 \oplus G_2 \oplus G_2 \tag{4.9c}$$

$$B_4 = G_1 \oplus G_2 \oplus G_3 \oplus G_4 \tag{4.9d}$$

The design of the comparators is very critical and directly affects the performance of the A/R converter. The comparator model is shown in Figure 4.13. A high gain stage amplifies the difference, and the output is then latched to either 0 or 1 digital output. The comparators compare the sampled input with the threshold levels. Any variation in the resistor values in the ladder may result in erroneous quantized value at the output. Careful layout techniques can maintain the mismatch between the resistors within 1%. Therefore, the effect of this error source can be neglected in our first-order analysis. The other two main non-ideality sources in the comparator are the comparator offset and the thermal noise. These two sources result in variation in the comparator reference level which represents here the threshold level. Both sources can be modeled as additive random variable noise sources at the input. Since the two sources have the same effect and assuming they are uncorrelated, we can merge them into one random variable noise source at the input as shown in Figure 4.13.
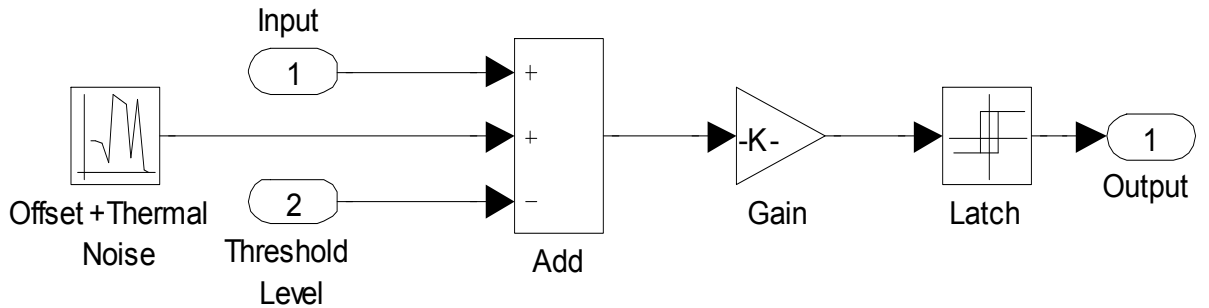
Figure 4.13. The comparator model

The effect of the comparator offset and thermal noise on the SNR is shown in Figure 4.14. To show how the proposed converter has more immunity against the comparator non-idealities, the SNR of proposed converter is compared to that of a 9-bit full-flash converter. The proposed converter relaxes the requirement of the comparators offset and thermal noise.
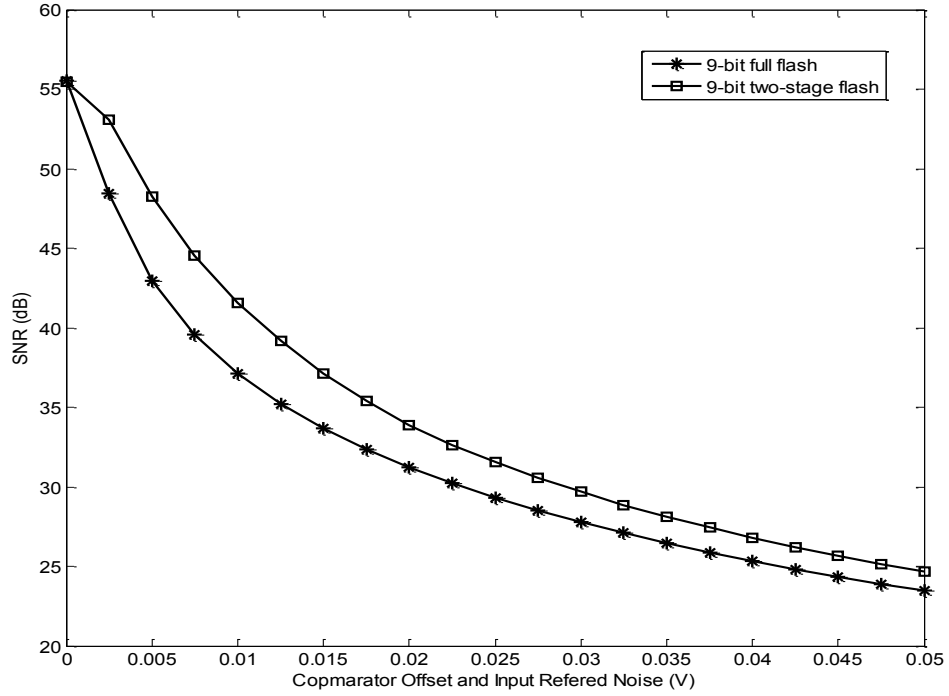
Figure 4.14. SNR vs. comparator offset and thermal noise

In the proposed converter, the DA ideally has a gain of $M/m_i = 56$ to amplify the difference and allow using the same reference voltages in the two-stage resistor ladders. The variation in this gain due to process variation can degrade the performance of the converter. The effect of $\mp 10\%$ variation in the DA gain on the SNR is shown in Figure 4.15.
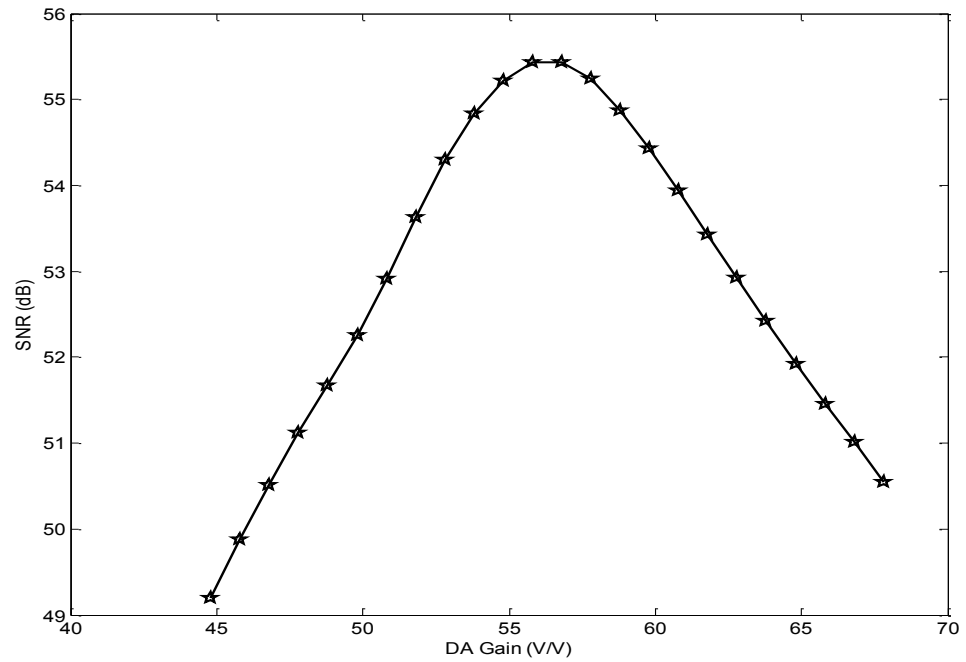
Figure 4.15. SNR vs. DA gain

4.1.2 Successive Approximation A/R Converter

An architecture based on the successive approximation principle has been introduced in [38] and [40] for direct conversion from analog to RNS representation. The proposed architecture is shown in Figure 4.16.
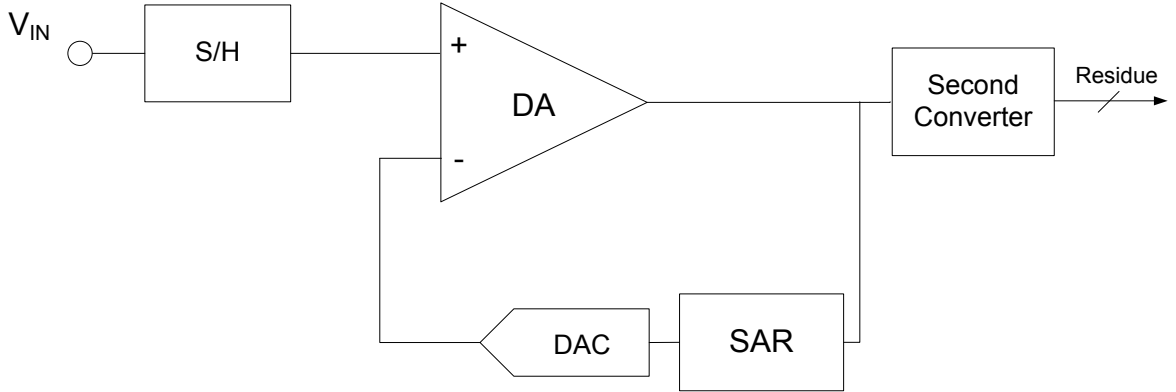
Figure 4.16. The successive Approximation A/R converter in [38] and [40]

The proposed architecture requires two stages, where the first stage generates the base value and the second stage generates the residue. The sampled input X is fed to the input of the first successive approximation converter. The comparator in a conventional successive approximation ADC is replaced by a difference amplifier (DA), and the DAC is modified by adding a weighting factor (m_r) to its output, where m_r is the mosulus. The base value is stored in the SAR of the first converter after $B + 1$ cycles, where:

$$B = \left\lceil \log_2 \left(\frac{2^n - 1}{m_r} \right) \right\rceil \tag{4.10}$$

where B is the number of bits of the SAR, and the resolution n is chosen such that $M \geq 2^n$.

