

HARDWARE IMPLEMENTATION OF
DIGITAL FIR FILTER IN
RESIDUE NUMBER
SYSTEM

By

KADAMBARI KALURI

Bachelor of Engineering

University of Madras

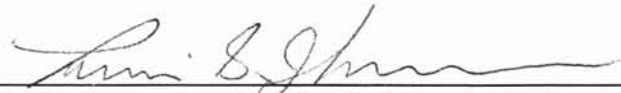
Chennai, India

2000

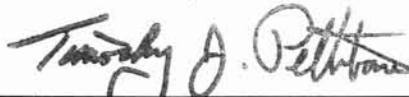
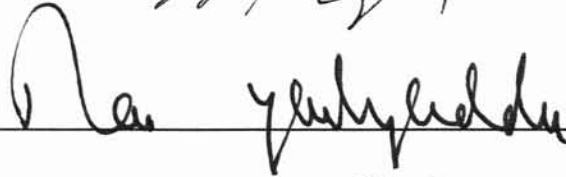
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2002

HARDWARE IMPLEMENTATION OF
DIGITAL FIR FILTER IN
RESIDUE NUMBER
SYSTEM

Thesis Approved:



Thesis Advisor



Dean of the Graduate College

Dedication

I would like to dedicate my thesis to my family for all their support and faith in me to achieve this today.

Acknowledgements

I would like to express my sincere thanks to my advisor Dr. Louis Johnson for his patience, inspiration, and guidance in helping me with my thesis and throughout my Master degree. I would also like to thank the Committee members, Dr. Chung and Dr. Yarlagadda for their guidance and support in completion of my thesis.

I would like to extend my thanks to Dr. Michael Soderstrand for his support and critical suggestions, effective guidance, throughout my thesis and career in Oklahoma State University.

My sincere thanks to the Digital Signal Processing and Communication Laboratories at the Oklahoma State University for supporting me with the most advanced resources which has helped me in using various mathematical and simulation software to achieve my goal efficiently.

I would like to thank Dr. Barrick for proofreading my thesis and making the necessary corrections.

I would also like to thank Kannan, for all his help during my Masters degree and the thesis, helping me throughout my career and being patient and bearing with me during the entire time.

Finally, I would like to thank my father Rangarao, mother Lavanya and sister Chinni, for their moral support and enthusiasm that I regard as important.

Preface

Over the past few years Digital Signal Processing has gained in popularity in the field of communication. The rise in the popularity of digital signal processing is due to the many advantages it possesses, such as high noise immunity, low cost, high speed, and flexibility. In recent times, many innovative methods are being developed to make communication devices as small and portable as possible. Hence efficient hardware implementation plays a key role in making communication devices more portable. FIR filters are the basic components used to manufacture communication devices

The hardware implementation of the RNS filter can be done using Look-Up-Table (LUT) in the block RAM's in FPGA's of the Xilinx Virtex FPGA's. The RAMs generated by the core generator are used for this purpose. The implementation was done on an example filter, to compare it with other implementations.

The result of this thesis is a highly efficient hardware realization of the desired FIR filter. The implementations of the Digital FIR filter in Residue Number System saves CLBs. CLBs are the basic building block of the FPGA. Residue Number System implementation utilizes the block RAMs present around the FPGA chip, which are generally not used by the communication circuitry. The hardware savings is obtained by using Block RAMs to implement the filter instead of CLBs. Conventional methods of implementations are faster than Residue Number System implementation, but that is the tradeoff to the savings in hardware.

Table of Contents

Chapter	Page
1. Introduction.....	1
1.1 Introduction to DSP.....	1
1.2 Motivation.....	2
1.3 Problem Definition.....	2
1.4 Thesis Organization.....	3
2. Background.....	5
2.1 Classification of Digital Filters.....	5
2.1.1 Finite Impulse Response Filters (FIR Filters).....	5
2.1.2 Infinite Impulse Response Filters (FIR Filters).....	7
2.2 Different Number System and previous work done.....	8
2.2.1 Two's Complement Number System.....	8
2.2.2 Canonical Signed Digit.....	9
2.2.3 Dempster Macleod.....	10
3. Residue Number System.....	11
3.1 Introduction.....	11
3.2 The Modulo Operation.....	11
3.3 Residue Number System.....	12
3.4 Multiplicative Inverse on Modulo Operation.....	15
3.5 Chinese Remainder Theorem.....	15
3.6 An Example using Residue Number System.....	16
4. Xilinx Virtex FPGAs.....	19
4.1 Introduction to Field Programmable Gate Arrays.....	19
4.2 Xilinx Virtex FPGAs.....	21
4.3 Configurable Logic Block.....	22
4.4 Input/Output Block.....	23
4.5 Block Select RAM.....	24
5. FIR Filter in Residue Number System.....	27
5.1 Introduction.....	27
5.2 Basic Concept.....	27
5.3 Modulo Addition.....	29

5.4	The Example Filter.....	29
5.5	Small Moduli RNS FIR Example.....	30
5.6	Large Moduli RNS FIR Example.....	33
6.	Hardware Implementation of the Filter.....	40
6.1	Introduction.....	40
6.2	Implementation Procedure.....	40
6.2.1	MIF File Generation.....	42
6.2.2	Core Generator and Xilinx Schematic Editor.....	42
6.2.3	Hardware Implementation.....	48
6.3	Simulation and Implementation Results of Small Moduli RNS Filter.....	49
6.4	Simulation and Implementation Results of Large Moduli RNS Filter.....	51
7.	Conclusion and Future Work.....	53
7.1	Xilinx Implementation of Example Filter.....	53
7.2	Comparison of Xilinx and RNS Implementation.....	54
7.3	Conclusion.....	55
7.4	Future Work.....	56
	Bibliography.....	57
	Appendix A.....	61

List Of Figures

Figure	Page
4.1	Types of Field Programmable Gate Array (FPGA).....20
4.2	Virtex Architecture Overview.....22
4.3	2-Slice Virtex Configurable Logic Block (CLB).....23
4.4	Virtex Input/Output Block (IOB).....24
4.5	Dual-Port Block SelectRAM.....25
4.6	Single-Port Block SelectRAM.....25
5.1	RNS Digital FIR Filter.....28
5.2	Example Digital FIR Filter Response.....30
5.3	Block Diagram of 4-moduli RNS Digital FIR Filter.....32
5.4	Binary to RNS (4 block Rams 256x16 and 3, 2's comp adders).....35
5.5	Multiplication by b coefficients in Moduli 256 (2 block RAMs 256x16).....36
5.6	Multiplication by b coefficients in Moduli 255 (2 block RAMs).....36
5.7	Multiply by b coefficients in modulus 511 (5 block RAMs 512x8).....37
5.8	Multiplication by b coefficients in Moduli 127(1 block RAMs 128x32).....38
5.9	RNS to Binary Conversion (9 block RAMs).....39
6.1	RNS Filter Implementation Procedure.....41
6.2	Xilinx Core Generator Window.....43
6.3	Single Port RAM Window.....44
6.4	Selecting the PINS for the Single Port RAM.....45
6.5	Selecting the MIF file for the Single Port RAM.....46
6.6	Final Appearance of the Single Port RAM Window.....47
6.7	Schematic Editor of Xilinx showing the RAMs as Components.....48
6.8	Xilinx Implementation of the Small Moduli RNS FIR Filter.....49
6.9	Comparison of ideal filter and actual output of Xilinx implementation.....50
6.10	Xilinx Implementation of the Large Moduli RNS FIR Filter.....51
6.11	Comparison of ideal filter and actual output of Xilinx implementation.....52
7.1	KCM Block Diagram.....54

List of Tables

Table	Page
7.1 Types of Field Programmable Gate Array (FPGA).....	55

Chapter 1

Introduction

1.1 Introduction to DSP

Signal processing can be broadly classified into two kinds Analog and Digital Signal Processing. In recent years, Digital Signal Processing (DSP) has gained a lot of popularity. DSP plays a very significant role in today's world. Some of the applications of DSP are in the fields of communication, medicine, and entertainment. In the early stages, the use of digital signals in communication devices was expensive. It was only after the invention of the microprocessor and the development of Integrated Circuits that the utilization of digital signal processing in communication was practical and feasible. The wide growth in the utilization of DSP was due to the many advantages it possesses like high noise immunity, low cost, high speed, and flexibility. The additional advantages of using equipment with digital signal processes are stability and consistency.

Digital filters are required in most communication equipment. To design the filter to meet the required specifications, many methods like Butterworth filter design[1], Chebyshev filter design [1], or Parks-McClellan filter design [1] can be used based on the type of filter required. To reduce the hardware utilized when implementing the filter, many methods have been developed for the representation of the filter coefficients.

1.2 Motivation

Since the digital filter is the most basic building block used for most DSP applications, an efficient implementation of the filter is essential to save hardware. A Digital Filter is made up of three basic components: adders, coefficient multipliers and delays. Efficient implementation of the digital filter can be achieved by selecting an efficient representation for the filter coefficient. This would help reduce the hardware required to implement the coefficient multipliers of the filter. The coefficients are implemented in Residue Number System in this thesis. The hardware implementation of the RNS filter can be done using Look-Up-Table (LUT) in either the block RAM's in FPGA's of the Xilinx Virtex FPGA's. The RAMs generated by the core generator are used for this purpose. The result is a highly efficient hardware realization of the desired FIR filter, with tradeoffs to achieve the savings.

1.3 Problem Definition

In this thesis, a hardware efficient implementation of digital filters is proposed. The coefficients of the desired filter are represented by using the number system called Residue Number System. The core generator provided by Xilinx is used to implement the filter. The schematic editor is used to represent the core generator components. VHDL code that is accepted by the core generator of the Xilinx Virtex – II board can also be used for implementation. To implement the filter coefficients in RNS, block RAMs are used. The data to be stored in the RAMs to represent the filter coefficients of the desired filter is generated using a Matlab program. The delays and the adders are also implemented using the core generator. Block RAMs are present around the Xilinx

Virtex – II board. These RAMs are generally not used by any other implementations. By utilizing the Block RAMs for implementing the coefficients, the CLBs on the chip are saved. These CLBs can be used for implementing the remaining structure of the communication system, thus increasing the utilization of the chip and making it possible to have a complex communication system on the same chip.

1.4 Thesis Organization

Chapter 2 deals with the review of different digital filters classified according to the type impulse response they have, and their advantages and disadvantages. It also deals with the different types of number representations that were previously used to represent the coefficients of the FIR filter and the previous work done to implement the filter coefficients to save hardware.

Chapter 3 introduces the Residue Number System, which will be used to represent the coefficients of the filter for implementation. Discussion about how a number is represented in Residue Number System and how to convert a given number from binary to the Residue Number System is done. The Chinese Remainder Theorem, which will be used for converting from Residue Number System back to the Binary Representation is also discussed in this chapter.

Chapter 4 talks about the Xilinx tools, which are used for simulation and implementation of the filter in hardware. The logic analyzer used to verify the hardware implementation of the filter is also discussed in this chapter.

Chapter 5 discusses the two different approaches possible for the implementation of a FIR Digital filter in the Residue Number System.

Chapter 6 contains the procedure required to follow in order to implement the filter in RNS. The Filter implementations are shown in this chapter. This chapter also deals with the simulation and implementation results of the example filters introduced earlier. Also, comparison of the Matlab simulation results, simulation of the implementation results and the results from actual implementation in hardware are provided.

Chapter 7 is the last chapter of the thesis. The results of the work done, to prove the efficiency of the method is given here. Discussion on comparison of the results with previous work done and the conclusion of the thesis are also provided in this chapter. Possible future work on the thesis topic is also discussed here.

Chapter 2

Background

2.1 Classification of Digital Filters

Digital filters are characterized by their impulse response, their transfer function or by difference equations. Digital Filters can be classified into two groups based on the type of impulse response they have, infinite impulse response (IIR) or finite impulse response (FIR). The impulse response of a filter is the response of the filter when the input signal is an impulse signal.

2.1.1 Finite Impulse Response Filters (FIR Filter)

Digital Filters [2] with finite impulse response are called Finite Impulse Response Filters[3]. FIR filters are very broadly used in communication equipment. The broad usage of FIR filters in communication is due to linear phase FIR filters are easily obtained. By making the filter coefficients symmetric, linear phase can be obtained in FIR filters, i.e. the first and the last coefficient are the same and so forth. Linear phase filters are very important, especially in devices, which handle signals carrying information in the phase. Nonlinear phase distortion can cause the information to be lost, making the signal useless. When FIR filters with symmetric coefficients are used to

implement the filter, a significant savings in hardware is also obtained, as only half of the filter coefficients have to be implemented.

A characteristic of FIR filter is that the impulse response of a FIR filter is the same as the filter coefficients. FIR filters do not have poles, they only have zeros. Hence the response of a FIR filter is only dependent on current and previous inputs and not on the output of the filter. Since a finite number of bits must be used to represent the the input, output and the coefficients of any digital filter, FIR filters can be designed with sufficient wordlength to guarantee that no rounding or truncation will be done in the multiplication of the filter input by the coefficients. This should be compared to an IIR filter, where we must always do rounding when the output is multiplied by a filter coefficient in order to prevent the wordlength from growing without bound. It is the fact that only the input is multiplied by the filter coefficients in an FIR filter that allows us to design FIR digital filters without error in the arithmetic operations. This also makes the application of RNS arithmetic particularly attractive for FIR filters.

The time response representation of FIR filter also called the difference equation and is given by

$$y_k = \sum_{i=0}^{n-1} a_i x_{k-i} \quad (2.1)$$

The transfer function of the FIR filter is given by

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{i=0}^{n-1} a_i z^{-i} \quad (2.2)$$

2.1.2 Infinite Impulse Response Filter (IIR Filter)

Digital Filters with infinite impulse response are called Infinite Impulse Response Filters [3]. There is a feedback from the output to the input in IIR filters, which makes the impulse response of IIR filters to be infinite. Due to this a finite number of bits cannot be used to represent the filter coefficients, as there is a feedback, which increases the filter coefficient bits in every cycle. Due to this, truncation and quantization noise appear in the filter, as only a finite number of bits can be used to represent the output. This one of the major disadvantages of IIR filters. Another disadvantage of IIR filters is that they have non-linear phase, which causes intersymbol interference. Due to the intersymbol interference, the information content of the signal is destroyed.

The time response representation of an IIR filter, also called the difference equation, is given by

$$y_k = \sum_{i=0}^{n-1} a_i x_{k-i} + \sum_{i=1}^{n-1} b_i y_{k-i} \quad (2.3)$$

The transfer function of the IIR filter is given by

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{n-1} a_i z^{-i}}{\sum_{i=1}^{n-1} b_i z^{-i}} \quad (2.4)$$

2.2 Different Number Systems and Previous Work Done

Many different number systems exist which have been used to implement the filters' coefficients efficiently to save hardware. Some of the number systems that were previous used and the work done on them is briefly explained here.

2.2.1 Two's Complement Number System

One of the earliest number system that was used to implement the filter coefficients is the Two's Complement Number System. In this number system, a positive number is represented by the binary representation of that number and a negative number is represented by the two's complement of the positive binary number. The two's complement of a number is obtained by first complementing the binary number and the adding a one to the least significant bit. For example the 2's complement representation of the number 5 in four digits is '0101'. The 2's complement representation of the number -5 in four digits is '1011'. The two's-complement number system can also be considered to be a binary weighted number system where the most significant bit (MSB) is weighted at -2^M , rather than $+2^M$.

Many authors [4][5][6] used the 2's complement representation and modified 2's complement representation in previous work for implementing the digital FIR filter.