After the base value is obtained, the output of the DA is the analog residue. This residue is converted into digital representation using a second converter which can be a flash converter or another successive approximation converter.

The proposed architecture is much simpler than the flash A/R converter architecture. However, it is not adequate for high speed applications. The latency of the proposed architecture is large compared to the flash converter. In addition, the conversion in the second stage requires the completion of the conversion in the first stage.

In our approach [41], we preserve the advantage of simplicity of the overall successive approximation ADC and only modify the weight of the SAR bits by modifying the gain of the DAC. Modifying the gain of the DAC is simply done by changing the reference voltage of the resistor ladder of the DAC. The proposed architecture is shown in Figure 4.17. The weight of the most significant ($\log_2 \frac{M}{m_n}$) bits is set to $2^i \times m_n$, and they represent the base value. The weight of the remaining ($\log_2 m_i$) bits is not changed and set to 2^i. The SAR has a total of B bits where $B = \left\lceil \log_2 \frac{M}{m_n} \right\rceil + \lceil \log_2 m_n \rceil$. The algorithm requires $B+1$ clock cycles to perform the conversion. Compared to the successive approximation ADC, the proposed converter requires an additional DAC and a summer operational amplifier. The overhead of the proposed converter is small and tolerable compared to the available successive approximation A/R converters proposed in [38] and [40].

To illustrate the operation of the proposed successive approximation A/R converter, consider a 12-bit converter with the moduli-set {15, 16, 17} where $M \approx 2^{12}$. To find the residue of the analog input $X = 95$ with respect to the modulus $m_3 = 17$, the final value stored in the SAR is 00000101-1010. The first eight most significant bits 00000101 represent the base value where $(2^0 \times 1 + 2^3 \times 1) \times 17 = 85$. The first four least significant bits 1010 represent the residue where $(2^1 \times 1 + 2^3 \times 1) \times 1 = 10$. The converter requires 14 clock cycles to carry out the conversion compared to 13 cycles for the successive approximation ADC. Therefore, the speed of the proposed converter is very close to that of a similar ADC and the need for a second converter is eliminated.
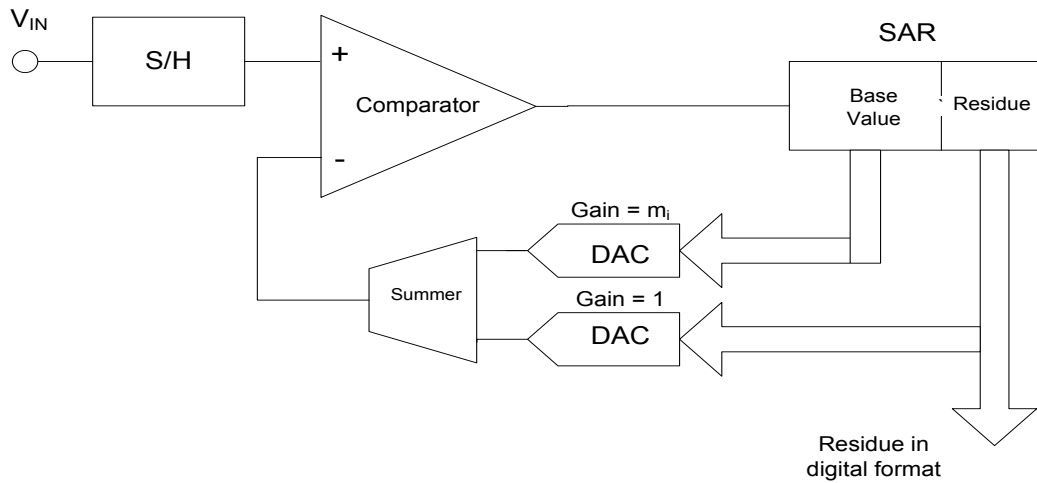
Figure 4.17. The proposed successive approximation A/R converter

Using the same moduli-set mentioned above, a Simulink model and a MATLAB code [46] are used to simulate the behavior of the 12-bit successive approximation A/R converter. The Simulink block diagram models the S/H circuit with the effect of clock jitter and thermal noise. It also includes the effect of the input referred offset and thermal noise of the comparator. The MATLAB code describes the successive approximation algorithm and takes into consideration the non-idealities of the DAC. The overall design is shown in Figure 4.18. The MATLAB code that describes the successive approximation algorithm is shown in Appendix I.
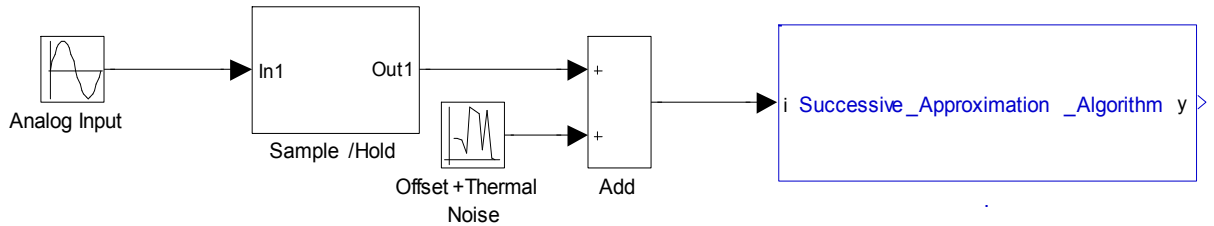
Figure 4.18. Simulink model of the proposed successive approximation A/R converter

A ramp is applied at the input, and the residue with respect to modulus $m_3 = 17$ is obtained at the output as shown in Figure 4.19. The other two residues can be obtained from the base value and the residue with respect to modulus m_3 using ROM look-up tables.
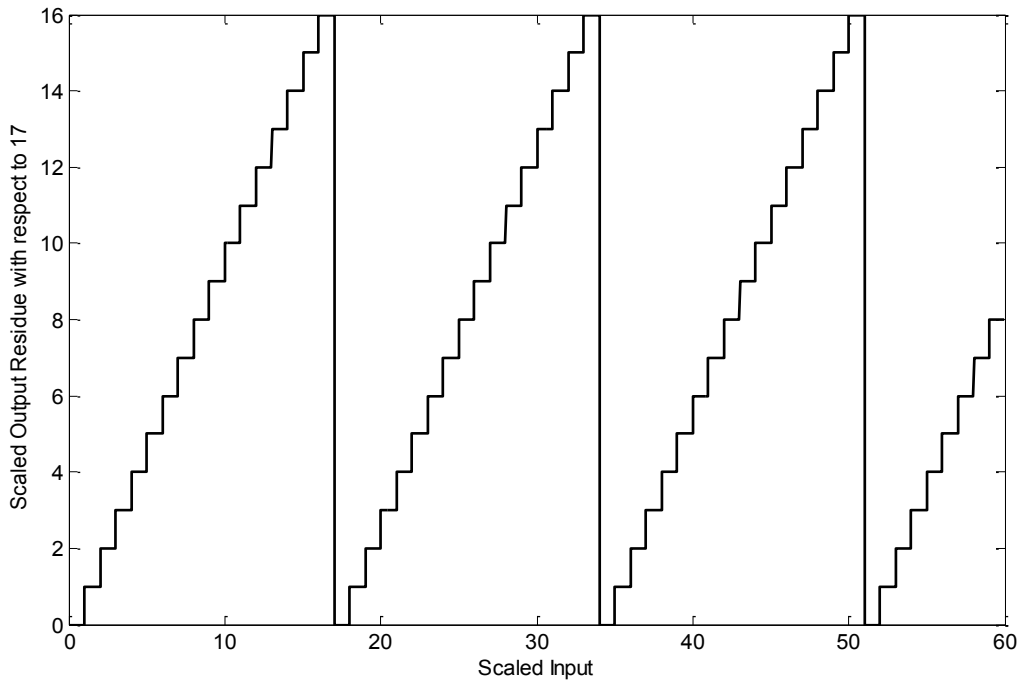
Figure 4.19. Output response to a ramp input

The simulated A/R converter has a dynamic range $M = 4080$. Hence, the equivalent number of bits is $\log_2 4080 \approx 12$ bits. The maximum SNR calculated from Equation (3.8) is 74 dB.