2.2.2 Canonical Sign Digit

Canonical Sign Digit (CSD) is another representation that has been used very frequently for the implementation of hardware efficient digital filters. In CSD the representation with minimum number of non-zero components is chosen. When the minimum occurs for more than one representation, the representation without any consecutive non-zero-digits is chosen. The digits used to represent a number in CSD are $\{0, 1, -1\}$.

Consider for example the number 13, this can be written as

$$23 = 2^4 + 2^2 + 2^1 + 2^0 \quad (2.5)$$

the same number can also be represented as

$$23 = 2^5 - 2^3 - 2^0 \quad (2.6)$$

Hence the CSD representation of 13 can be shown in two different ways. One representation is $\{10111\}$; the other representation is $\{10-100-1\}$. We can see that the first representation requires 3 additions but the second representation only requires two subtractions. Hence the second representation is the representation for the filter.

CSD representation was used to implement FIR filters to obtain multiplierless FIR filters [7][8]. Also in the previous work that was done, the CSD representation was used along with other hardware savings methods like Scaling, Order Augmentation and Adder Extraction to further reduce the hardware. Husinga [9] and Naren [10] did work on this by developing an automated MATLAB program, which, when given the filter specifications, initially using Ramez method in MATLAB, obtains a set of coefficients. These coefficients are then converted to CSD representation and then further

optimization methods are used to reduce the hardware. The hardware implementation of the filter was also done to test the result.

2.2.3 Dempster Macleod

The Dempster and Macleod (DM) representation has been recently used to further reduce the hardware utilization during implementation. To understand the representation method of DM, an example is considered. The number 32 can be represented as

$$45 = 3 \times 15 = (4-1)(16-1) \quad (2.7)$$

Cascading can be used to use implement the multiplication and the power of two can be done by using shifting, hence requiring only two adders, where binary and CSD require 3 adders.

The DM technique was used in many works [11][12][13] done previously to reduce hardware by using the technique of cascading.

Other optimization [14] methods such as Scaling, Order Augmentation and Adder Extraction were also used along with DM representation to reduce the hardware. Work on this was done by Leong [15] who developed an automated MATLAB program, which, given the filter specifications and a cost function reduced the hardware utilization by outputting the optimum filter coefficients that met the filter coefficients. The MATLAB program also generated a package VHDL file that contains description of the filter coefficients which can be used by a VHDL program to construct the FIR filter to implement the filter. The VHDL code for hardware implementation was developed by Howe[16]. The package file along with VHDL code is combined automatically by a script file, which generates the filter for implementation.

Chapter 3

Residue Number System

3.1 Introduction

In this chapter we will introduce the Residue Number System (RNS) and the Chinese Remainder Theorem (CRT) based on [17] and [18]. The Residue Number System [19] was founded by the Chinese scholar Sun Tzu in the first Century AD, who stated the Chinese Remainder Theorem. It was in the year 1734 that Euler provided the proof for the Chinese Remainder Theorem and introduced the concept of the ‘Modulo’ operation. The residue number system has been used for a long time to implement FIR digital filters. The modulus operation is the basic operation that is used in the conversion of a number from Binary to Residue Number System. The Chinese Remainder Theorem is used for converting back to binary from Residue Number System.

3.2 The Modulo operation

In the modulus operation, the result is reset to the least value after the maximum value has been reached, i.e. the results of the modulus operation are the remainder left after division by the chosen modulus and these reminders are called ‘residues’ and they

repeat. The modulus operation [18] can be seen as in a clock where the maximum number that is reached is twelve, and beyond which the clock starts over at one and repeats itself.

Consider a binary integer 'a' and positive modulus 'm', then we have

$$b = a(\text{mod } m) \quad (3.1)$$

Here, 'b' is the remainder obtained after 'a' is divided by 'm'. The divisor 'm' is called the modulus. The integer 'b' is called the residue and is not unique as more than one remainder can be obtained during the division. The possible residues for a modulo 'm' operation are 0 to m-1. For example, we can have

$$2 = 16(\text{mod } 7) \quad (3.2)$$

but, we can also have

$$9 = 16(\text{mod } 7) \quad (3.3)$$

Two integers that have the same residue are said to be equivalent. If two integers 'a' and 'b' are equivalent, then:

$$b(\text{mod } m) = a(\text{mod } m) \quad (3.4)$$

and, we can express them as

$$b = a(\text{mod } m) \quad (3.5)$$

The main advantage with the 'mod' operation is that modular addition subtraction and multiplication is carry free.

3.3 Residue Number System

The Residue Number System is based on the Modulus Operation. In RNS [17], a number is represented by a set of Moduli, i.e. the residue representation of an integer 'x' is a set of residues $\{r_1, r_2 \dots r_n\}$, obtained by performing the 'mod' operation on the

integer 'x' using the corresponding set of moduli $\{m_1, m_2 \dots m_n\}$. The integer r_i is also defined by the set of equations

$$x = q_i m_i + r_i \quad \text{Where } i = 1, 2, \dots, n \quad (3.6)$$

The integer q_i is such that $0 \leq r_i < m_i$. Hence we can conclude that q_i is the quotient obtained from $\frac{x}{m_i}$, also denoted by $\left[\frac{x}{m_i} \right]$ and r_i is the remainder obtained from

$\frac{x}{m_i}$ or as the modulus operation performed on 'x'. The modulus of a number is not

unique hence, we define the integer r_i as the least positive remainder obtained by the division of $\frac{x}{m_i}$ and is called the *residue of x modulo m_i* , [18] also written as $x \bmod m_i$ or

as $|x|_{m_i}$.

The residue representation of a number is unique as the least positive remainder of a number when divided by any number is unique. But the converse of this statement is not true.

For example, in a three moduli system with $m_1 = 3, m_2 = 4$ and $m_3 = 5$ the residue representation of both 11 and 71 is $\{2, 3, 1\}$. This ambiguity occurs only in those numbers that meet the conditions stated by the theorem given below.

In [18] it states that, "two integers x and x' have the same residue representation for moduli $m_1, m_2 \dots m_n$ if and only if $(x - x')$ is an integer multiple of the least common multiple of the moduli, denoted by M ".

Hence, to have a unique mapping from the Residue representation, the integer 'x' must have a range of M (for positive number we would have x lie between 0 and $M-1$,

however we normally require both positive and negative numbers. In this case the range of x is different, depending on whether M is even or odd. For even M , $-\frac{1}{2}M \leq x < \frac{1}{2}M$ and for odd M $-\frac{M-1}{2} \leq x \leq \frac{M-1}{2}$).

Since the Residue Number System is based on the modulo operation, all the advantages present in the modulo arithmetic are also present in Residue arithmetic. Hence in the residue number system, addition, subtraction and multiplication are carry-free, i.e. the result of the arithmetic operation is independent of the neighboring digits. In multiplication there are no partial products, hence parallel operations can be carried out without having to wait for the results from adjacent bits.

Another factor to be considered in the residue number system is the Dynamic Range. Dynamic Range of the RNS is the product of the residue moduli. Dynamic Range is the total number of residues that can be uniquely represented. From the dynamic range of the RNS, the number of bits that will be representing the number is obtained. It is given as

$$bits = \frac{\log_{10}(\text{product of moduli})}{\log_{10}(2)} \quad (3.7)$$

The moduli $\{3, 5, 7\}$ corresponds to

$$bits = \frac{\log_{10}(3 \times 5 \times 7)}{\log_{10}(2)} \quad (3.8)$$

$$bits = \frac{\log_{10}(105)}{\log_{10}(2)} \quad (3.9)$$

$$bits = 6.7142 \quad (3.10)$$

The dynamic range must be considered because when the Residue Number System is used to implement the filter coefficients, the number of bits required to represent the filter coefficient are calculated.

3.4 Multiplicative Inverse on Modulo Operation

Multiplicative inverse [18] of a number 'b' with modulo 'm' is represented as $\left| \frac{1}{b} \right|_m$. The inverse of a number then multiplied by the number itself is 1. This property is used to find the multiplicative inverse of a given number. For example, the inverse of $|5|_9$ i.e. $\left| \frac{1}{5} \right|_9$ can be found as shown below.

Let $|x|_9 = \left| \frac{1}{5} \right|_9$, then based on the above explained property,

$$\left| |5|_9 \times |x|_9 \right|_9 = |1|_9 \quad (3.11)$$

This implies that $|x|_9 = 2$, as $0 \leq |x|_9 \leq 8$ and $2 \times 5 \bmod 9 = 1$.

3.5 Chinese Remainder Theorem

In this section discussion on how to convert from the Residue Number System is discussed. The conversion is done by using the Chinese Remainder Theorem, which is stated in [18] as, "For a given residue number representation $\{r_1, r_2 \dots r_n\}$, we can determine the integer 'x' using the Chinese Remainder Theorem, if the moduli $\{m_1, m_2 \dots m_n\}$ are mutually prime", i.e. the greatest common divisor for the moduli is '1'.

The Chinese Remainder Theorem states that

$$|x|_M = \left| \sum_{j=1}^n \hat{m}_j \left| \frac{r_j}{\hat{m}_j} \right|_{m_j} \right|_M \quad (3.12)$$

where

$$\hat{m}_j = \frac{M}{m_j} \quad (3.13)$$

$$M = \prod_{j=1}^n m_j \quad (3.14)$$

and

$$(m_j, m_k) = 1 \quad \text{for } j \neq k \quad (3.15)$$

In the equations above, in the notation $|x|_M$, 'x' is the integer obtained from the Chinese Remainder Theorem.

3.6 An Example using Residue Number System

An example is considered to clearly explain and to provide a better understanding of the Residue Number System and the Chinese Remainder Theorem. Consider the integer $x = 35$ and the moduli $m_1 = 3$, $m_2 = 4$ and $m_3 = 5$. The residue representation for the number in this moduli system is $\{2, 3, 0\}$.

To convert back from residue representation the Chinese Remainder Theorem is used. The product of the moduli, $M = 3 \times 4 \times 5 = 60$. The weights are now calculated.

$$\hat{m}_1 = \frac{M}{m_1} = \frac{60}{3} = 20, \quad \hat{m}_2 = \frac{M}{m_2} = \frac{60}{4} = 15, \quad \hat{m}_3 = \frac{M}{m_3} = \frac{60}{5} = 12 \quad (3.16)$$

Now to find $\left| \frac{r_j}{\hat{m}_j} \right|_{m_j}$

$$\left| \frac{r_j}{\hat{m}_j} \right|_{m_j} = \left| r_j \right|_{m_j} \times \left| \frac{1}{\hat{m}_j} \right|_{m_j} \quad (3.17)$$

Now we need to find the inverse of \hat{m}_j to find $\left| \frac{r_j}{\hat{m}_j} \right|_{m_j}$.

The inverses of $\hat{m}_1, \hat{m}_2, \hat{m}_3$ are

$$\left| \frac{1}{\hat{m}_1} \right|_{m_1} = \left| \frac{1}{20} \right|_3 \Rightarrow \left| 20 \right|_3 \times \left| X_1 \right|_3 = \left| 1 \right|_3 \quad (3.18)$$

$$\Rightarrow \left| 2 \right|_3 \times \left| X_1 \right|_3 = \left| 1 \right|_3 \quad (3.19)$$

$$\Rightarrow \left| 2 \times X_1 \right|_3 = \left| 1 \right|_3 \quad (3.20)$$

$$\Rightarrow \left| X_1 \right|_3 = 2 = \left| \frac{1}{20} \right|_3 \quad (3.21)$$

Similarly,

$$\left| \frac{1}{\hat{m}_2} \right|_{m_2} = \left| \frac{1}{15} \right|_4 = 3 \quad (3.22)$$

$$\left| \frac{1}{\hat{m}_3} \right|_{m_3} = \left| \frac{1}{12} \right|_5 = 3 \quad (3.23)$$

Finally to find to $|x|_M$,

$$|x|_M = \left| 20 \times \left| 2 \times 2 \right|_3 + 15 \times \left| 3 \times 3 \right|_4 + 12 \times \left| 0 \times 3 \right|_5 \right|_{60} \quad (3.24)$$

$$= \left| 20 \times \left| 1 \right|_3 + 15 \times \left| 1 \right|_4 + 0 \right|_{60} \quad (3.25)$$

$$= |35|_{60} \tag{3.26}$$

Thus, we obtain the integer number $x = 35$, uniquely from its residue representation using the Chinese Remainder Theorem.

Chapter 4

Xilinx Virtex FPGAs

4.1 Introduction to Field Programmable Gate Arrays

A Field Programmable Gate Array (FPGA) [20] [21] is a programmable VLSI chip that is used for rapid prototyping to test digital designs. FPGA's are made up of many identical blocks and interconnects which can be programmed by the user to obtain the desired design. Implementing and testing designs in FPGA's can avoid the cost and time involved in manufacturing an ASIC chip. FPGA's can be reprogrammed many number of times to test many designs, hence any design errors can be corrected easily without much effort or cost such as involved in wafer fabrication. All these advantages make the usage of FPGA's often would be a better choice for testing over conventional masked gate array or ASIC's.

There are four different kinds of FPGA's [21] based on the type of architecture: Symmetrical array, Row-based, Hierarchical PLD, and Sea-of-gates as shown in Fig 4.1 below.

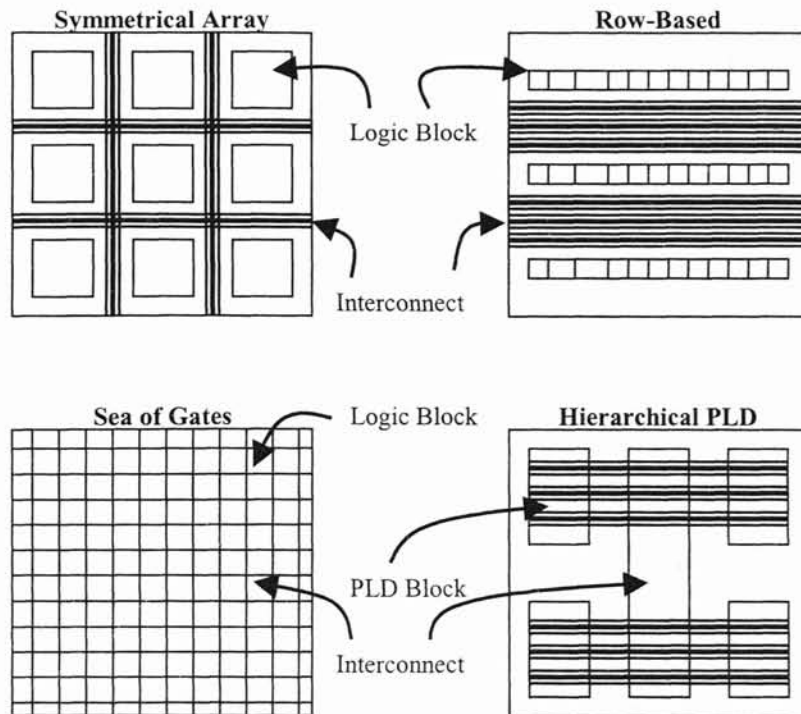


Figure 4.1: Types of Field Programmable Gate Array (FPGA) [22].
 (Printed with permission from the website <http://www.vcc.com/fpga.html>)

There are also four different technologies [21] that are used currently to program FPGAs. These technologies are: Static RAM cells, Anti-fuse, EPROM transistors and EEPROM transistors, with each technology having its own advantage based on the application.

When the Static RAM FPGA's are programmed, the circuit is built into the FPGA chip by making connections in the chip using pass-transistors, transmission gates, or multiplexers. These different connectors are controlled, i.e. turned on or off by the Static RAM cells. Due to the usage of Static RAM cells, large chips are required, but this also has the advantage of having the ability to program within the circuit.