In a similar way to that used in modeling the two-stage flash A/R converter, the effect of the thermal noise and the clock jitter of the S/H circuit on the SNR is modeled. The resulting SNR with respect to thermal noise and clock jitter is shown in Figures 4.20. and 4.21., respectively.
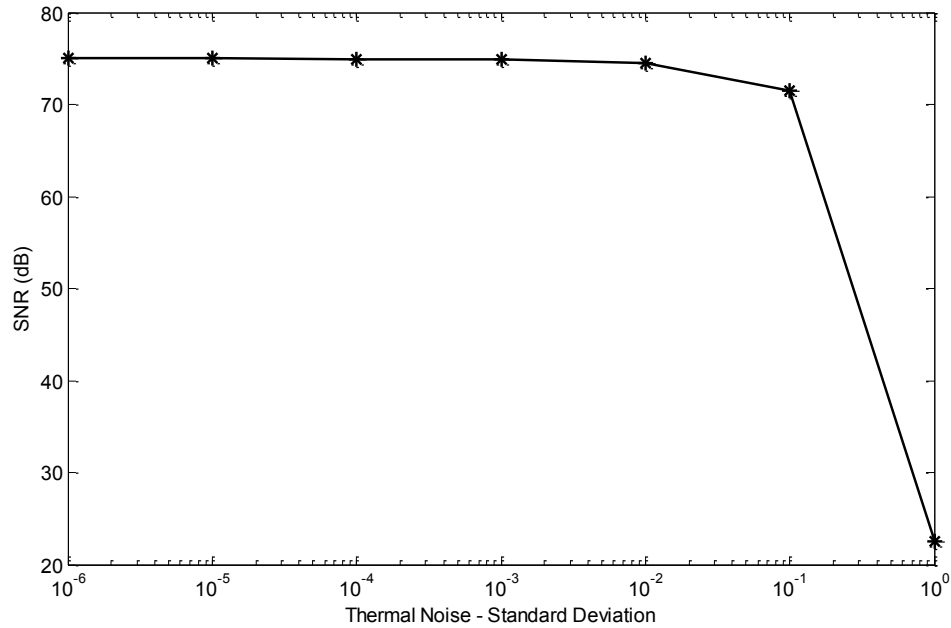
Figure 4.20. SNR vs. S/H thermal noise
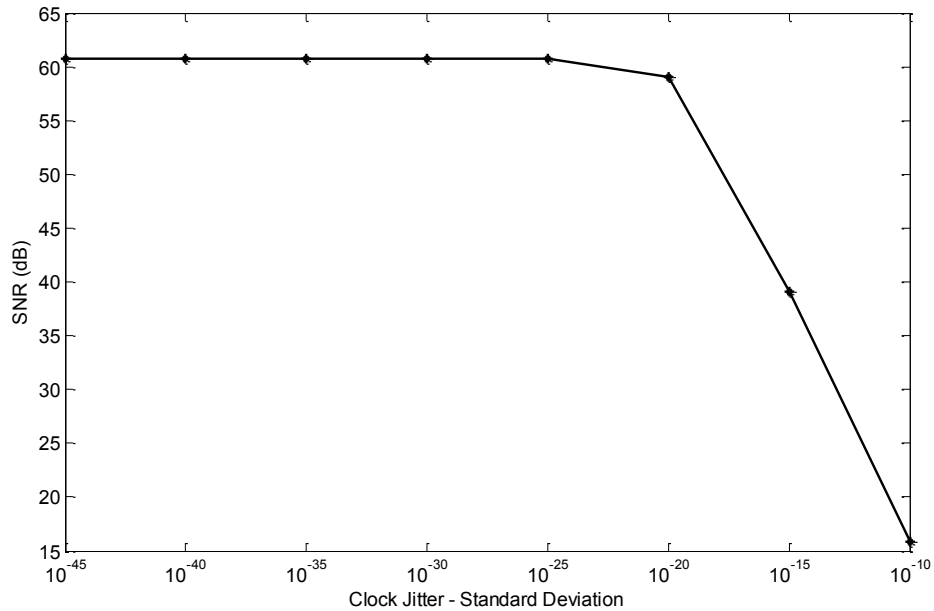
Figure 4.21. SNR vs. clock jitter

The input referred thermal noise and voltage offset of the comparator are modeled as one random variable noise source added to the sampled input. The effect of the comparator non-idealities on the SNR is shown in Figure 4.22.
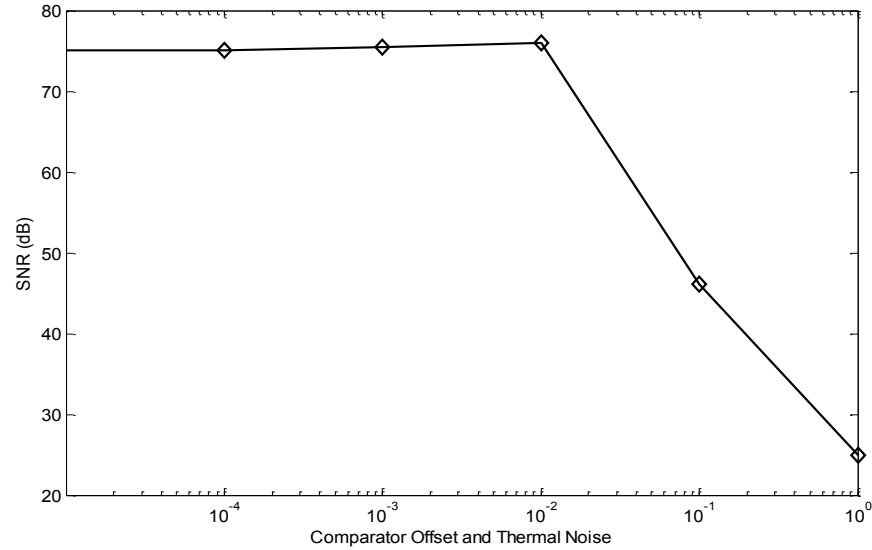
Figure 4.22. SNR vs. comparator offset and thermal noise

The dynamic characteristics of the DACs affect the performance of the A/R converter and degrade the maximum SNR. The effect of the finite bandwidth and slew rate of the DAC is included in the MATLAB code of the successive approximation algorithm. The effect of the finite bandwidth and slew rate on the maximum SNR is shown in Figures 4.23. and 4.24., respectively.
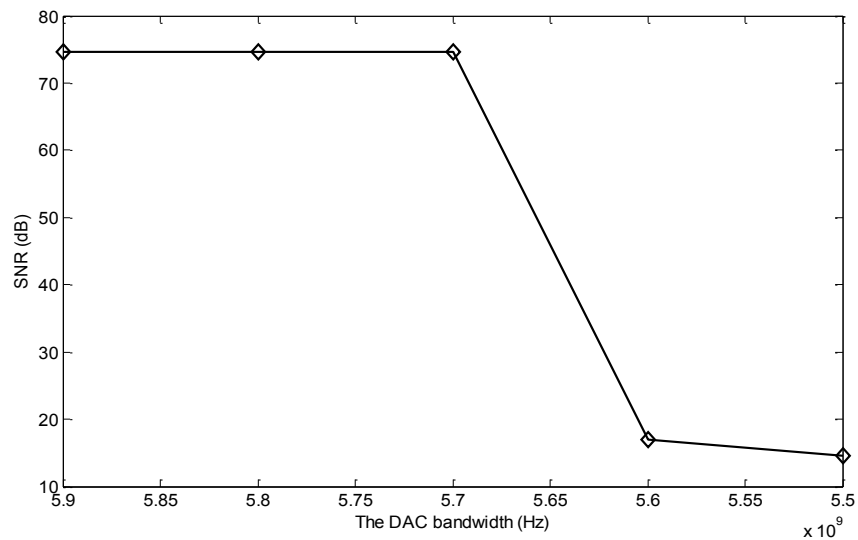
Figure 4.23. SNR vs. the DAC bandwidth
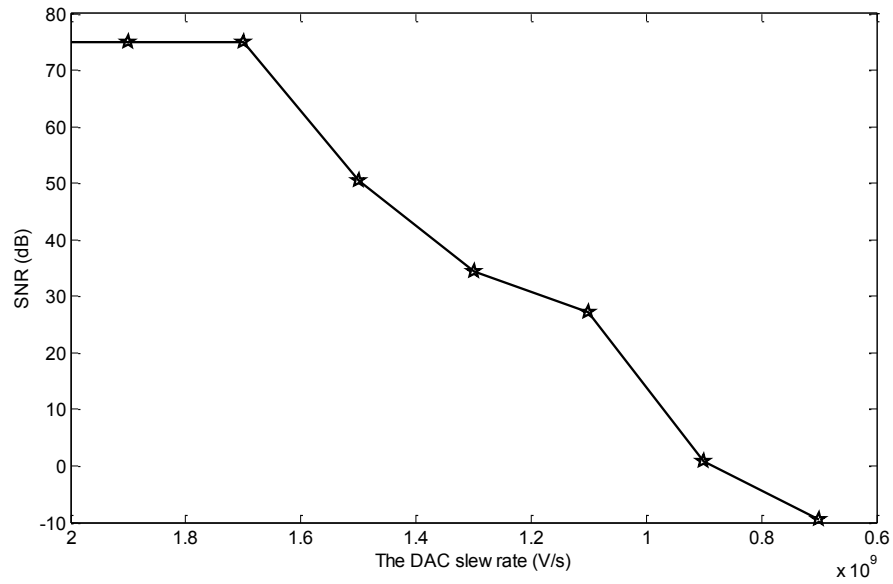
Figure 4.24. SNR vs. the DAC slew rate