In this technology, anti-fuse programmable connections are used. When programmed, these connections go into low impedance in accordance with the requirements of the circuit. A disadvantage of this technology is that it can be programmed only once, but it is much cheaper than the Static RAM technology.

The principle behind the EPROM and EEPROM transistors is the same as that used in memories. Electrical impulses or ultraviolet light, depending on the type of RAM being used, is used to erase the RAM to reprogram it. These FPGA chips can be reprogrammed without external storage of the configuration. The EPROM transistors cannot be re-programmed within the circuit, however, and must be taken out of the circuit for programming.

An FPGA chip provided by Xilinx [23] is used for the implementation of the Digital FIR Filter designed in this thesis. The Xilinx Virtex FPGA chip uses Static RAM FPGA programmable connections. Details about the Xilinx Virtex FPGA architecture are discussed in the next section of this chapter.

4.2 Xilinx Virtex FPGA

The Xilinx Virtex FPGA [24] chip uses Static RAM FPGA's. Internal memory cells are present on the chip, which store the user's circuit configuration. This configuration is programmed onto the FPGA chip to obtain the user's design. The overview of the Virtex FPGA architecture is shown in Fig. 4.2.

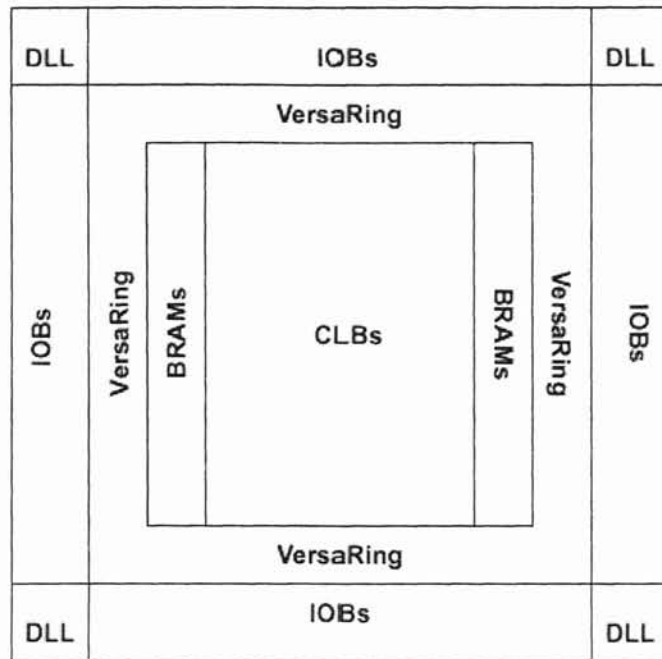


Figure 4.2: Virtex Architecture Overview [24].

From the overview it can be seen that the Virtex FPGA is mainly comprised of an array of Configurable Logic Blocks (CLB) and programmable Input/Output Blocks (IOB) which are interconnected by a hierarchy of fast and versatile routing. The Virtex chip also consists of on-chip true dual-read/write synchronous Block RAMs.

4.2.1 Configurable Logic Block

The basic building block of the Configurable Logic Block [24] (CLB) is a Logic Cell. Figure 4.3 shows the structure of the Virtex CLB. Each CLB consists of 2 Slices, which consists of two logic cells. The logic cell is comprised of a 4-input function generator, carry logic and a storage element. The function generators are implemented by using 4-input look up tables (LUTs). Each LUT can also provide a 16 x 1-bit synchronous RAM or a 16-bit shift register. The storage elements present in the CLB can

be programmed to behave as edge-sensitive D-type flip-flops or level-sensitive D-type latches. In addition to clock and clock – enable signal, the set and reset signals are also provided for each slice. Separate carry control logic is present, which enables, the implementation of high-speed arithmetic functions to provide fast arithmetic carry.

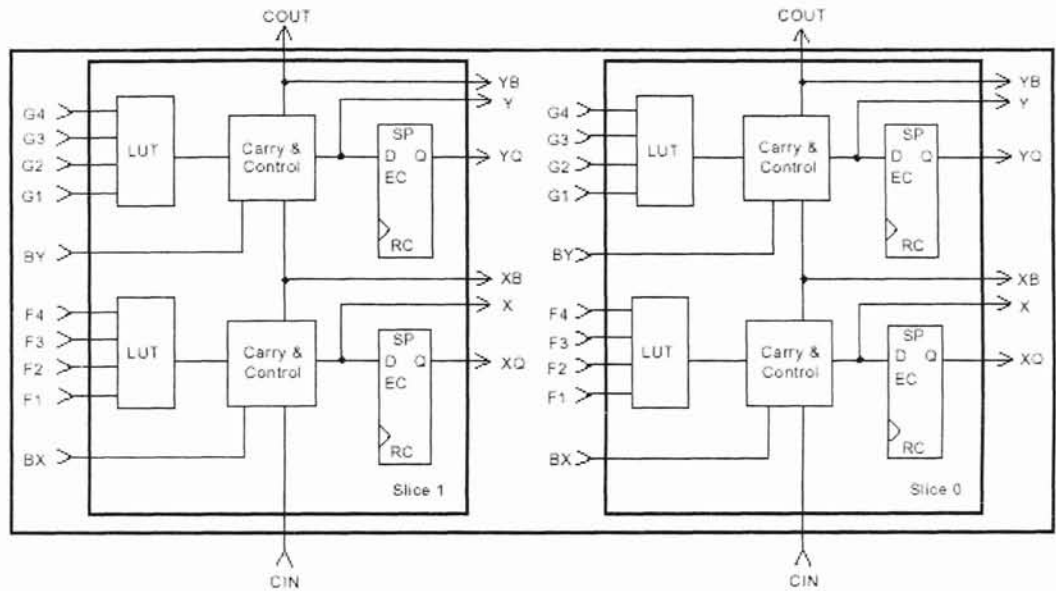


Figure 4.3: 2-Slice Virtex Configurable Logic Block (CLB) [24].

4.2.2 Input/Output Block

The Virtex Input/Output block (IOB) [24] is compatible with a wide range of input/output signaling standards. The IOB block is the interface between the internal circuitry and the external pins. The Block figure of the Input/Output Block is shown in Figure 4.4.

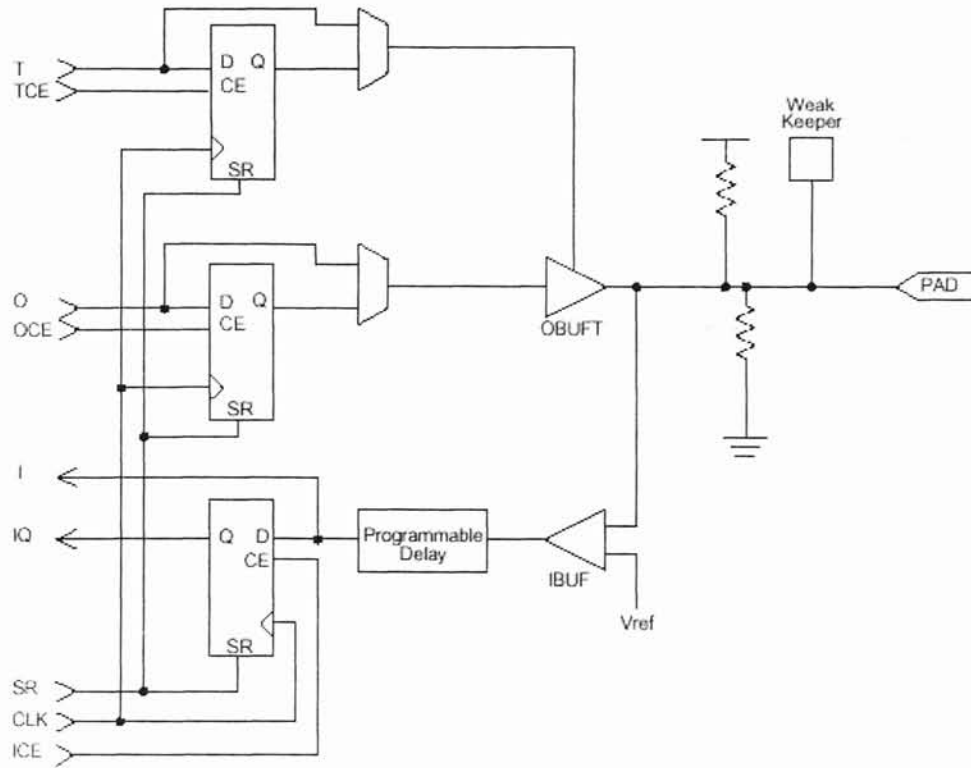


Figure 4.4: Virtex Input/Output Block (IOB) [24].

4.2.3 Block Select RAM

The Xilinx Virtex chip also consists of blocks of true dual-read/write port synchronous RAM, with 4096 memory cells called the Block SelectRAM [24]. Block RAMs are organized into columns on both the sides of the Xilinx Virtex FPGA chip. Block RAMs are present on all Virtex FPGA chips. There are a total of 32 block Select RAMs in the Virtex FPGA. The most appealing feature of the block RAM is that all the ports of the Block RAMs can be configured to a read/write port, a read port, or a write port independently. A generic Dual-port Block RAM and Single Port Block RAM are shown in Figures 4.5 and 4.6 respectively.

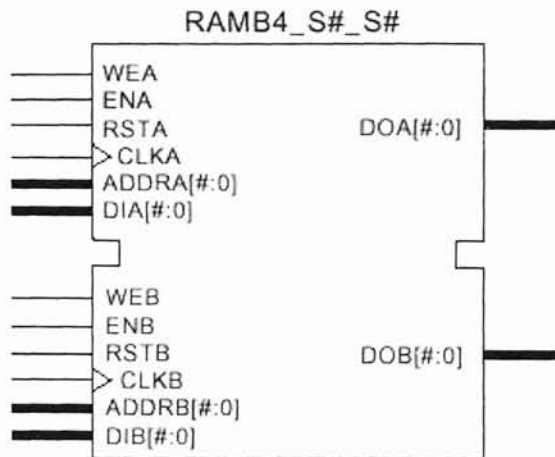


Figure 4.5: Dual-Port Block SelectRAM [24].

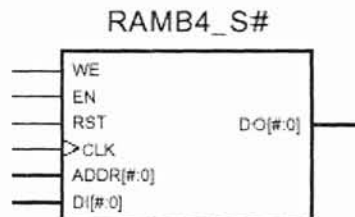


Figure 4.6: Single-Port Block SelectRAM [24].

Some of the characteristics of Block RAMs are: The block RAM is edge-sensitive, i.e. all the inputs in the Block RAMs are read with the rising edge of the clock. Also the setup and hold time clock timing specifications have to be satisfied. Similarly there is a clock to output time delay specification. Static RAM memories are used to implement Block RAMs and there is no combinational path for address to the output, the Lookup tables in the CLBs are used to perform this function. Read and write operations require only one clock edge. The output port maintains the previous value it has outputted till another operation is performed to change the output.

The Block RAMs can be invoked by using Schematic Editor or HDL description language and turning off the synthesizer during synthesis of the block RAM code in the HDL description of the code.

Chapter 5

FIR Filter in Residue Number System

5.1 Introduction

Chapter 5 discusses the two different approaches possible for the implementation of a FIR Digital filter in Residue Number System.

5.2 Basic Concept

Figure 5.1 shows a block diagram of a generic RNS FIR digital filter. The implementation using this RNS technique proceeds step by step as shown in the block diagram in Figure 5.1. The first conversion of the binary input to RNS is done by applying the appropriate moduli. The input is multiplied independently in each moduli with the coefficients (also expressed in RNS) and the filter is generated for each of the moduli. Finally, the end conversion back to binary from RNS is done, which generates the binary output of the filter based on the Chinese Remainder Theorem.

The basic concept in the RNS FIR digital filters is the fact that once conversion takes place from a single n -bit binary input to a series of RNS moduli whose product

spans the range of the n -bit binary input, it is possible to process the FIR filter independently in each of these smaller moduli without interaction between the arithmetic within the moduli. Hence, a 16-bit input word can be converted into a number of moduli that are 5-bit or less in which the hardware required to implement the filter is much less than in a 16-bit implementation. There are two radically different approaches to implementing RNS filters.

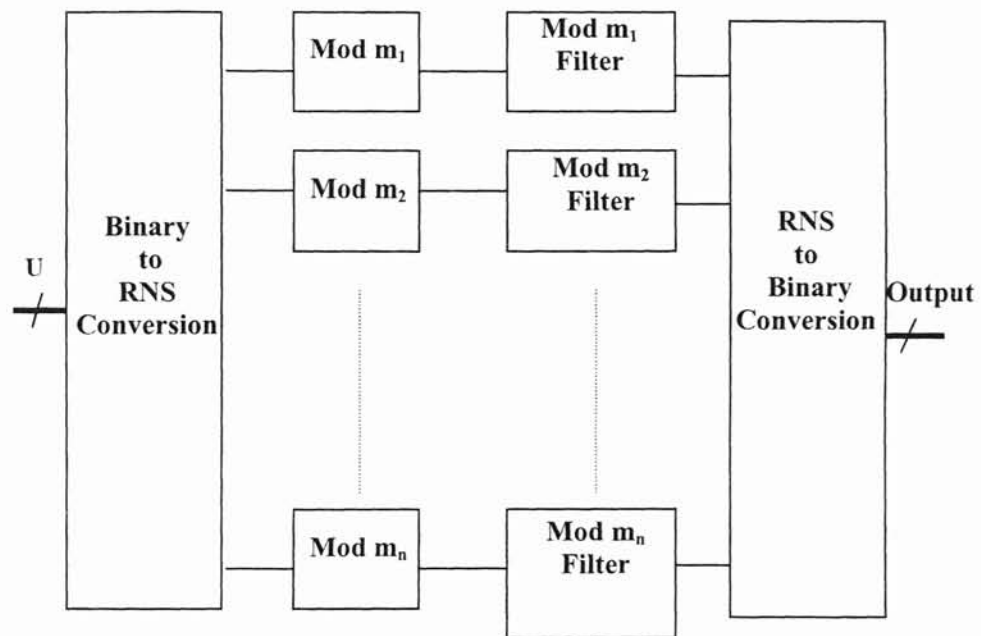


Figure 5.1: RNS Digital FIR Filter.

One uses large moduli, one of which is a power of two and the others are a power of two minus one. These large moduli are chosen to have a product that spans the 16-bit input plus the coefficient width. The other RNS approach uses small moduli that typically consist of consecutive prime numbers starting with three and has a product sufficient to span the 16-bit input number plus the coefficient bit size. Usually one of the

numbers is not a prime, but rather a power of two and often it is convenient to combine some of the smaller prime moduli into larger moduli by multiplying them together to form a larger modulus. Large moduli systems where all moduli are either a power of two or a power of two minus one have the advantage that conversion to and from binary is generally easy and requires only adders and a few look-up-tables (LUT's).

5.3 Modulo Addition

The implementation of an FIR filter in the Residue Number System requires the implementation of the filter in each of the moduli. This necessitates the requirement of 'modulo-adders' in the moduli that the filters are implemented i.e. when '511' is used as one of the modulo to implement the filter in RNS, a 'modulo-511 adder' is required. The arithmetic in the modulus, which is 2^N , is standard 2's complement arithmetic and the arithmetic in 2^N-1 moduli is 1's complement arithmetic. Hence an N-bit 2's complement adder behaves as a ' 2^N modulo adder' and an N-bit 1's complement adder behaves as a ' 2^N-1 modulo adder'. It can be then concluded that a 'modulo-511 adder' can be implemented by using a 9-bit, 1's complement adder.

When a 'modulo-adder' for a modulus other than 2^N or 2^N-1 is needed, a 2's complement adder can be used with additional circuitry to loop-back the values after the maximum value has been reached.

5.4 The Example Filter

The example filter used is an 8-tap fixed-coefficient linear-phase FIR filter with 16-bit input, 14-bit coefficients and 30-bit output. The example filter has been optimized

for implementation in Xilinx Application Notes [26] this provides a comparison for the Residue Filter. The pass-band ripple $r_p=0.1222$ ($\alpha_p=3\text{db}$), pass-band edge $\omega_p=0.3$, stop-band ripple $r_s=0.01$ ($\alpha_s=40\text{db}$) and $\omega_s=0.65$. Figure 5.2 shows the frequency response of this ideal filter as simulated in MatLab.

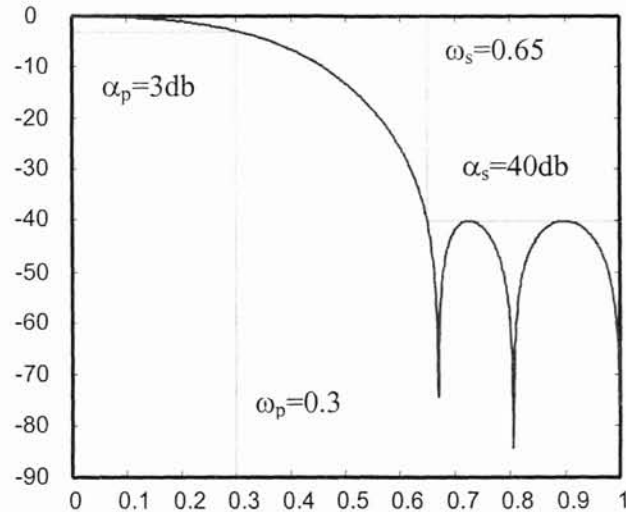


Figure 5.2: Example Digital FIR Filter Response.

The filter coefficients are $b=\{-0.0240156, 0.0055081, 0.1661459, 0.3523616, 0.3523616, 0.1661459, 0.0055081, -0.0240156\}$. This filter is implemented in a large modulus RNS and a small modulus RNS.

5.5 Small Moduli RNS FIR Example

The example filter shown in the previous section is first implemented in the Small moduli system [27]. The moduli that were chosen to implement the filter are (4,13,29,31). The dynamic range is calculated for the chosen moduli to be,

$$bits = \frac{\log_{10}(\text{product of moduli})}{\log_{10}(2)} \quad (5.1)$$

$$bits = \frac{\log_{10}(4 \times 13 \times 29 \times 31)}{\log_{10}(2)} \quad (5.2)$$

$$bits = \frac{\log_{10}(46748)}{\log_{10}(2)} \quad (5.3)$$

$$bits = 15.51 \quad (5.4)$$

The original example has a 16-bit input and 14-bit coefficients, which results in a 30-bit output. Our simplified filter supports an 8-bit input with 7.51-bit coefficients, which results in a 15.51-bit output. The example FIR filter has symmetric coefficients, hence only half the filter coefficients have to be implemented. The block diagram of the simplified 4-moduli RNS FIR filter is shown in Figure 5.3.

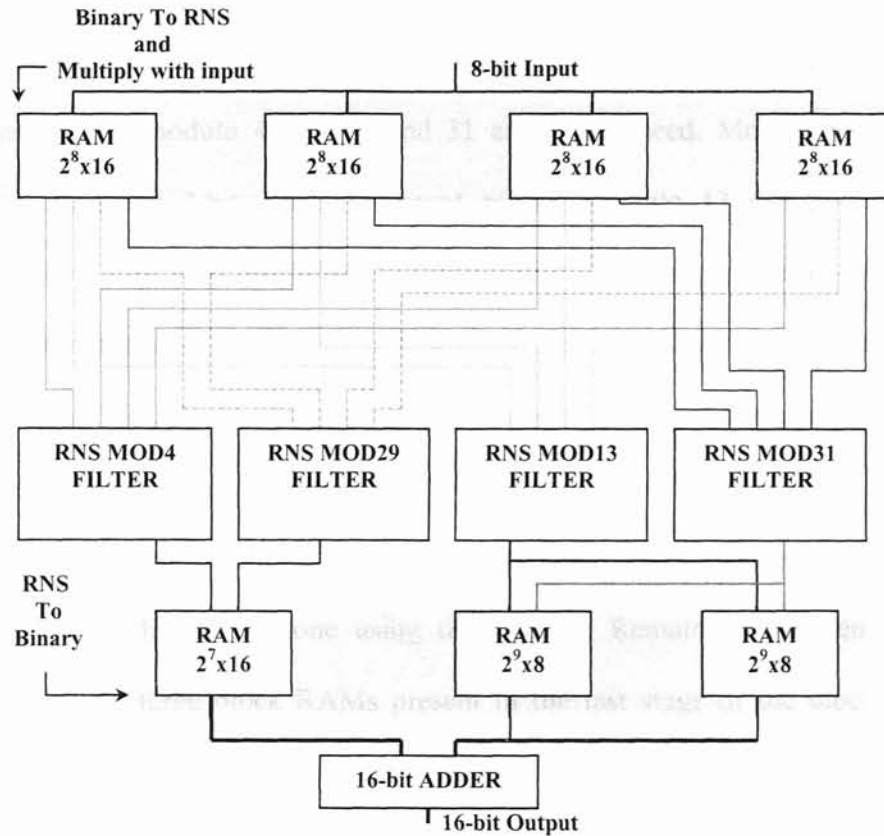


Figure 5.3: Block Diagram of 4-moduli RNS Digital FIR Filter.

The conversion from Binary to Residue Number System is done using the first four block RAMs. The multiplication with the input is also done in these block RAMs. In each of the $2^8 \times 16$ block RAMs, the 8-bit input of the simplified FIR filter is taken in as the input for the RAM. Each of the RAMs is designed such that the output of the block RAM is the product of the input and the coefficient represented in the moduli (4,13,29,31). For example, the first block RAM is designed such that the 16-bit output of the Block RAM consists of the product of the filter input and the first coefficient represented in the moduli (4,13,29,31). Hence, each of the blocks has one of the four coefficients multiplied by the input and represented in the moduli (4,13,29,31).

The moduli coefficients from each of the block RAMs are combined to form each of the modulo filter, as shown in the next section of Figure 5.3. To implement the filter in each of the moduli, modulo 4, 13, 29 and 31 adders are need. Modulo 4 adders are implemented by using 2-bit 2's complement adders, Modulo 13 and 29 adders are implemented by using 4-bit and 5-bit 2's complement adders respectively with a RAM connected to the output of each adder to loop-back the output after the maximum value has been reached. Modulo 31 adders are implemented by using 1's complement 5-bit 1's complement adders. In the next stage, the RNS output of the different filters is converted back to binary and combined to get the output of the filter. The conversion from Residue Number System to Binary is done using the Chinese Remainder Theorem. This is implemented in the three block RAMs present in the last stage of the block diagram shown in Figure 5.3. The binary output obtained from the Block RAMs is combined using a 16-bit adder, to generate the final output of the FIR filter.

5.6 Large Moduli RNS FIR Example

The example filter is now implemented in Residue Number System using Large Moduli [28]. The large modulus RNS is based on moduli that are powers of two or powers of two minus one. The moduli that are chosen to implement the FIR filter in the large moduli are $\{511, 256, 255, 127\}$. The moduli are mutually prime to satisfy the condition to use the Chinese Remainder theorem to convert back to binary. The dynamic range for the moduli chosen is calculated for the chosen moduli,

$$bits = \frac{\log_{10}(\text{product of moduli})}{\log_{10}(2)} \quad (5.5)$$

$$bits = \frac{\log_{10}(511 \times 256 \times 255 \times 127)}{\log_{10}(2)} \quad (5.6)$$

$$bits = \frac{\log_{10}(4236476160)}{\log_{10}(2)} \quad (5.7)$$

$$bits = 31.98 \quad (5.8)$$

The dynamic range of the filter using the moduli {511, 256, 255, 127} is 31.98, which satisfies the requirements to represent the example filter with 30-bit output. The Large moduli RNS FIR filter that is implemented in this section supports a 16-bit input with 15.98-bit coefficients, which results in a 31.98-bit output. The example FIR filter has linear phase characteristics and hence it symmetric coefficients, so only half of the filter coefficients have to be implemented to implement the entire filter.

The first step in the implementation of the large moduli filter is the conversion from the RNS of the input. Figure 5.4. Shows the details of the binary to RNS conversion.

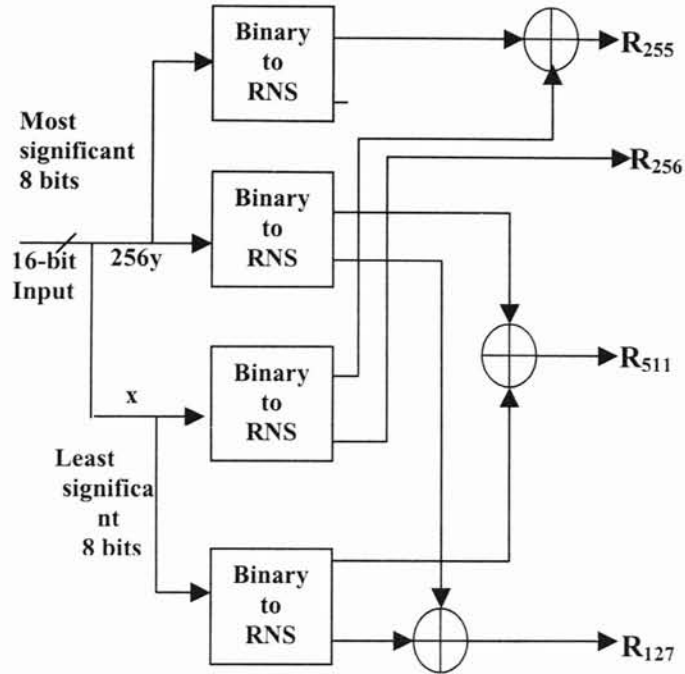


Figure 5.4: Binary to RNS (4 block Rams 256x16 and 3, 2's comp adders).

The conversion is done separately for the Least Significant 8-bits and Most Significant 8-bits. The bits are then combined after the conversion is done using adders. The three adders from top to bottom are in modulus 255, modulus 511 and modulus 127. Hence, each can be implemented with a one's-complement adder. An 8-bit 1's complement adder is used for combining the LSB and MSB of modulo 255, 9-bit 1's complement adder is used to combine the LSB and MSB of modulo 511 and 7-bit 1's complement adder is used to combine the LSB and MSB of modulo 127 respectively. The MSB of modulo 256 become zero, hence no adder is required to combine the bits of modulo 256. The Binary to RNS boxes are Block Ram LUT's each 256x16.

In the next stage, the filter input that was converted into modulo in the previous section is now multiplied with the filter coefficients. Each of the moduli outputs a total of

32 bits, eight bits for each coefficient. Figures 5.5 and 5.6 show the multiplication of the modulus 256, modulus 255 inputs by the modulus 256 and modulus 255 coefficients. Moduli 255 and Moduli 256 require two RAMs of size 256 X 16.

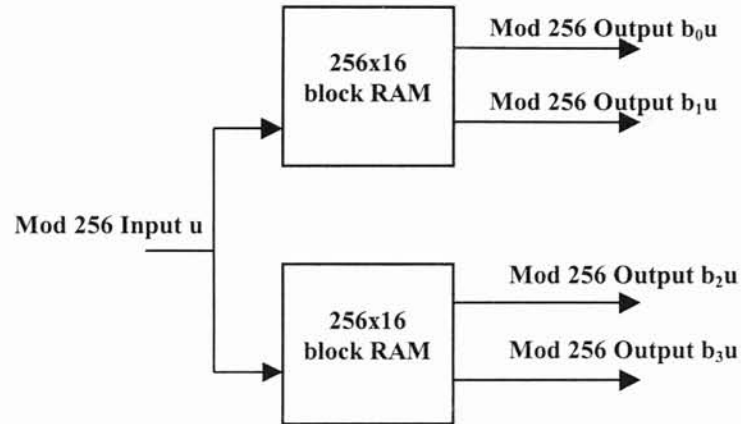


Figure 5.5: Multiplication by b coefficients in Moduli 256 (2 block RAMs 256x16).

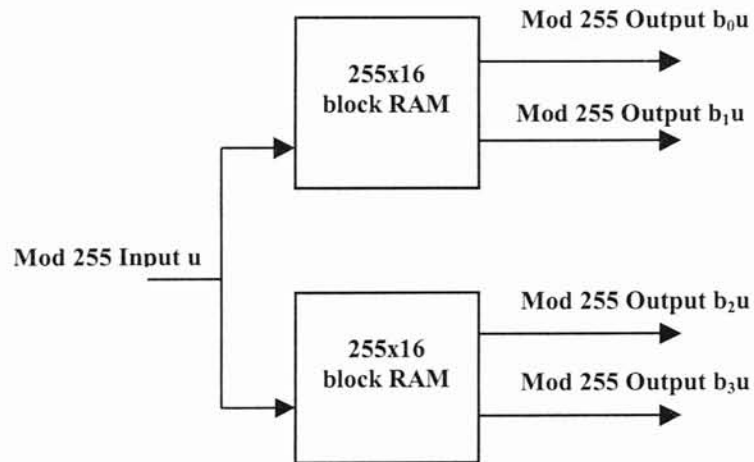


Figure 5.6: Multiplication by b coefficients in Moduli 255 (2 block RAMs).

In the Figure 5.7 the multiplication of the modulus 511 inputs, by the four-modulus 511 coefficients is shown.

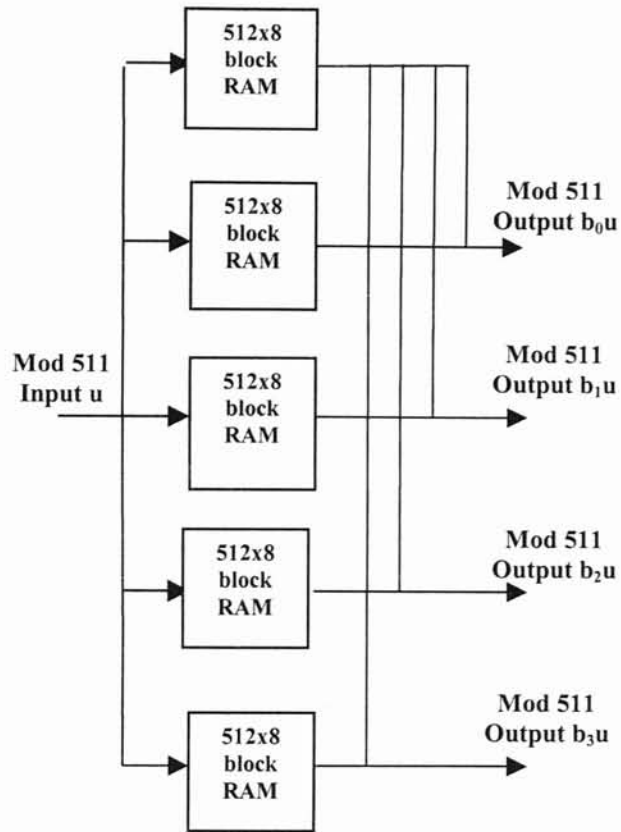


Figure 5.7: Multiply by b coefficients in modulus 511 (5 block RAMs 512x8).