4.1.3 Folding A/R Converter

A scheme based on folding principle is introduced in [42] for direct A/R conversion. The architecture is shown in Figure 4.25. for a three-moduli A/R converter.
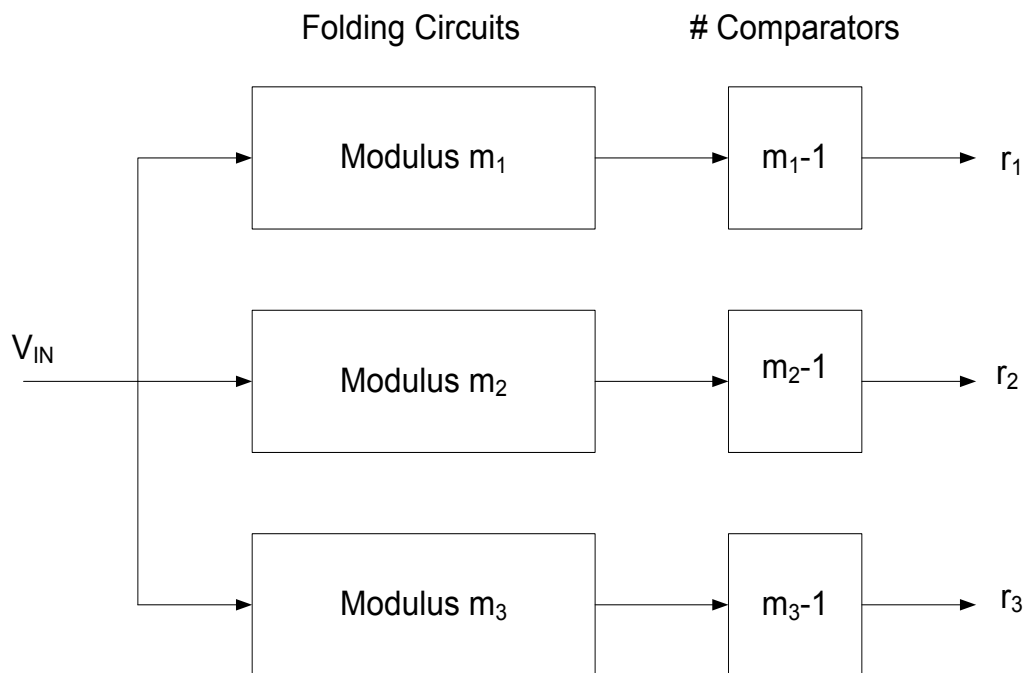
Figure 4.25. A three-moduli folding A/R converter architecture

This conversion technique uses analog folding circuits to fold the analog input. The RNS can be represented as a sawtooth folding waveform as shown in Figure 4.26., where the residue repeats itself at the multiples of m_i. However, realizing sawtooth waveforms is very difficult using analog circuits because of the discontinuity at m_i and its multiples. Instead, a differential pair can be used to realize one period of the waveform shown in Figure 4.27.

This waveform has one-to-one correspondence with the RNS representation. The residue r_i with respect to modulus m_i is related to the value d obtained from the folding stage as follows:

$$r_i = \begin{cases} d & : FB = 0 \\ 2m_i - d - 1 & : FB = 1 \end{cases} \tag{4.11}$$

where FB is the folding bit, which is 0 if the slope of the folding segment is positive, and 1 if the slope of the folding segment is negative. A single-stage folding circuit shown in [42] is capable of performing both operations: folding the input and obtaining the folding bit. Further processing to obtain the residue when $FB = 1$ can be done using combinational logic.
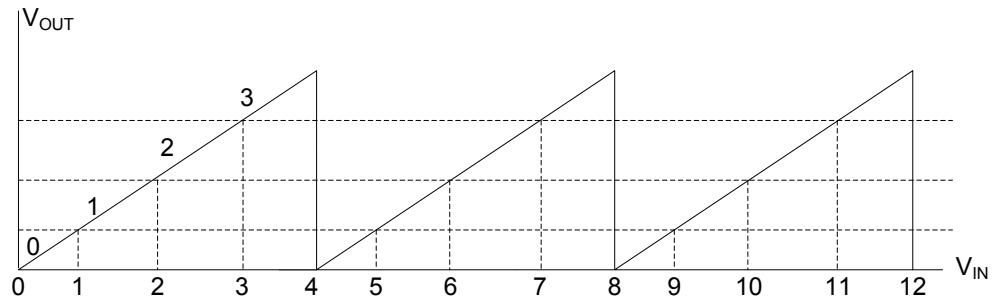
Figure 4.26. Folding wave form with respect to modulus 4
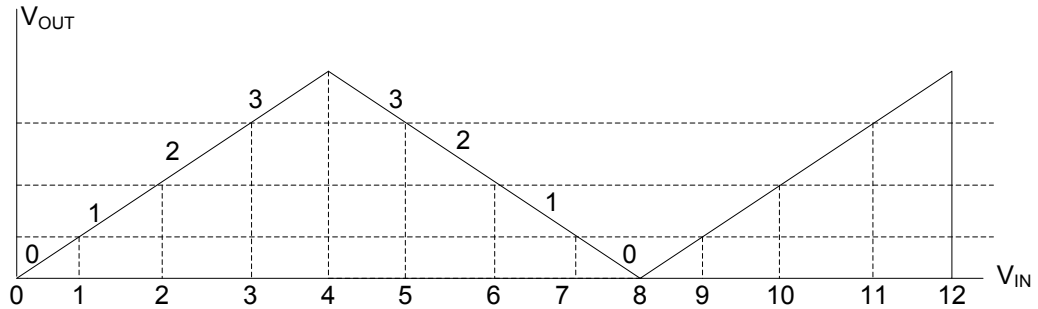
Figure 4.27. Output waveform of the folding circuit

The proposed architecture reduces the hardware complexity by reducing the number of comparators. For a three-moduli RNS, a folding A/R converter requires only $(m_1 + m_2 + m_3 - 3)$ comparators. However, the complexity is transformed to the folding circuits. The folding factor can be quite large for a large dynamic range. This requires a large number of folding stages, which results in large latency, power consumption and input capacitive loading.

4.2. Reverse Conversion from RNS to Analog Representation

In this section, we discuss the process of reverse conversion when the output is required to be in analog form. Some applications require direct interaction with the analog world. Usually, the residue-to-analog (R/A) conversion is performed in two steps where conversion to binary is an intermediate stage. This degrades the performance of the overall RNS by adding an extra overhead and increasing the latency. Therefore, a direct R/A converter is sought to solve that problem and make the RNS efficient. The problem of direct R/A conversion has not been sufficiently investigated yet. In this research area, the author in [47] tackled that problem and suggested a direct R/A converter based on MRC. The main drawback with the MRC based converter is the sequential nature of the algorithm, which makes it slow for large dynamic range applications. As a main contribution to this field, we propose a direct R/A converter architecture based on the CRT [48]. The proposed converter eliminates the need for an intermediate binary stage and can perform even better than the conventional R/B converter. The need for a large modulo adder is eliminated. Instead, a summer operational amplifier along with a folding circuit is used to perform modulo addition in the analog domain. The proposed converter facilitates the implementation of the CRT when direct conversion to analog form is required and it is very adequate for large dynamic range applications.

First, we present the scheme proposed in [47] for direct R/A conversion based on the MRC technique. Then, we present the CRT based R/A converter and its proposed architecture. A brief comparison between the performance of the two converters, and between the proposed R/A converter and the R/B converter is also presented.

4.2.1 MRC based R/A Converter

An architecture for direct conversion from RNS to analog representation was proposed in [47]. The architecture proposed is based on MRC. As it is obvious from Equation (2.31), the

MRC is a sequential algorithm. To generate z_i's, we require the knowledge of z_{i-1}'s. This is the main disadvantage of the MRC algorithm.