The multiplication of the input by the coefficient for each of the four coefficients requires five RAMs of size 512 x 8. Hence the output is a total of 40 bits from the five RAMs, of which only 36 bits are utilized, i.e. nine for each of the four coefficients. The RAMs are designed such that 8-bits are obtained from the first four, 512 x 8 block RAMs and the 9th bit for each of the coefficient products is obtained from the last 512 x 8 block RAM.

For the last Moduli 127 only one RAM of size 128 x 32 is required. The requirement is only 28 bits, of the 32 bits seven bits for each of the four coefficients. This is shown in Figure 5.8.

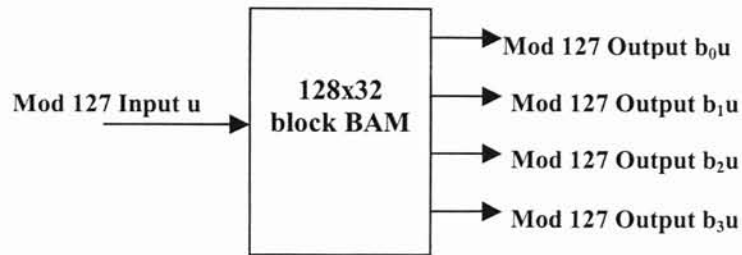


Figure 5.8: Multiplication by b coefficients in Moduli 127(1 block RAMs 128x32).

In the next stage the FIR filter is implemented in each of the four moduli. Hence four filters each in one of the moduli are implemented. Delays and adders are used to build the FIR filter in the normal fashion. Specific adders have to be used for the different moduli filters based on the requirements of the modulo. For moduli 511, 255 and 127, each of the adders is a 1's-complement adder of 9, 8 and 7 bits respectively. For modulus 256, the adders are 2's-complement adders of 8 bits. The adders and delays can be combined to use a single component called the 'adder-delay'. Each adder with delay takes LUT's and FF's equal to the number of outputs of the adder.

The last stage of the implementation of the FIR filter in RNS is the conversion of the 4 modulo filters from RNS back to binary using Chinese Remainder Theorem and to combine the outputs to get the final output of the filter. Figure 5.9. shows the conversion from RNS to Binary, which requires 4 Block RAMs. The number of output bits is 32.

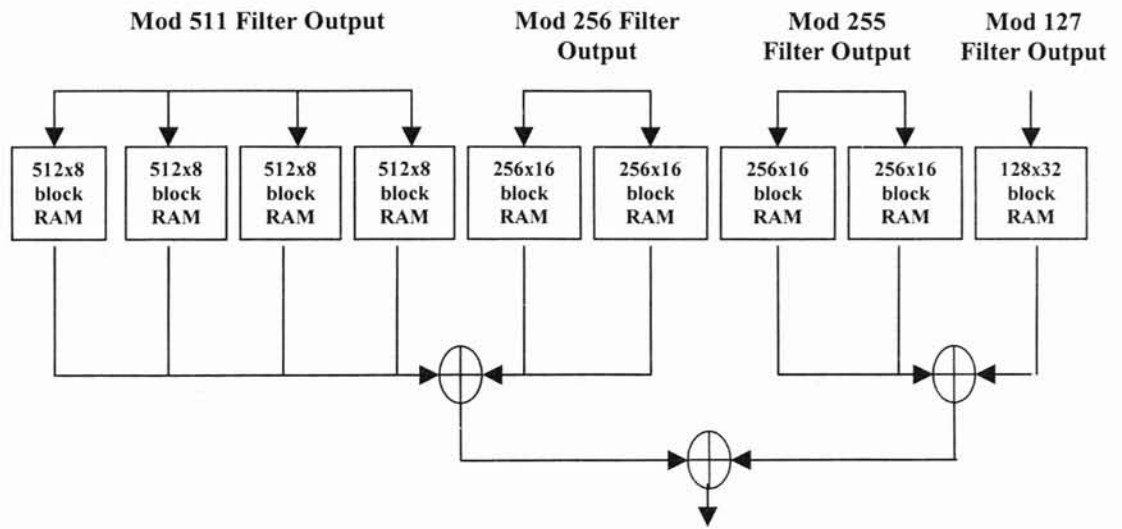


Figure 5.9: RNS to Binary Conversion (9 block RAMs).

This is the final output of the FIR filter implemented in the Residue Number System. The design and implementation of the Block RAMs and the complete implementation of the FIR filter are discussed in the next chapter.

Chapter 6

Hardware implementation of the Filter

6.1 Introduction

Chapter 6 contains the procedure required to implement the filter in RNS. The Filter implementations are shown in this chapter. This chapter also deals with the simulation and implementation results of the filters introduced earlier. A comparison of the Matlab simulation results of the original example filter and simulation of the implementation results of the small moduli and large moduli FIR filters are also shown in this chapter.

6.2 Implementation Procedure

Implementation of the RNS FIR filter can be achieved through schematic layout or by writing VHDL code. The Schematic Layout Editor is used in this thesis to implement the FIR filter. The RAMs are implemented using the core generator. The data to be stored in the RAMs can be added by using a .mif file. The .mif file contains the converted RNS value for its corresponding binary value, the value obtained after multiplying the input by the coefficient, or the RNS output converted back to binary at

the end. The file is included when generating the RAMs. A MATLAB program is used to generate the .mif file. The adder-delays and the adders are also generated by using the core generator. The top-level file uses these components to generate the filter. This is depicted in the Flowchart in Figure 6.1 to provide a better understanding of the concept.

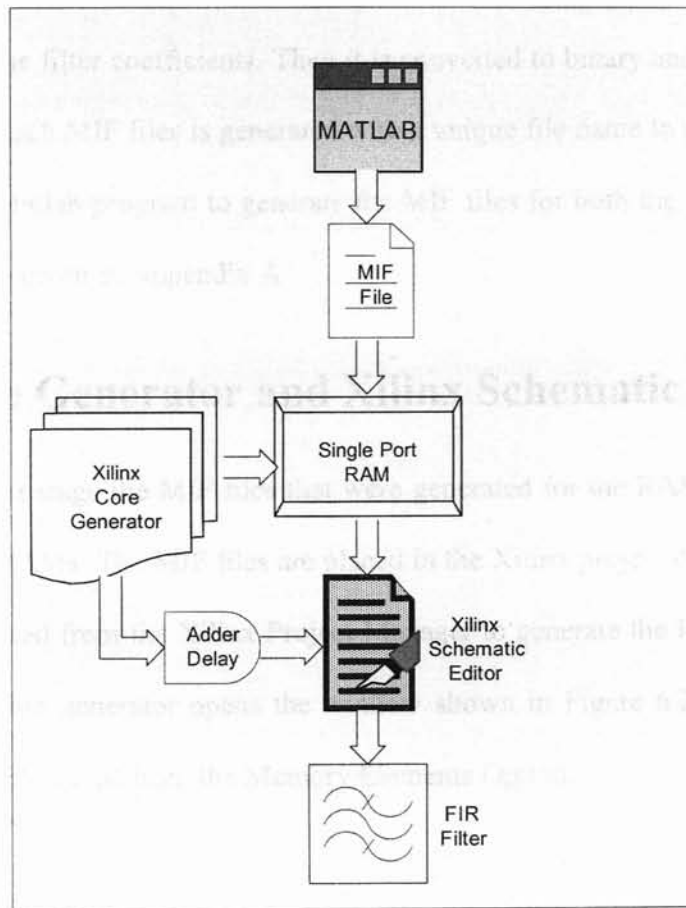


Figure 6.1: RNS Filter Implementation Procedure.

6.2.1 MIF File Generation

A MIF file consists of 0's and 1's, which are the values to be stored in the Block RAMs. Each MIF file is generated to meet the specific requirements of the individual RAM. For Example, in the Small Moduli case, to generate the RAMs in the first stage of the RNS filter, the MATLAB program converts the decimal input to RNS and also multiplies it by the filter coefficients. Then it is converted to binary and stores the results in the MIF file. Each MIF files is generated with a unique file name to distinguish it from the others. The Matlab program to generate the MIF files for both the Small Moduli and Large Moduli are given in Appendix A.

6.2.2 Core Generator and Xilinx Schematic Editor

In the next stage the MIF files that were generated for the RAMs are loaded into their respective RAMs. The MIF files are placed in the Xilinx project directory. The Core Generator is opened from the Xilinx Project Manager to generate the RAMs required for the filter. The Core generator opens the window shown in Figure 6.2. The Single Port Block Memory is selected from the Memory Elements Option.

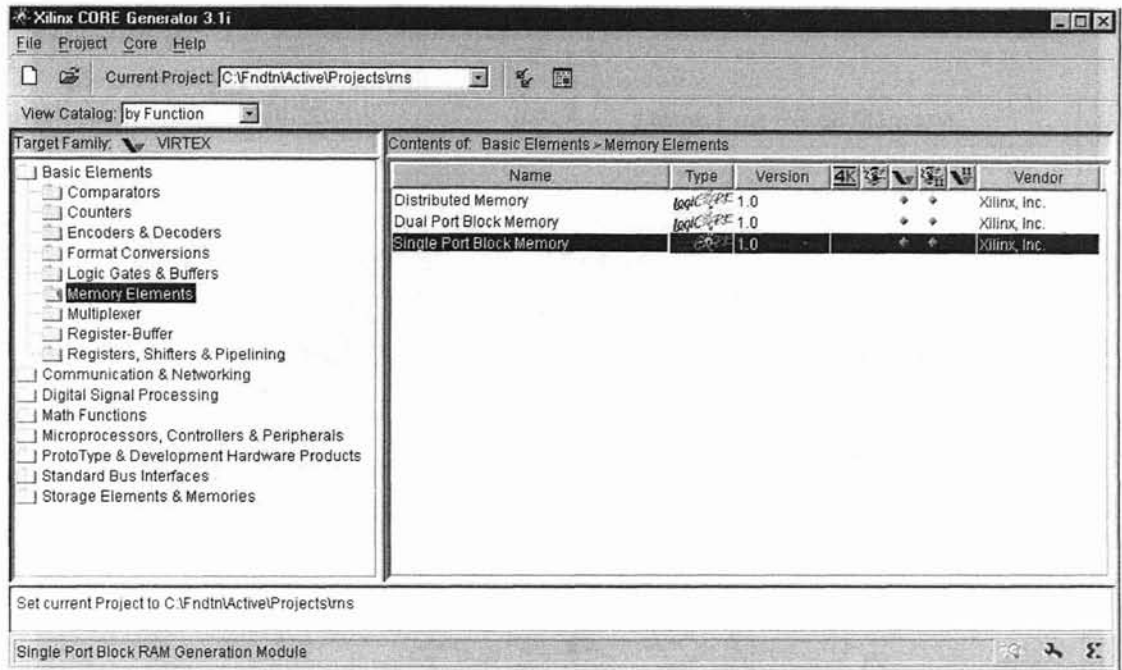


Figure 6.2: Xilinx Core Generator Window.

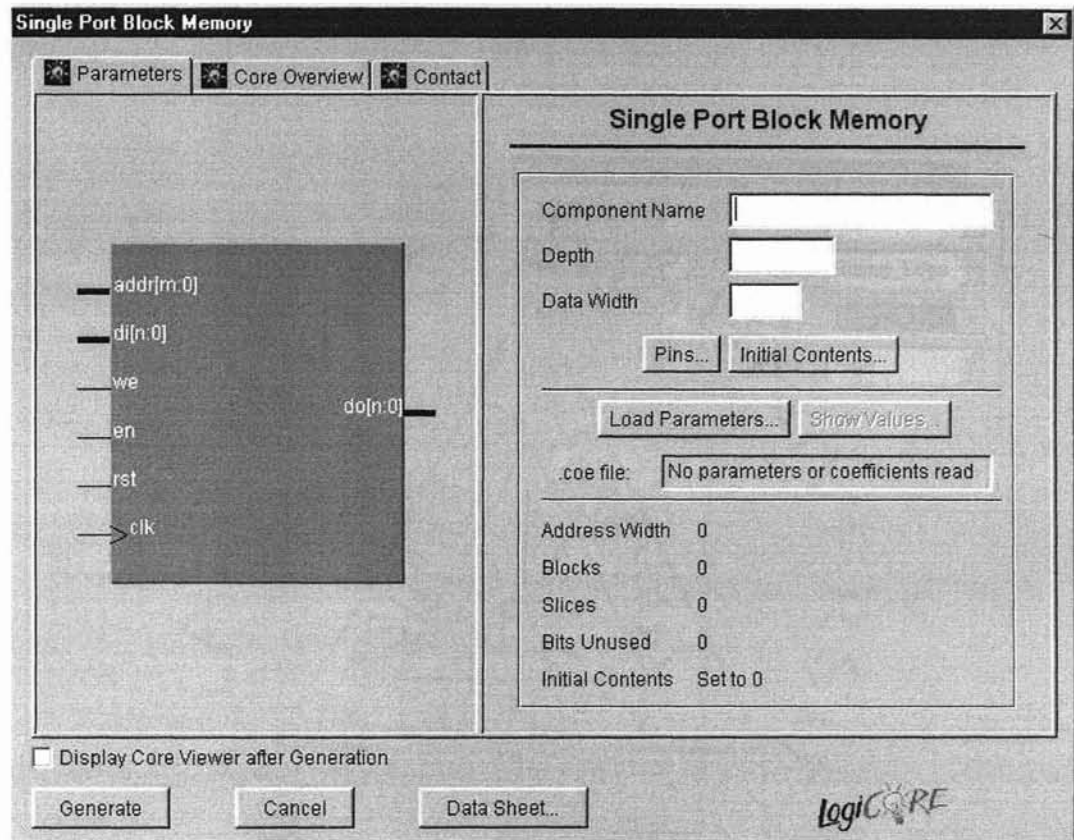


Figure 6.3: Single Port RAM Window.

A window as shown in Figure 6.3, appears when the Single Port Block Memory is selected. The component name in the window distinguishes one RAM from another when the RAM is generated, the Depth specifies the number of entries present in the RAM and hence the number of bits in the address lines to the RAM. For example, if we specify a Depth of '16' then there will '4' address lines since $2^4 = 16$. The Data Width specifies the number of bits in the output. The default values stored in the Block RAM is Zero. The number of Slices and Block RAMs used on the Virtex Board to generate the particular Block RAM is also shown in the window.

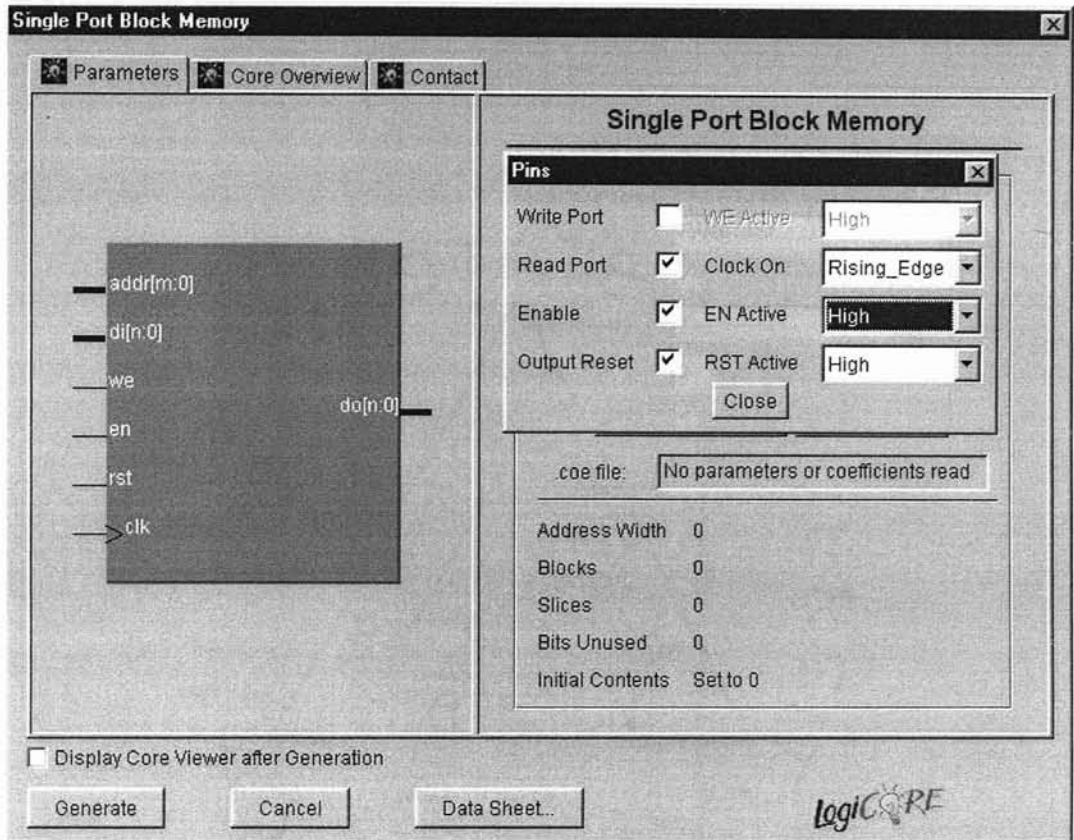


Figure 6.4: Selecting the PINS for the Single Port RAM.

The 'Pins' button on the 'Single Block Memory' Window is selected to select the desired pins. This opens a smaller 'Pins' Window as shown in Figure 6.4. The Write Port is not selected as the RAM is being used as 'Read Only'. The Read Port, Enable and Output Reset is activated. The RAM is set to Clock in the address in the address lines on the Rising_Edge of the clock. The Enable and Reset are set for Active High. The close button is pressed to save the options selected for the RAM.

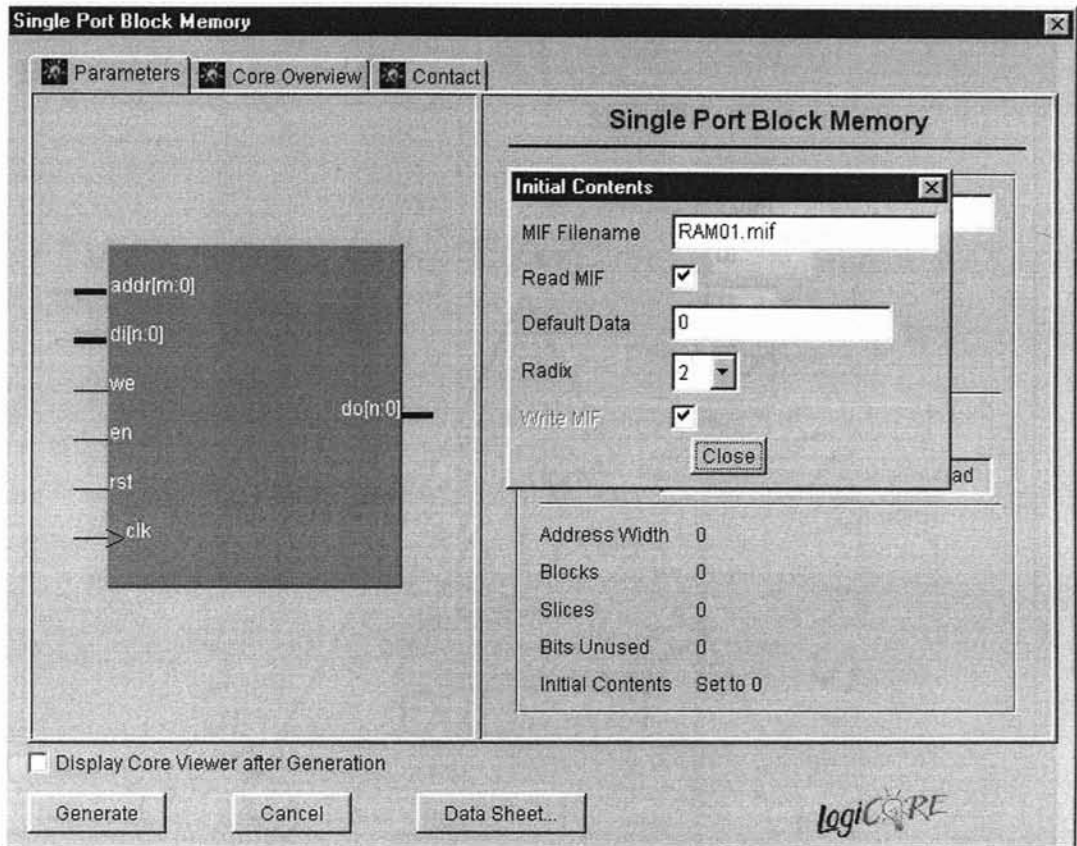


Figure 6.5: Selecting the MIF file for the Single Port RAM

The 'Initial Contents' button on the 'Single Block Memory' Window is pressed to load the MIF file for the RAM. This opens a smaller 'Initial Contents' Window, as shown in Figure 6.5. opens to load the MIF file. The MIF file relevant to the RAM being generated is specified in the MIF Filename option, the Read MIF file is selected and Radix 2 is selected as the data present in the MIF file in binary. The close button is pressed to save the options selected for the RAM.

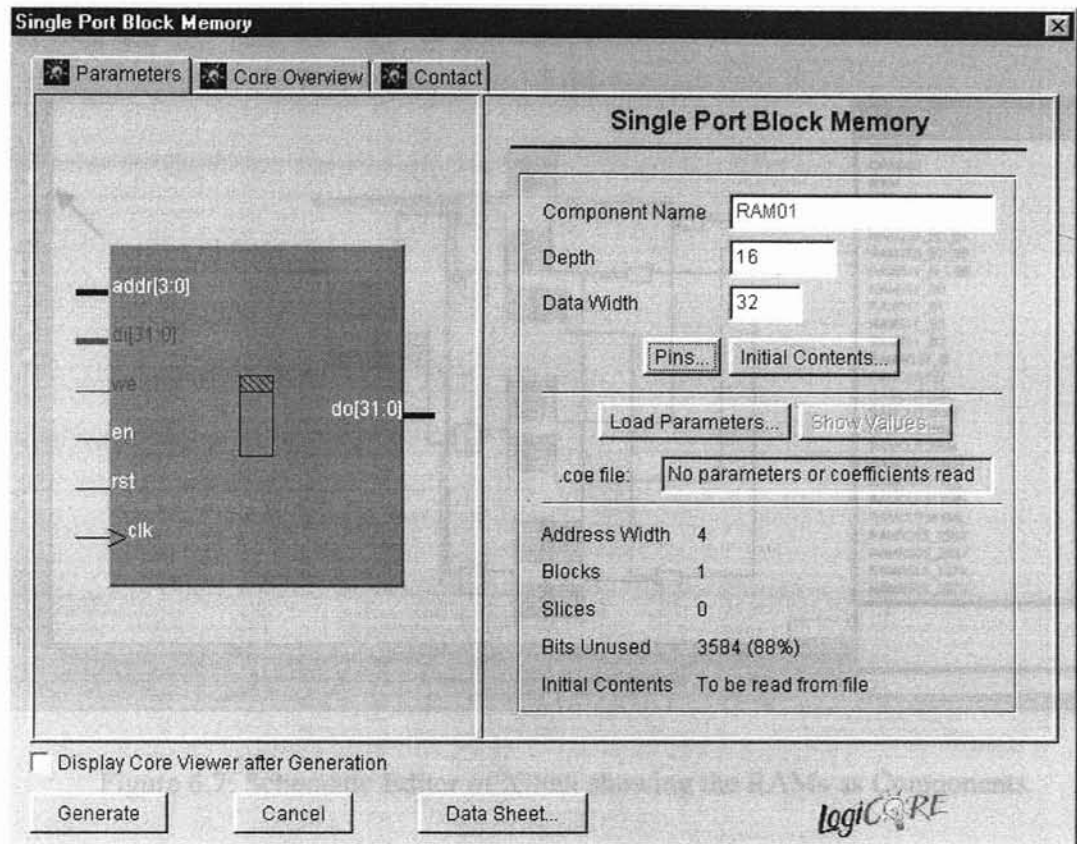


Figure 6.6: Final Appearance of the Single Port RAM Window

The final 'Single Block Memory' Window is shown in Figure 6.6. In this example the Component is named 'RAM01, it has a depth of '16' and hence '4' address lines, the data width is 32 bits. Figure 6.6 also shows that the initial contents are read from the MIF file. To generate the 'RAM01' component the 'Generate' button on the 'Single Block Memory' Window is pressed. This generates the 'RAM01' component and places it in the list of components in the Xilinx Project.

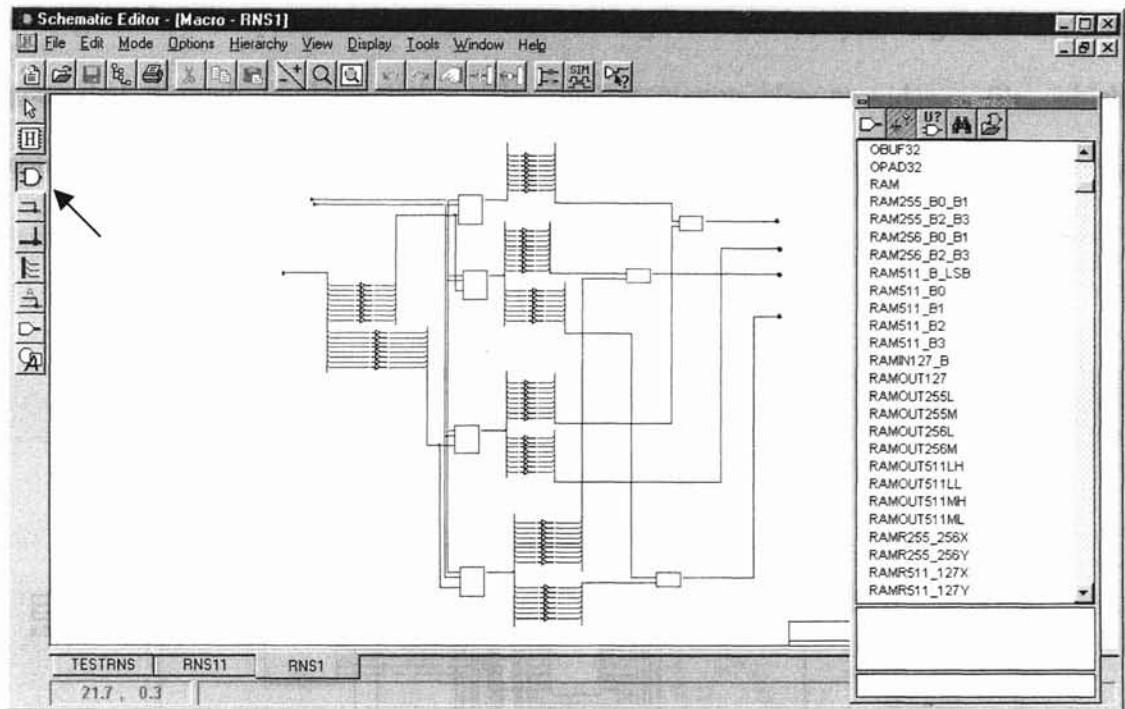


Figure 6.7: Schematic Editor of Xilinx showing the RAMs as Components.

The core generator is also used to generate the adder-delays required to implement the FIR filter in Residue Number System. Figure 6.7 shows the Schematic Editor of Xilinx. The List of components shown in Figure 6.7 contains the RAMs and the adder-delays that were generated using the Core Generator. These components are wired together appropriately in the Schematic Editor design of the RNS FIR filter.

6.2.3 Hardware Implementation

A top-level file combining the lower level files is created. From the top level file the filter is synthesized and implemented using the Xilinx Synthesizer built into the project manager software. This creates a file that contains the timing information of the implemented filter. This file is loaded into the simulator of the Xilinx project manager for

testing. Thus the implemented filter is tested through simulation, by testing the timing file generated through implementation. The implementation also provides a file, which contains the hardware that was required to implement the filter.

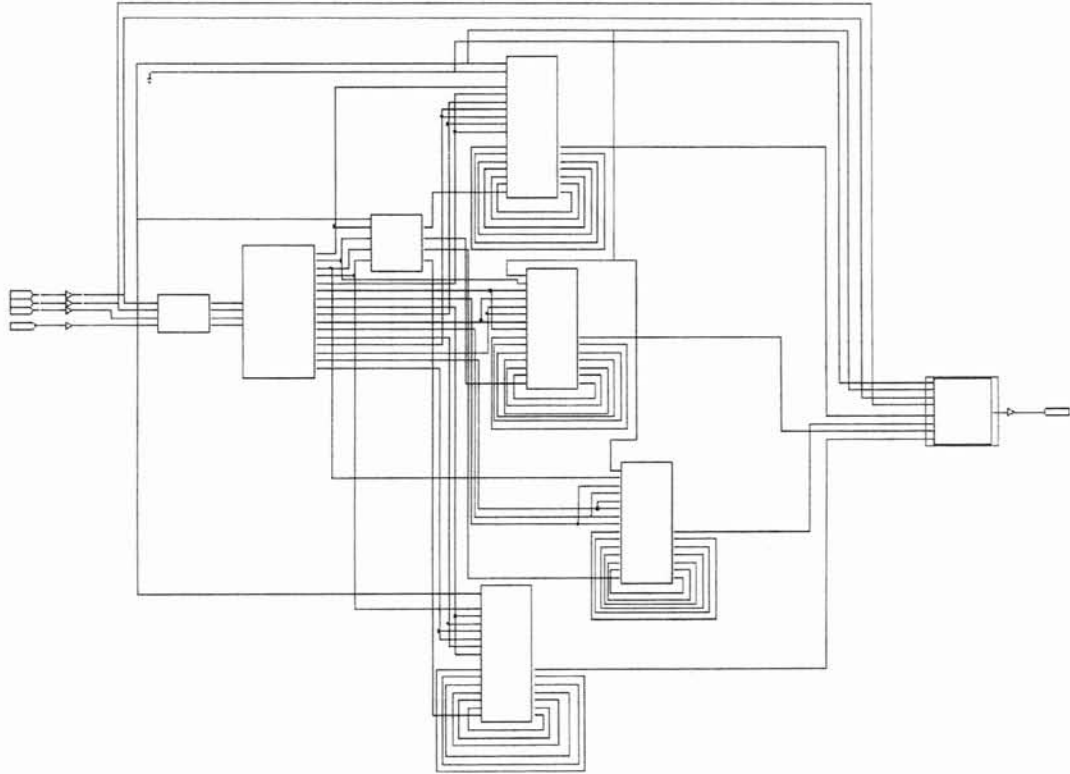


Figure 6.8: Xilinx Implementation of the Small Moduli RNS FIR Filter

6.3 Simulation and Implementation Results of Small Moduli RNS Filter

The top-level Schematic of the implementation of the Small Moduli Residue Number System FIR Filter discussed in the previous chapter (see Figure 5.3) is shown in Figure 6.8. The implementation [27] required 143 slices, 135 Flip-Flops, 191 LUT's and 7 block RAMs. Figure 6.9. shows a plot of the ideal filter (dotted line) and the actual

output of the Xilinx circuit of Figure 6.8. Based on the comparison of Figure 6.9, the actual output meets both pass-band and stop-band filter requirements. This demonstrates the use of the RNS technique implemented using block rams results in filters which can meet the filter specifications using fewer FPGA resources than non-RNS techniques.