Consider an RNS with the moduli-set $\{m_1, m_2, m_3\}$. Based on the architecture proposed in [47], a combination of ROM implementation and an analog summer operational amplifier can be used. The architecture is shown in Figure 4.28.
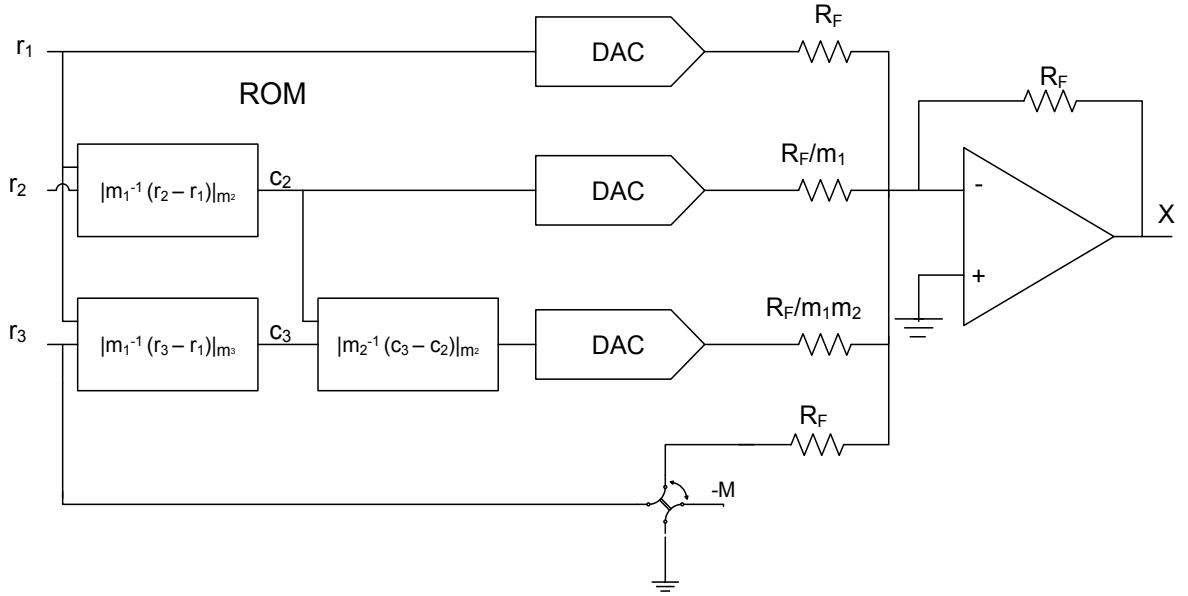
Figure 4.28. MRC based R/A converter

If each residue has k bits, then the z_i's can be generated using three ($2^{2k} \times k$)-bit ROMs. All z_i's can be converted to analog voltages using three k-bit DACs. The analog outputs are added using a summer operational amplifier. The resistors at the input and in the feedback are chosen and scaled to satisfy Equation (2.31). The result of Equation (2.31) is always positive since all z_i's and m_i's are positive. However, some applications require representing negative numbers. To achieve that, we can partition the full dynamic range $[0: M - 1]$ into two approximately equal halves: the upper half represents the positive numbers, and the lower half represents the negative numbers. The numbers X that can be represented using the new convention have to satisfy the following relations [4]:

$$-\frac{M-1}{2} \leq X \leq \frac{M-1}{2} \quad \text{if } M \text{ is odd} \tag{4.12}$$

$$-\frac{M}{2} \leq X \leq \frac{M}{2} - 1 \quad \text{if } M \text{ is even} \tag{4.13}$$

When the result of Equation (2.31) is in the negative half of the interval, the result is corrected by adding $-M$. The negative quantity of X is equivalent to the positive quantity $M - X$ in 2's compliment representation. To detect the sign, we can set m_n to be a power of 2, where m_n is the largest modulus. In this case, the most significant bit of z_n is a sign bit for X. If the sign bit is 1, the correction is performed by adding $-M$ to the result, and if the sign bit is zero, no modification is required.

The total time delay t_d of the architecture shown in Figure 4.28. is:

$$t_d = 2t_{ROM} + t_{DAC} + t_{Summer} \tag{4.14}$$

4.2.2 CRT based R/A Converter

In contrast to MRC, the CRT is not a sequential algorithm. The intermediate values can be generated in parallel using ROM look-up tables. A proposed architecture for direct conversion from RNS to analog representation is shown Figure 4.29.
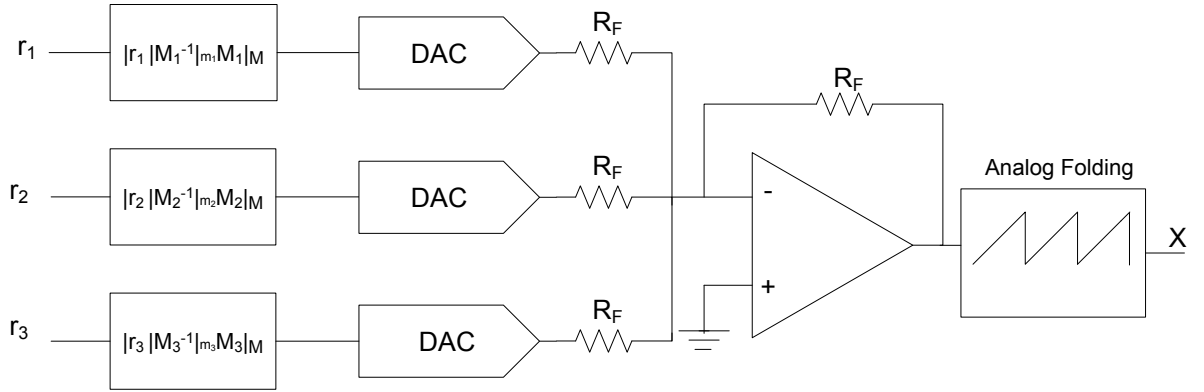
Figure 4.29. CRT based R/A converter

In order to realize an R/A converter based on the implementation of the CRT, we need to modulo add the intermediate values (partial sums of the CRT) generated by the ROMs. Assume each residue has k bits, then the partial sums are generated using three ($2^k \times 3k$)-bit ROMs. These values are converted into analog form using three k-bit DACs. Conventional addition is carried out by a summer operational amplifier. The summer operational amplifier should be capable of operating at input range of $3V_{REF}$, where V_{REF} is the reference voltage of each DAC.

To implement the modulo addition in the analog domain, we need a circuit that has the transfer function shown in Figure 4.30.
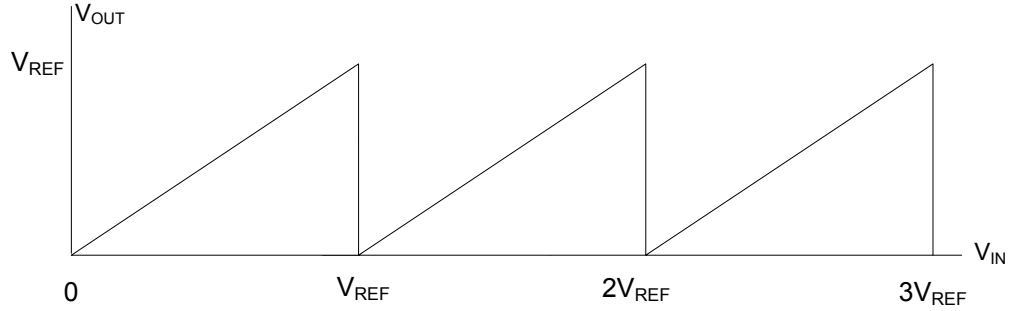
Figure 4.30. Folded sawtooth waveform