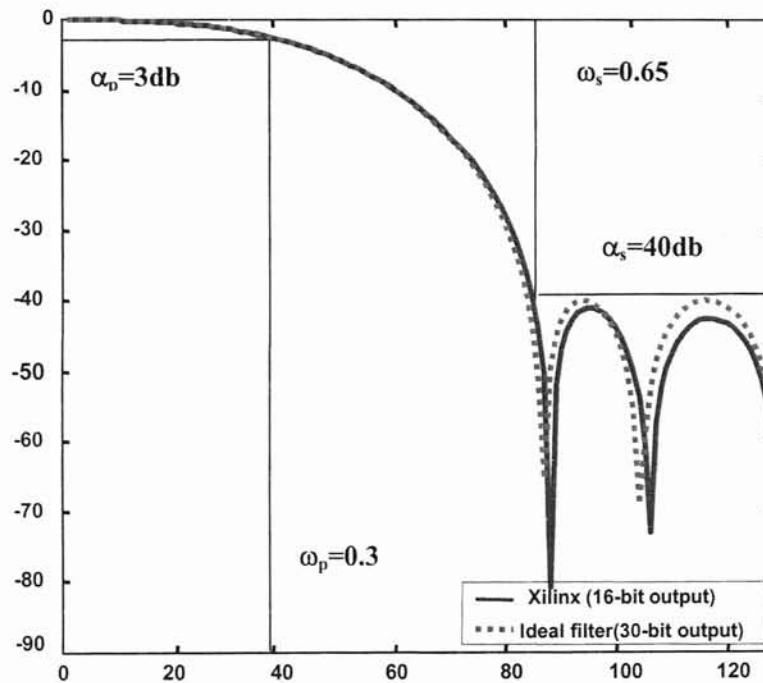


Figure 6.9: Comparison of ideal filter and actual output of Xilinx implementation.

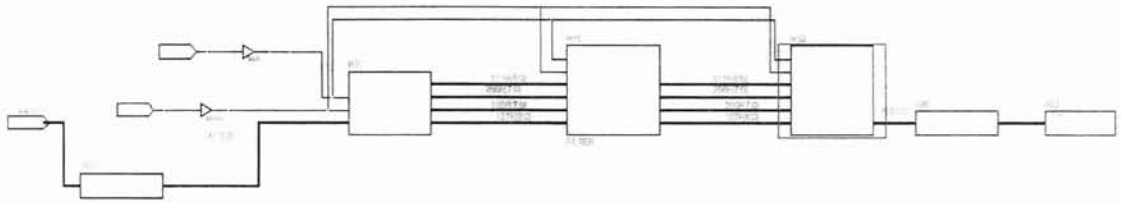


Figure 6.10: Xilinx Implementation of the Large Moduli RNS FIR Filter

6.4 Simulation and Implementation of Large Moduli RNS Filter

The top-level Schematic of the implementation of the Large Moduli Residue Number System FIR Filter discussed in the previous chapter (see Figures 5.4 and 5.9) is shown in Figure 6.10. The implementation [28] of this filter required 195 slices, 256 Flip-Flops, 344 LUT's and 23 block RAMs. Figure 6.11. Shows a plot of the ideal filter (dotted line) and the actual output of the Xilinx circuit of Figure 6.10.

From the comparison shown in Figure 6.11 we can say that the use of RNS techniques of implementing filters in block RAMs result in filters that meet the filter specification and use fewer FPGA resources than non-RNS techniques.

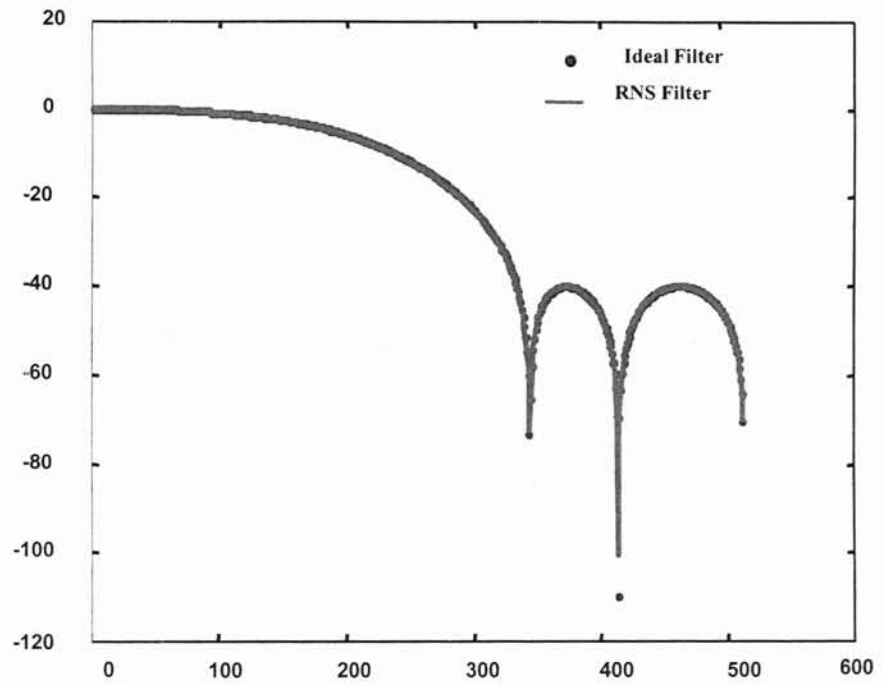


Figure 6.11: Comparison of ideal filter and actual output of Xilinx implementation.

Chapter 7

Conclusion and Future Work

7.1 Xilinx Implementation of the Example Filter

The example filter used for implementation in the thesis was originally implemented in the Xilinx Application Notes [26]. This provides a method of comparison between the RNS implementation and the Xilinx implementation. The authors of the application notes use the basic building blocks to build an 8-tap lowpass FIR filter with 16 bits input and 14 bits signed coefficients. The multipliers of FIR filter, are implemented such that the inputs to the multiplier are the tap data and the constant coefficient. These multipliers are called KCM since one of the inputs is a constant. The authors achieve efficient implementation of the KCM by storing the pre-computed partial products of the fixed coefficients. These partial products are stored in ROMs using distributed memory in Xilinx FPGAs. The 16-bit input is divided into 4-bits and they are inputs to 4 different ROMs. The output of the RAM is the partial product of the input and the coefficient. The output of the ROM with respect to its input is added to the partial product output from the other ROM. The KCM block diagram is shown in Figure 7.1.

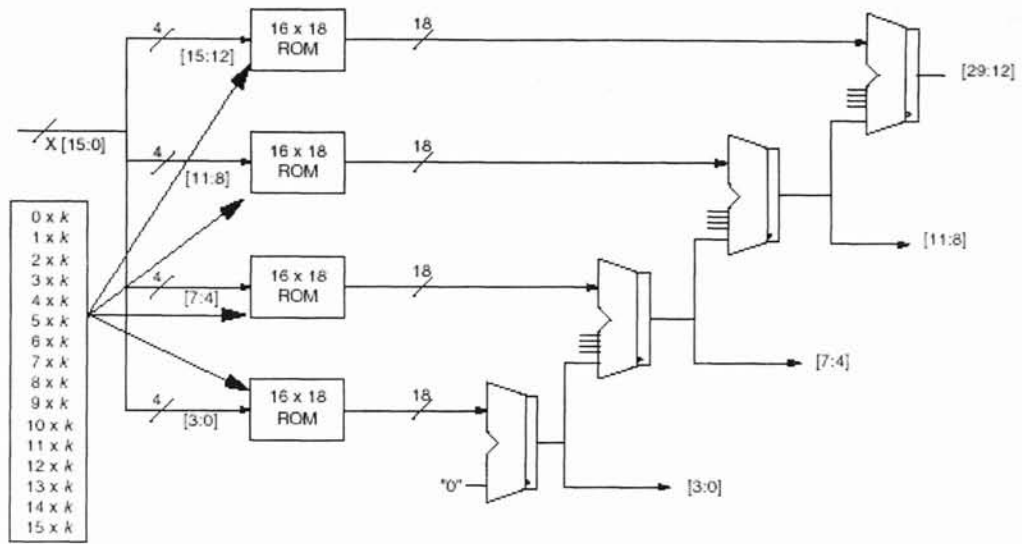


Figure 7.1: KCM Block Diagram [26].

7.2 Comparison of Xilinx and RNS Implementation

The Digital FIR filter was implemented using Residue Number System in hardware using the schematic layout approach. A VHDL code in collaboration with Xilinx CORE Generator tool can also be used to implement the Digital FIR.

The implementation of the simplified version of the filter in Small Moduli required 143 slices, 135 Flip-Flops, 191 LUT's and 7 block RAMs. The comparison between the Matlab simulation graph and the hardware implementation graph demonstrates that the use of the RNS technique implemented using block rams results in filters which can meet the filter specifications.

The complete filter in the example filter was implemented in the large moduli. Table 7.1 shows the results of this filter compared to an 8-tap FIR filter optimized in the Xilinx application notes.

Technique	Number of Slices	Number of LUT's	Number of FF's	Number of Block Rams
FPGA Express	645	977	807	0
Synplify Pro	584	931	755	0
RNS	195	344	256	23

Table 7.1: Comparisons of Various Optimization Tools.

7.3 Conclusion

From Table 7.1 it can be seen that the RNS version offers a considerable improvement over the optimized circuit discussed in the application note. In fact, in terms of slices, LUT's and FF's, the new RNS technique offers less than one - third of the hardware of the application note. However, this is obtained at the expense of 23 block RAMs of the 28 total on the Virtex 800 chip. But many implementations do not use the Xilinx Block RAMs but require the CLBs for their implementation, so by using the block RAMs and saving CLBs, larger and complex implementation can be done using the remaining CLBs. Hence we can conclude that only 1/3 of the hardware was required when compared to the implementation in the Xilinx Application notes. However since that conversion between binary and residue and back has to be done, RNS is slower than other methods of implementation, but in applications where the hardware savings is the

essential requirement than speed, RNS provides a solution. Recently, a method to convert IIR filters into FIR approximations using linear difference equations was proposed [29]. Hence Residue Number System can be used for IIR filters also, after they are converted into their FIR approximation.

7.4 Future Work

In this thesis a single example digital FIR filter was implemented in Residue Number System. Further Research can be done to compare of the speed, i.e. the clock rate, between the Residue Number System and conventional methods to obtain trade offs between hardware savings and speed can be done.

The work in this thesis can be extended to build a generic code Matlab code that generates the .mif files for any given file. Also a generic VHDL / Verilog code for the implementation of the RAMs that is accepted by the Core generator can be written. A script file can be written to combine the top-level files to generate the desired filter to meet the filter specification. These modifications can be incorporated into the Public domain-Matlab program for designing that was previous written by Leong [15] and into the generic VHDL code that was written by Howe [16].

Bibliography

- [1] A. V. Oppenheim and R. W. Schaffer, *Discrete Time Signal Processing*, Prentice Hall, 1989.
- [2] Lawrence R. Rabiner and Charles M. Rader, *Digital signal processing*, IEEE Press, 1972.
- [3] William D. Stanley, Gary R. Dougherty, Ray Dougherty, *Digital signal processing*, 2nd Ed: Reston Pub. Co, 1984.
- [4] Zhan Yu, Meng-Lin Yu, K. Azadet, A.N. Willson, "The use of reduced two's-complement representation in low-power DSP design," IEEE International Symposium on Circuits and Systems, vol. 1, pp. 77-80, 2002.
- [5] Zhan Yu, Meng-Lin Yu, K. Azadet, A.N. Willson, "A low power adaptive filter using dynamic reduced 2's-complement representation," in Proc. IEEE Custom Integrated Circuits Conf., pp. 141-144, 2002.
- [6] O. Salomon, J.M. Green, H. Klar, "General algorithms for a simplified addition of 2's complement numbers," IEEE Journal of Solid-State Circuits, vol. 30, pp. 839-844, July 1995.
- [7] R.M. Hewlitt, E.S. Swartzlantler Jr, "Canonical signed digit representation for FIR digital filters," IEEE Workshop on SiPS '00, pp. 416 -426, 2000.

- [8] Y.M. Hasan, L.J. Karam, M. Falkenburg, A. Helwig, M. Ronning, "*Canonical signed digit FIR filter design*," Thirty-Fourth Asilomar Conf. Signals Systems and Computers 2000, vol. 2, pp. 1653 -1656, 2000.
- [9] Dannielle L. Husinga, "*Design of Optimized Filters Using CSD Coefficient Representation*," MS Thesis, UC Davis, California, Mar. 1996.
- [10] Naren B. Balasubramanian, "*Optimal Design of Digital Filter using CSD Coefficients*," MS Thesis, Univ. of California, Davis, CA, 1997.
- [11] Andrew G. Dempster and Malcolm D. Macleod, "*Use of minimum-adder multiplier blocks in FIR digital filters*," Trans. IEEE Circuits and Systems, vol. 42, pp. 407-413, Oct. 1994.
- [12] Andrew G. Dempster and Malcolm D. Macleod, "*Constant integer multiplication using minimum adders*," Proc. IEE Circuits Devices Systems, vol. 141, pp. 569-576, Sept. 1995.
- [13] Andrew G. Dempster and Malcolm D. Macleod, "*Comparison of fixed-point FIR digital filter design techniques*," Trans. IEEE Circuits and Systems, vol. 44, pp. 591-593, July 1997.
- [14] Kah-HoweTan, Wen F. Leong, Sameer Kadam, M.A Soderstrand, L.G Johnson, "*Public-domain MATLAB program to generate highly optimized VHDL for FPGA implementation*," IEEE ISCAS '01, vol. 4, pp.514 –517, May 2001.
- [15] Wen F. Leong, "*Optimizing FIR filters coefficients using CSD representation and DM technique*," MS Thesis, Oklahoma State Univ., OK, May 2002.
- [16] Kah-Howe Tan, "*Optimizing FIR filters coefficients using CSD representation and DM technique*," MS Thesis, Oklahoma State Univ., OK, May 2002.

- [17] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien and F. A. Taylor, "*Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*," IEEE Press, 1988.
- [18] Nicholas S. Szabo and Richard I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*, McGraw-Hill.
- [19] P.V. Ananda Mohan, *Residue Number Systems: Algorithms and Architectures*, Kluwer Academic Publishers.
- [20] FPGA general information, <http://www.andraka.com/whatisan.htm>
- [21] John V. Oldfield; Richard C. Dorf, *Field-Programmable Gate Arrays: Configurable Logic for rapid prototyping and implementation of digital systems*, John Wiley & Sons Inc., 1995.
- [22] FPGA general information, <http://www.vcc.com/fpga.html>
- [23] Xilinx official web page, <http://xilinx.com>
- [24] Xilinx Virtex Application Notes,
<http://direct.xilinx.com/bvdocs/publications/ds003.pdf>.
- [25] Xilinx Virtex SelectBlock RAM Application Notes,
<http://www.xilinx.com/xapp/xapp130.pdf>
- [26] V. Pasham, A. Miller and K. Chapman, *Transposed Form FIR Filters*, Xilinx Application Note, XAP219 ver. 1.1, <http://www.xilinx.com/xapp/xapp219.pdf>.
- [27] Kadambari Kaluri, Wen F. Leong, K. H. Tan, L.G. Johnson, M.A. Soderstrand, *Comparison of RNS and optimized FIR digital filters in Xilinx FPGA's*," Proc. 44th IEEE MWSCAS '01, vol. 1, pp. 438 -441, 2001.