The main framework of the modulo addition operation is the sawtooth waveform. However, implementing this waveform using analog circuits is difficult to achieve because of the sharp transitions at V_{REF} and $2V_{REF}$. In practice, the circuit shown in Figure 4.31. is commonly used in folding ADCs to fold the analog input signal. The ideal relation between the input and the output is shown in Figure 4.32. Assuming a square-law MOS IV characteristic, the transfer function in each folding segment is described by:

$$V_{out} = A_v(V_{in} - V_R)\sqrt{1 - \frac{(V_{in} - V_{REF})^2}{4I_B/\beta}} \tag{4.15}$$

where $A_v = g_m R$ is the voltage gain, g_m is the MOS transconductance, I_B is the tail current, and β is a constant dependent on the MOS characteristic and dimensions.
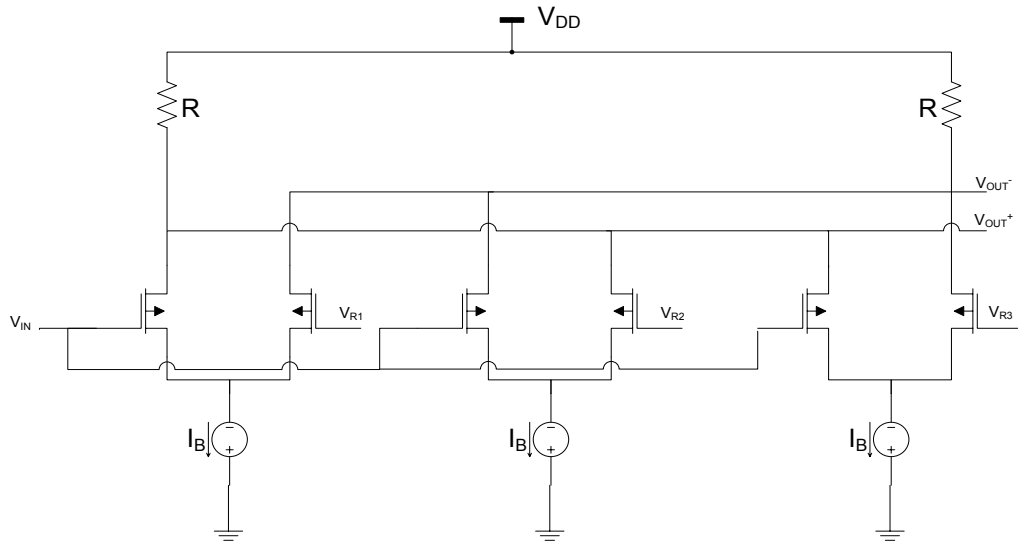
Figure 4.31. Folding circuit
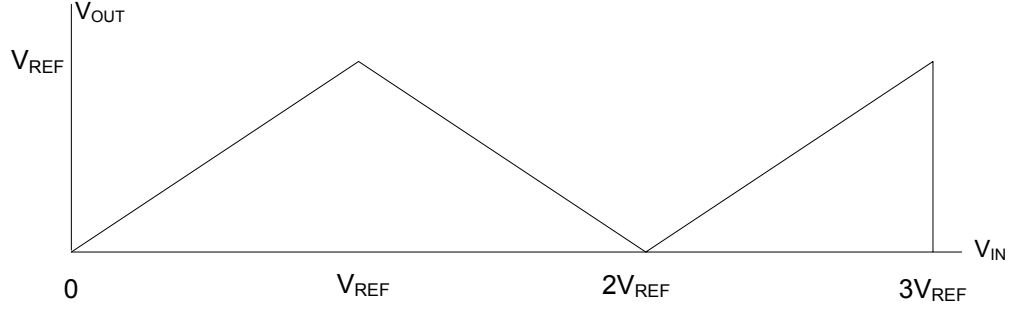
Figure 4.32. Folded triangle waveform

Assume the input signal is in the range $V_{REF} \leq V_{in} < 2V_{REF}$, and the output is taken differentially as $V_{out}^+ - V_{out}^-$. We notice that the slope of the folded signal in this region is negative. This will compensate for the negative sign resulting from the inverting summer amplifier, and the folded signal represents the modulo addition of the three inputs of the DACs. If the input of the folding circuit is in one of the other two regions ($0 \leq V_{in} < V_{REF}$ or $2V_{REF} \leq V_{in} < 3V_{REF}$), then the slope of the folded signal is positive. We need to multiply the folded signal by -1 to compensate for the inverting summer amplifier. However, since the output is taken differentially, multiplying by -1 is equivalent to interchanging the output nodes and take the differential output as $V_{out}^- - V_{out}^+$. An analog multiplexer will chose between one of the two possible configurations of the output ($V_{out}^+ - V_{out}^-$ or $V_{out}^- - V_{out}^+$) based on the information given about the folding region. This information can be obtained easily by a set of two comparators and an AND gate to detect the region $V_{REF} \leq V_{in} < 2V_{REF}$. The output of this folding region detector instructs the analog multiplexer to select between the two possible output configurations. The folding region detector is shown in Figure 4.33.

The total time delay t_d of the proposed architecture is given by:

$$t_d = t_{ROM} + t_{DAC} + t_{Summer} + t_{folding} \tag{4.16}$$

Compared to Equation (4.14), we notice that Equation (4.16) has an additional term $t_{folding}$. In addition, the DACs used in CRT based R/A converter are $3k$ bits instead of k bits compared to the MRC based R/A converter. However, the critical path of the ROM in the proposed architecture is reduced from two to one because the CRT is not sequential like the MRC. Also, the size of each ROM in the proposed converter is reduced from ($2^{2k} \times k$) bits to ($2^k \times 3k$) bits. For large k, the size of the ROM becomes a very important factor in the design for speed requirements.
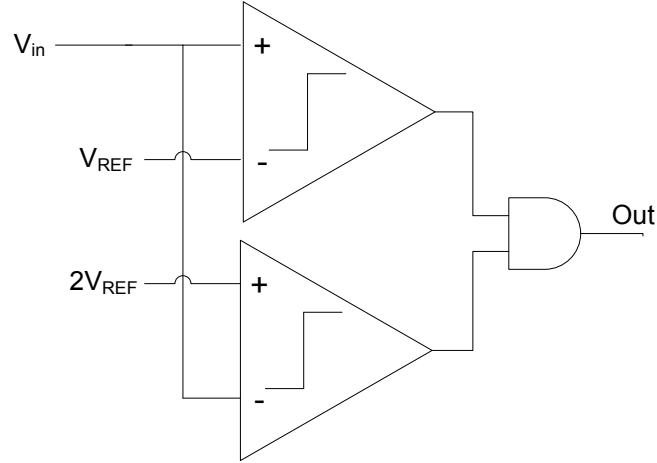
Figure 4.33. Folding region detector

In a conventional R/B reverse converter (Figure 2.8.), the total time required to obtain the binary values is given by:

$$t_d = t_{ROM} + t_{mod\_adder} \tag{4.17}$$

For large k, $t_{DAC} + t_{folding} \ll t_{mod\_adder}$. Therefore, the proposed direct R/A converter can be more efficient than a conventional R/B converter and it eliminates the need for a large modulo M digital adder.

A comparison that summarizes the hardware complexity and the latency in the proposed CRT R/A converter, the MRC R/A converter, and the CRT R/B converter is presented in Table 4.3.

Table 4.3. Hardware complexity and latency comparison among
different reverse conversion schemes

Converter	CRT R/A	MRC R/A	CRT R/B
ROM size	$2^k \times 3k$	$2^{2k} \times k$	$2^k \times 3k$
DAC resolution	$3k$-bits	k-bits	-
Modulo adder	-	-	Multi-operand modulo M adder
Latency	$t_{ROM} + t_{DAC} + t_{Summer} + t_{folding}$	$2t_{ROM} + t_{DAC} + t_{Summer}$	$t_{ROM} + t_{mod\_adder}$

Chapter 5

Conclusion and Future Work

In this chapter, we summarize the major points the key features of this work:

- The essentials of the RNS representation and its properties, advantages, and disadvantages were introduced. The enhanced speed and the low power consumption make the RNS very encouraging in applications with intensive multiply-and-accumulate operations. DSP applications are good candidates of such applications. The reduction in power consumption makes the RNS very encouraging for portable devices.

- Different schemes and architectures for forward data conversion from conventional representation to RNS representation were discussed. This process is considered as a preprocessing step to encode the data into residue form with respect to some given moduli. The overhead of this conversion stage has to be minimized to exploit the advantages of the RNS. The conventionally represented data can be in binary form or in analog form. In Chapter 2, we dealt with already quantized data in binary representation, while in Chapter 4 we discussed the conversion when the input is a real world analog signal.

- Different schemes and architectures for reverse data conversion from RNS representation to conventional representation were discussed. After the residue encoded data is processed by the RNS processor, they have to be converted back into conventional representation. This process is considered as a post-processing step. The reverse conversion is one of the most difficult RNS operations and has been a major, if not the major, limiting factor to a wider use of RNS [4]. The required output data

can be in binary form or in analog form. All reverse conversion algorithms are, in a way or another, based on Chinese Remainder Theorem (CRT) or Mixed-Radix Conversion (MRC). Various schemes for the implementation of these algorithms when the output is in binary representation were discussed in Chapter 2, while in Chapter 4 we discussed the implementation when the interaction with the analog world requires direct conversion from RNS to analog representation.

In this thesis, we focused on the algorithmic side of the design where most of the available and proposed data converters in RNS rely on similar data conversion techniques in digital systems. Many of these data converters (in digital) were developed and reported in the literature. Therefore, we focused on the high-level design and developed some models that describe the behavior of the proposed converters.

Among this work, the major contributions are:

- A direct analog-to-residue (A/R) converter based on the two-stage flash conversion principle was proposed. The proposed converter obviates the need for an intermediate binary stage. Explanation and analysis to demonstrate the efficiency of the proposed scheme were provided. The proposed A/R converter was compared to its ADC counterpart and to similar A/R converters in the literature. The proposed converter possesses the following advantages:
  - Reduced area and hardware complexity.
  - Reduced power consumption.
  - Reduced input capacitive loading.
  - Relaxed offset and thermal noise requirements of the comparators.
  - Possibility of using interpolating and folding techniques for further reduction in area and power consumption.
  - Possibility of pipelining to increase the throughput.

  A high-level model of the proposed two-stage flash A/R converter was presented. The model includes most of the non-idealities that affect the performance of the converter and degrade the maximum obtained SNR. The effect of each of these non-idealities on the SNR was analyzed.

- A direct A/R converter based on the successive approximation algorithm was proposed. The proposed converter preserves the simplicity of the successive approximation ADC and extends the principle to A/R conversion. The proposed converter performs better than the successive approximation A/R converters in the literature as it eliminates the need for a second stage converter to obtain the residue. In summary, the proposed successive approximation A/R converter has the following advantages:

    - The design is compact and provides reasonable speed, good resolution, and low power consumption.

    - The overhead of the proposed converter is small and tolerable compared to the available successive approximation A/R converters.

    - The speed of the proposed converter is very close to that of a similar ADC and the need for a second converter is eliminated.

  A high-level model of the proposed successive approximation A/R converter was presented. Using the proposed model, the effect of the components non-idealities on the maximum obtained SNR was analyzed.

- A direct residue-to-analog (R/A) converter based on the CRT was proposed. The need for an intermediate binary stage was eliminated. The proposed converter facilitates the implementation of the CRT when direct conversion to analog representation is required and it is very adequate for large dynamic range applications. The proposed R/A converter was compared to the MRC based R/A converter, and to the CRT based R/B converter. The key features of the proposed R/A converter are:

    - The need for a large modulo adder was eliminated. Instead, a summer operational amplifier along with a folding circuit was used to perform the modulo addition in the analog domain.

    - The proposed architecture reduces the size of the ROM which is a very important factor in the design of reverse converters for large dynamic range applications.

The work presented in this thesis can be extended in several ways. As RNS seems to be suitable for many modern algorithms, investigating more applications is one possibility of future work. We plan to explore integration to mixed-signal and multiple-valued FPGAs [49] and synthesis [50]. Further applications in signal processing, e.g., echo cancellation [51] are worth further investigation, as well as the circuits with imprecision [52]. Moreover, the isolation between the modulo channels facilitates error detection and correction. One new direction could be in conjunction with debug uses, as debug is a major issue in modern technologies. Applying RNS to some of these debug approaches [53, 54] is another encouraging possible future work.

In this thesis, we focused on developing efficient conversion schemes which we found very promising on the course of facilitating the implementation of RNS in different applications. Here, we mention the following as basic guidelines for any future work in this field:

- The variety of the available ADC topologies makes it very encouraging to investigate other possible A/R conversion architectures by manipulating the ADC architectures to accommodate the variations needed to generate residue-represented digits instead of binary digits. Among these architectures, folding and interpolating ADCs are promising. Pipelined architectures can also be studied and implemented. The two above-mentioned architectures are good candidates due to their capability of meeting speed requirements of the A/R conversion process.

- Developing efficient algorithms for reverse conversion is another possible path of future work. Variations on CRT or MRC can result in more efficient implantation of these algorithms. More effectively, developing a new efficient reverse conversion method can open new vistas of RNS implementation.

- Implementing and testing the proposed architectures on IC technology is recommended as future work. Acquiring experimental results will give credibility and reliability to the obtained results from theoretical simulations. Moreover, it will help encounter the technical problems that may arise in practical implementation of the proposed architectures.

References

[1] E. Grosswald, *"Topics from the Theory of Numbers,"* New York: McMillan, 1996.

[2] W. K. Jenkins, *"Finite Arithmetic Concepts in Handbook for DSP,"* S. K. Mitra and J. F. Kaiser, eds., Wiley, 1993, pp. 611-675.

[3] F. J. Taylor, *"Residue arithmetic: A tutorial with examples,"* Computer (IEEE), vol. 17, no. 5, pp. 50-63, May 1984.

[4] A. Omondi and B. Premkumar, *"Residue Number System: Theory and Implementation,"* Imperial College Press 2007, ISBN 978-1-86094-866-4.

[5] N. Szabo and R. Tanaka, *"Residue Arithmetic and its Applications to Computer Technology,"* New York: McGraw Hill, 1967.

[6] M. A. Soderstrand, 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.

[7] Wei Wang, M. N. S. Swamy, and M. O. Ahmad, *"An area-time-efficient residue-to-binary converter,"* Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems, Vol. 2, pp. 904-907, 2000.

[8] F. Barsi and P. Maestrini, *"Error correcting properties of redundant residue number systems,"* IEEE Transactions on Computers, vol. 23, no.9, pp. 915-923.

[9] R. Conway and J. Nelson, *"Improved RNS FIR Filter Architectures,"* IEEE Transactions On Circuits and Systems II, Vol. 51, No. 1, pp. 26-28, 2004.

[10]   W. Wei et al., *"RNS application for digital image processing,"* Proceedings of the 4th IEEE international workshop on system-on-chip for real time applications, Canada, pp. 77-80, 2004.

[11]   S. Yen, S. Kim, S. Lim and S. Moon, *"RSA Speedup with Chinese Remainder Theorem Immune against Hardware Fault Cryptanalysis,"* IEEE Transactions on Computers, vol. 52, no. 4, pp. 461-472, 2003.

[12]   J. Ramirez, et al., *"Fast RNS FPL-Based Communications Receiver Design and Implementation,"* Proceedings of the 12th Int'l Conf. Field Programmable Logic, pp. 472-481, 2002.

[13]   R. W. Watson and C. W. Hastings, *"Self-checked computation using residue arithmetic,"* Proceedings of the IEEE, 1966, vol. 54, pp. 1920-1931.

[14]   G. C. Cardarilli, A. Nanareelli, and M. Re, *"Reducing power dissipation in FIR filters using the residue number system,"* Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems, Vol. 2, pp. 320-323, 2000.

[15]   A. Nanareelli, M. Re, and G. C. Cardarilli, *"Tradeoffs between residue number system and traditional FIR filters,"* The 2001 IEEE International Symposium on Circuits and Systems, Vol. 2, pp. 305-308, 2001.

[16]   B. Cao, C. Chang, and T. Sirkanthan, *"A residue-to-binary converter for a new five-moduli set,"* IEEE Transactions on Circuits and Systems, 35 (11), 1998.

[17]   R. M. Capocelli and R. Giancarlo, *"Efficient VLSI networks for converting an integer from binary system to residue number system and vice versa,"* IEEE Transactions on Circuits and System, 35(11), pp. 1425-1431, 1998.

[18] B. Parhami and C. Y. Hung, *"Optimal table lookup schemes for VLSI implementation of input/output conversions and other residue number operations,"* In: *VLSI Signal Processing VI*, IEEE Press, New York, 1994.

[19]   G. Alia and E. Martinelli, *"VLSI binary-residue converters for pipelined processing,"* The Computer Journal, 33(5):473-475, 1990.

[20] A. Mohan, *"Efficient design of binary to RNS converters,"* Journal of Circuits and Systems, 9(3/4): 145-154, 1999.

[21]   G. Bi and E. V. Jones. 1988. *"Fast Conversion between binary and residue numbers,"* Electronic Letters, 24(9):1195-1997.

[22]   B. Vinnakota and V. B. B. Rao. 1994. *"Fast conversion techniques for binary-residue number systems,"* *IEEE* Transaction on Circuits and Systems, 14(12):927-929.

[23] B. Parhami and C. Y. Hung, *"Optimal Table Lookup Schemes for VLSI Implementation of Input/Output Conversions and other Residue Number Operations,"* In: VLSI Signal Processing VII, IEEE Press, New York, 1997.

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

[25] A. Oppenheim and R. Schafer, *"Discrete-Time Signal Processing,"* Pearson Higher Education 2010, ISBN 978-0-13-198842-2.

[26] D. Johns and K. Martin, *"Analog Integrated Circuit Design,"* John Wiley & Sons Inc. 1997, ISBN 0-471-14448-7.

[27] F. Maloberti, *"Data Converters,"* Springer 2007, ISBN 10 0-387-32485-2.

[28] M. Hotta, K. Maio, N. Yokozawa, T. Watanabe, and S. Ueda, *"A 150-mW 8-Bit video-frequency A/D converter,"* IEEE Journal of Solid State Circuits, vol. SC-21, pp. 318-323, Apr. 1986.

[29] Y. Wang, B. Razavi, *"An 8-bit 150MHz CMOS A/D converter,"* IEEE Journal of Solid-State Circuits, Vol. 35, pp. 308-317, Mar. 2000.

[30] R. Roovers, M. Steyaert, *"A 175 Ms/s, 6-b 160-mW 3.3-V CMOS A/D converter,"* IEEE Journal of Solid-State Circuits, Vol. 31, No. 7, pp. 938-944, July 1996.

[31] K. Kusomoto et al. *"A 10b 20MHz 30mW pipelined interpolating CMOS ADC,"* IEEE Int. Solid-State Circuits Conference, pp. 62-63, San Francisco, 1994.

[32] B. S. Song, S. H. Lee, and M. F. Tompsett, *"A 10-b 15-MHz CMOS recycling two-step A/D converter,"* IEEE Journal of Solid-State Circuits, vol. 25, pp. 1328-1338, Dec. 1990.

[33] M. P. V. Kolluri, *"A 12-bit 500-ns sub-ranging ADC,"* IEEE Journal of Solid-State Circuits, vol. 24, pp. 1498-1506, Dec. 1989.

[34] T. Gratzek, B. Brannon, J. Camp, and F. Murden, *"ADCs for digital receivers: the whole world tunes in,"* IEEE MTT-S Digest, pp. 1335-1338, 1996.

[35] A. Arbel and R. Kurz, *"Fast ADC,"* IEEE Transactions on Nuclear Science, vol. NS-22, pp. 446-451, Feb. 1975.

[36] A. Moscovici, *"High Speed A/D Converters,"* Kluwer Academic Publishers 2002, ISBN 0-306-46994-4.

[37] S. Mandyam and T. Stouraitis, *"Efficient analog-to-residue conversion schemes,"* Int. Symp. Circuits and Systems, pp. 2885-2888, 1990.

[38] T. Stouraitis, *"Analog- and binary-to-residue conversion schemes,"* IEE Proc. Circuits Devices Syst., pp. 135-139, 1994.

[39] A. P. Preethy and D. Radhakrishnan, *"A VLSI architecture for analog-to-residue conversion,"* Third Int. Conf Advanced A/D and D/A Conversion Techniques and Their Applications, pp. 83-85, 1999.

[40] A. P. Preethy and D. Radhakrishnan, *"A new approach to data conversion: A direct analog-to-residue converter,"* IEEE Int. Conf. Acoustics, Speed and Signal Processing, pp. 3013-3016, 1998.

[41] O. Abdelfattah, A. Swidan, and Z. Zilic, *"Efficient direct analog-to-residue conversion schemes,"* IEEE International Conference on Signals and Electronic Systems (ICSES), pp. 85-88, 2010.

[42] D. M. Pham, A.B. Premkumar, and A.S. Madhukumar, *"Reduced complexity analogue-to-residue conversion employing folding number system,"* Circuits, Devices & Systems, IET, pp. 30-41, 2010.

[43] F. Maloberti, P. Estrada, A. Valero, P. Malcovati, *"Behavioral modeling and simulation of data converters,"* Proc. of IMEKO 2000, vol. 10, pp. 229-236, Sept. 2000.

[44] J. M. Rabaey. *"Digital Integrated Circuits - A Design Perspective,"* Prentice Hall Electronics and VLSI Series, 1996.

[45] Y. Chuang, H. Ou, and B. Liu, *"A novel bubble tolerant thermometer-to-binary encoder for flash A/D converter,"* IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test, pp. 315-318, 2005.

[46] http://www.mathworks.se/matlabcentral/fileexchange/15417-successive-approximation-adc, by Fabrizio Conso.

[47] W. K. Jenkins, *"Techniques for residue-to-analog conversion for residue-encoded digital filters,"* IEEE Transactions On Circuits and Systems , Vol. 25, pp. 555-562, 1978.

[48] O. Abdelfattah, A. Swidan, and Z. Zilic, *"Direct residue-to-analog conversion scheme based on Chinese Remainder Theorem,"*, accepted in IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Dec. 2010.

[49] Z. Zilic and Z. G. Vranesic, *"Multiple valued logic in FPGAs",* Proceedings of 36th Midwest Symposium on Circuits and Systems, pp. 1553-1556, Detroit, Michigan, Aug. 1993.

[50] C. Côté and Z. Zilic, *"Automated systemC to VHDL translation in hardware/software codesign",* Proceedings of IEEE International Conference on Electronic Circuits and Systems, ICECS, pp. 717-720, Sept. 2002.

[51] J. Radecki, Z. Zilic and K. Radecka, *"Echo Cancellation in IP Networks ", Midwest Symp. Circuits and Systems*, pp. 219-222. 2002.

[52] Y. Pang, K. Radecka and Z. Zilic, *"Optimization of Imprecise Circuits Represented by Taylor Series and Real-Valued Polynomials", IEEE Trans. CAD,* Vol. 29, No. 8, Aug. 2010, pp. 1177 – 1190.

[53] M. Boulé, J-S. Chenard and Z. Zilic, *"Adding Debug Enhancements to Assertion Checkers for Hardware Emulation and Silicon Debug",* Proceedings of IEEE International Conference on Computer Design, ICCD 06, pp. 294-299, Oct. 2006.

[54] M. Boulé and Z. Zilic*, "Efficient Automata-Based Assertion-Checker Synthesis of SEREs for Hardware Emulation",* Proceedings of the 12th Asia and South Pacific Design Automation Conference, ASP-DAC2007, pp. 324-329, Jan. 2007.

Appendix I

MATLAB code to implement the successive approximation algorithm with finite bandwidth and slew rate of the DAC:

```matlab
function
[counter2,counter,r_thresholds]=SAR_AR(input,m,nbit,nbit_r,f_bw,sr,f_s)
% input= input sample
% m = modulus
% nbit=  number of base value bits
% nbit_r=  number of residue bits
% f_bw=  DAC bandwidth [f_s]
% sr=    DAC slew-rate [V_fs/T_s]
% f_s=   sampling frequency


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                 global variables                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                 %
 threshold(1,nbit+1)=0;  % threshold array
 counter=0;              % converter decimal output
 counter2=0;
 threshold(1)=0.5;       % 0 threshold
 threshold(2)=0.5;       % first threshold
 threshold_id=0.5;       % next ideal threshold



 r_threshold(1)=0.5;      % 0 threshold
 r_threshold(2)=0.5;      % first threshold
 r_threshold_id=0.5;      % next ideal threshold



 tau=1/(2*pi*f_bw);      % DAC output pole
 in=input/(m*2^nbit);

 Tmax=1/(f_s*nbit);       % clock period

 nbit;                   % Base value bits
 nbit_r;                  % Residue bits



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 for i=2:(nbit+1)        % conversion cycle for the base value

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                  finite  bandwidth & slew-rate                        %
%                      error calculation                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```matlab
%                                                            %
    deltaV=abs(threshold_id-threshold(i-1));
    slope=deltaV/tau;
    if slope > sr
        tslew=(deltaV/sr) - tau;
            if tslew >= Tmax                % only slewing
                error = deltaV - sr*Tmax;
            else
                texp = Tmax - tslew;
                error = (deltaV-sr*tslew)*exp(-texp/tau);
            end

    else                            % only exponential settling
        texp = Tmax;
        error = deltaV*exp(-texp/tau);
    end

    threshold(i) = threshold_id - sign(threshold_id-threshold(i-1))*error;
%                                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       successive approximation        %
%          conversion algorythm         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                       %
    if (in-threshold(i)) > 0
        threshold_id=threshold_id+1/2^i;
        bit=1;
    else
        threshold_id=threshold_id-1/2^i;
        bit=0;
    end
 counter=(counter+bit*2^(nbit-i+1));
 thresholds(i-1)=threshold(i);
 end
%                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


in2=(input-counter*m)/(2^nbit_r);


%                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 for i=2:(nbit_r+1)        % conversion cycle for the residue

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                   finite  bandwidth & slew-rate                      %
%                      error calculation                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                      %
    deltaV=abs(r_threshold_id-r_threshold(i-1));
    slope=deltaV/tau;
```

```matlab
    if slope > sr
        tslew=(deltaV/sr) - tau;
            if tslew >= Tmax                  % only slewing
                error = deltaV - sr*Tmax;
            else
                texp = Tmax - tslew;
                error = (deltaV-sr*tslew)*exp(-texp/tau);
            end

    else                               % only exponential settling
        texp = Tmax;
        error = deltaV*exp(-texp/tau);
    end

    r_threshold(i) = r_threshold_id - sign(r_threshold_id-r_threshold(i-
1))*error;
%                                                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      successive approximation         %
%         conversion algorythm          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                       %
    if (in2-r_threshold(i)) > 0
        r_threshold_id=r_threshold_id+1/2^i;
        bit=1;
    else
        r_threshold_id=r_threshold_id-1/2^i;
        bit=0;
    end
 counter2=(counter2+bit*2^(nbit_r-i+1));
 r_thresholds(i-1)=r_threshold(i);
 end
%                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              Output             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                 %
 counter2=counter2;
 if nargout > 1
    r_thresholds=r_thresholds;
 end
```