- [28] Kadambari Kaluri, Wen. F. Leong, K.H. Tan, L.G. Johnson, M.A.Soderstrand, "*FPGA hardware implementation of an RNS FIR digital filter,*" 35th Asilomar Conference on Signals, Systems and Computers, vol. 2, pp. 1340 -1344, 2001.
- [29] R.K Mallik , K.V.Rangarao, "*On the impulse response of a discrete-time linear IIR system*" accepted and to appear in Digital Signal Processing Journal, December 2002.

Appendix A

Matlab Files for the generation of .mif files for the RAMs

Binary.m

```
function [B] = binary(D,k1,k2)

% function [B] = binary(D,k1,k2)
%
% returns rounded binary quantization of the absolute value
% of a decimal quantity
%
% INPUTS:   D = decimal scalar input
%          k1 = bits to the left of decimal point (integer)
%          k2 = bits to the right of decimal point (fraction)
%
% OUTPUT:   B = binary vector output, MSB = B(1)
%

signD = sign(D);
D = abs(D);
D = D + 2^(-k2-1);
B = zeros(1,k1+k2);
for bit = (k1-1):(-1):(-k2),
    D_old = D;
    D = rem(D,2^(bit));
    if D_old - D > 0,
        B(-(bit-k1)) = 1;
    end
end

return;
```

Small moduli ramin.m

```
%To convert from Binary to RNS the input and multiply with the
%coefficients

b=[-0.0240156 0.0055081 0.1661459 0.3523616 0.3523616 0.1661459
0.0055081 -0.0240156];
br=round(182*182*b/max(b)/518);
m=[31 29 13 4];
for i=1:256
    for j=1:4
        r(i,j)=mod((i-1)*br(1),m(j));
    end
    raminb0_dec(i)=r(i,1)+32*(r(i,2)+32*(r(i,3)+16*r(i,4)));
    raminb0(i,:)=binary(raminb0_dec(i),16,0);
end
fid=fopen('raminb0.mif','w');
for i=1:256
    for j=1:16
        fprintf(fid,'%d',raminb0(i,j));
    end
    fprintf(fid,'\n');
end
fclose(fid);
%-----
for i=1:256
    for j=1:4
        r(i,j)=mod((i-1)*br(2),m(j));
    end
    raminb1_dec(i)=r(i,1)+32*(r(i,2)+32*(r(i,3)+16*r(i,4)));
    raminb1(i,:)=binary(raminb1_dec(i),16,0);
end
fid=fopen('raminb1.mif','w');
for i=1:256
    for j=1:16
        fprintf(fid,'%d',raminb1(i,j));
    end
    fprintf(fid,'\n');
end
fclose(fid);
%-----
for i=1:256
    for j=1:4
        r(i,j)=mod((i-1)*br(3),m(j));
    end
    raminb2_dec(i)=r(i,1)+32*(r(i,2)+32*(r(i,3)+16*r(i,4)));
    raminb2(i,:)=binary(raminb2_dec(i),16,0);
end
fid=fopen('raminb2.mif','w');
for i=1:256
    for j=1:16
```

```

        fprintf(fid, '%d', raminb2(i, j));
    end
    fprintf(fid, '\n');
end
fclose(fid);
%-----
for i=1:256
    for j=1:4
        r(i, j)=mod((i-1)*br(4), m(j));
    end
    raminb3_dec(i)=r(i, 1)+32*(r(i, 2)+32*(r(i, 3)+16*r(i, 4)));
    raminb3(i, :)=binary(raminb3_dec(i), 16, 0);
end
fid=fopen('raminb3.mif', 'w');
for i=1:256
    for j=1:16
        fprintf(fid, '%d', raminb3(i, j));
    end
    fprintf(fid, '\n');
end
fclose(fid);

```

Small moduli ramout crt.m

```

%To convert back to Binary from RNS using Chinese Remainder theorem

k4=3;
k13=5;
k29=12;
k31=14;
m=[4 13 29 31];
for j=1:128
    i=j-1;
    r4(j)=fix(i/2^5);
    r29(j)=i-2^5*r4(j);
    out(j)=k4*r4(j)/4+k29*r29(j)/29;
    ram1(j)=fix(2^16*(out(j)-fix(out(j))));
    ramout1(j, :)=binary(ram1(j), 16, 0);
end
fid=fopen('ramout1.mif', 'w');
for i=1:127
    for j=1:16
        fprintf(fid, '%d', ramout1(i, j));
    end
    fprintf(fid, '\n');
end
for j=1:16
    fprintf(fid, '%d', ramout1(128, j));
end

fclose(fid);

for j=1:512

```



```

    i=j-1;
    r13(j)=fix(i/2^5);
    r31(j)=i-2^5*r13(j);
    out(j)=k13*r13(j)/13+k31*r31(j)/31;
    ram2(j)=fix(2^8*(out(j)-fix(out(j))));
    ramout2(j,:)=binary(ram2(j),8,0);
    out(j)=2^8*out(j)-ram2(j);
    ram3(j)=fix(2^8*out(j));
    ramout3(j,:)=binary(ram3(j),8,0);
end
fid=fopen('ramout2.mif','w');
for i=1:511
    for j=1:8
        fprintf(fid,'%d',ramout2(i,j));
    end
    fprintf(fid,'\n');
end
for j=1:8
    fprintf(fid,'%d',ramout2(512,j));
end

fclose(fid);

fid=fopen('ramout3.mif','w');
for i=1:511
    for j=1:8
        fprintf(fid,'%d',ramout3(i,j));
    end
    fprintf(fid,'\n');
end
for j=1:8
    fprintf(fid,'%d',ramout3(512,j));
end

fclose(fid);

```

Large moduli rns lev1.m

```
%To convert from Binary to RNS the input
%From input, u to R
%u=256y+x
%*****
%For 511 and 127
clear
clear r1 r2
m1=[511,127];
for i=1:256,
    for j=1:2,
        r1(i,j)=mod((i-1)*256,m1(j));
        r2(i,j)=mod((i-1),m1(j));
    end
    R511_127y_dec(i)=r1(i,2)+128*r1(i,1);
    R511_127x_dec(i)=r2(i,2)+128*r2(i,1);
    R511_127y(i,:)=binary(R511_127y_dec(i),16,0);
    R511_127x(i,:)=binary(R511_127x_dec(i),16,0);
end

%Make it into Mif file
fid=fopen('ramR511_127y.mif','w');
for i=1:256,
    for j=1:16,
        fprintf(fid,'%d',R511_127y(i,j));
    end
    fprintf(fid,'\n');
end
fclose(fid);

fid=fopen('ramR511_127x.mif','w');
for i=1:256,
    for j=1:16,
        fprintf(fid,'%d',R511_127x(i,j));
    end
    fprintf(fid,'\n');
end
fclose(fid);
%*****

clear r1 r2
%for 255 and 256
m2=[255, 256];
for i=1:256,
    for j=1:2,
        r2(i,j)=mod((i-1),m2(j));
        r1(i,j)=mod((i-1)*256,m2(j));
    end

    R255_256y_dec(i)=r1(i,2)+256*r1(i,1);
    R255_256x_dec(i)=r2(i,2)+256*r2(i,1);
    R255_256y(i,:)=binary(R255_256y_dec(i),16,0);
```

```

R255_256x(i,:)=binary(R255_256x_dec(i),16,0);

end

%Make it into Mif file
fid=fopen('ramR255_256y.mif','w');
for i=1:256,
    for j=1:16,
        fprintf(fid,'%d',R255_256y(i,j));
    end
    fprintf(fid,'\n');
end
fclose(fid);

fid=fopen('ramR255_256x.mif','w');
for i=1:256,
    for j=1:16,
        fprintf(fid,'%d',R255_256x(i,j));
    end
    fprintf(fid,'\n');
end
fclose(fid);
%*****

```

Large moduli rns lev2.m

```

%To multiply with the coefficients of FIR filter
clear;
b=[-0.0240156 0.0055081 0.1661459 0.3523616 0.3523616 0.1661459
0.0055081 -0.0240156]; %Since is a 8 tap symmetry
br=round(64643.49*64643.49*b/max(b)/183457.8171968796);
m=[511 256 255 127];

```

```

%*****

```

```

%From R511
clear r
for i=1:512,
    for j=1:4,
        r(i,j)=mod((i-1)*br(j),511);
    end
    ramin511_b0(i,:)=binary(r(i,1),9,0);
    ramin511_b1(i,:)=binary(r(i,2),9,0);
    ramin511_b2(i,:)=binary(r(i,3),9,0);
    ramin511_b3(i,:)=binary(r(i,4),9,0);
end

```

```

%Write to mif file

```

```

fid=fopen('ram511_b_lsb.mif','w');

```

```

for i=1:512,
    fprintf(fid,'%d',ramin511_b0(i,9));
    fprintf(fid,'%d',ramin511_b1(i,9));
    fprintf(fid,'%d',ramin511_b2(i,9));
    fprintf(fid,'%d',ramin511_b3(i,9));
    fprintf(fid,'%d',0);
    fprintf(fid,'%d',0);
    fprintf(fid,'%d',0);
    fprintf(fid,'%d',0);
    fprintf(fid,'\n');
end
fclose(fid);

fid=fopen('ram511_b0.mif','w');
for i=1:512,
    for j=1:8,
        fprintf(fid,'%d',ramin511_b0(i,j));
    end
    fprintf(fid,'\n');
end
fclose(fid);

fid=fopen('ram511_b1.mif','w');
for i=1:512,
    for j=1:8,
        fprintf(fid,'%d',ramin511_b1(i,j));
    end
    fprintf(fid,'\n');
end
fclose(fid);

fid=fopen('ram511_b2.mif','w');
for i=1:512,
    for j=1:8,
        fprintf(fid,'%d',ramin511_b2(i,j));
    end
    fprintf(fid,'\n');
end
fclose(fid);

fid=fopen('ram511_b3.mif','w');
for i=1:512,
    for j=1:8,
        fprintf(fid,'%d',ramin511_b3(i,j));
    end
    fprintf(fid,'\n');
end
fclose(fid);

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

%From R256
clear r
for i=1:256,
    for j=1:4,

```

```

        r(i,j)=mod((i-1)*br(j),256);
    end

    ramin256_b0(i,:)=binary(r(i,1),8,0);
    ramin256_b1(i,:)=binary(r(i,2),8,0);
    ramin256_b2(i,:)=binary(r(i,3),8,0);
    ramin256_b3(i,:)=binary(r(i,4),8,0);
end

%Write to mif file
fid=fopen('ram256_b0_b1.mif','w');
for i=1:256,
    for j=1:8,
        fprintf(fid,'%d',ramin256_b0(i,j));
    end
    for j=1:8,
        fprintf(fid,'%d',ramin256_b1(i,j));
    end
    fprintf(fid,'\n');
end
fclose(fid);

%Write to mif file
fid=fopen('ram256_b2_b3.mif','w');
for i=1:256,
    for j=1:8,
        fprintf(fid,'%d',ramin256_b2(i,j));
    end
    for j=1:8,
        fprintf(fid,'%d',ramin256_b3(i,j));
    end
    fprintf(fid,'\n');
end
fclose(fid);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%From R255
clear r
for i=1:256,
    for j=1:4,
        r(i,j)=mod((i-1)*br(j),255);
    end
    ramin255_b0(i,:)=binary(r(i,1),8,0);
    ramin255_b1(i,:)=binary(r(i,2),8,0);
    ramin255_b2(i,:)=binary(r(i,3),8,0);
    ramin255_b3(i,:)=binary(r(i,4),8,0);
end

%Write to mif file
fid=fopen('ram255_b0_b1.mif','w');
for i=1:256,
    for j=1:8,
        fprintf(fid,'%d',ramin255_b0(i,j));
    end
    for j=1:8,
        fprintf(fid,'%d',ramin255_b1(i,j));

```

```

        end
        fprintf(fid, '\n');
    end
    fclose(fid);

%Write to mif file
fid=fopen('ram255_b2_b3.mif', 'w');
for i=1:256,
    for j=1:8,
        fprintf(fid, '%d', ramin255_b2(i,j));
    end
    for j=1:8,
        fprintf(fid, '%d', ramin255_b3(i,j));
    end
    fprintf(fid, '\n');
end
fclose(fid);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%From R127
clear r
for i=1:127,
    for j=1:4,
        r(i,j)=mod((i-1)*br(j), 127);
    end
    ramin127_b0(i,:)=binary(r(i,1), 7, 0);
    ramin127_b1(i,:)=binary(r(i,2), 7, 0);
    ramin127_b2(i,:)=binary(r(i,3), 7, 0);
    ramin127_b3(i,:)=binary(r(i,4), 7, 0);
end

%Write to mif file
fid=fopen('ramin127_b.mif', 'w');
for i=1:127,
    for j=1:7,
        fprintf(fid, '%d', ramin127_b0(i,j));
    end
    for j=1:7,
        fprintf(fid, '%d', ramin127_b1(i,j));
    end
    for j=1:7,
        fprintf(fid, '%d', ramin127_b2(i,j));
    end
    for j=1:7,
        fprintf(fid, '%d', ramin127_b3(i,j));
    end
    fprintf(fid, '%d', 0);
    fprintf(fid, '%d', 0);
    fprintf(fid, '%d', 0);
    fprintf(fid, '%d', 0);
    fprintf(fid, '\n');
end
fclose(fid);

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

```

Large moduli rns bin.m

```
%Using CRT to convert RNS to Binary
clear

k511=346;
k256=127;
k255=253;
k127=106;

m=[511 256 255 127];

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

%For mod511
clear i j out k
for j=1:512,
    out(j)=k511*(j-1)/m(1);          *** This is the CRT
    to convert back to binary ***
    ram511(j)=fix(2^32*(out(j)-fix(out(j)))); *** This is to
    obtain the fractional part ***
    ramout511(j,:)=binary(ram511(j),32,0); *** This is to
    convert from decimal to binary ***
end
fid=fopen('ramout511mh.mif','w');
for i=1:512,
    for k=1:8,
        fprintf(fid,'%d',ramout511(i,k));
    end
    fprintf(fid,'\n');
end
fclose(fid);

fid=fopen('ramout511ml.mif','w');
for i=1:512,
    for k=9:16,
        fprintf(fid,'%d',ramout511(i,k));
    end
    fprintf(fid,'\n');
end
fclose(fid);

fid=fopen('ramout511Lh.mif','w');
for i=1:512,
    for k=17:24,
        fprintf(fid,'%d',ramout511(i,k));
    end
    fprintf(fid,'\n');
end
fclose(fid);
```

```

fid=fopen('ramout511L1.mif','w');
for i=1:512,
    for k=25:32,
        fprintf(fid,'%d',ramout511(i,k));
    end
    fprintf(fid,'\n');
end
fclose(fid);

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```

%For mod256
clear i j out k
for j=1:256,
    out(j)=k256*(j-1)/m(2);
    ram256(j)=fix(2^32*(out(j)-fix(out(j))));
    ramout256(j,:)=binary(ram256(j),32,0);
end

```

```

fid=fopen('ramout256m.mif','w');
for i=1:256,
    for k=1:16,
        fprintf(fid,'%d',ramout256(i,k));
    end
    fprintf(fid,'\n');
end
fclose(fid);

```

```

fid=fopen('ramout256l.mif','w');
for i=1:256,
    for k=17:32,
        fprintf(fid,'%d',ramout256(i,k));
    end
    fprintf(fid,'\n');
end
fclose(fid);

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```

%For mod255
clear i j out k
for j=1:256,
    out(j)=k255*(j-1)/m(3);
    ram255(j)=fix(2^32*(out(j)-fix(out(j))));
    ramout255(j,:)=binary(ram255(j),32,0);
end

```

```

fid=fopen('ramout255m.mif','w');
for i=1:256,
    for k=1:16,
        fprintf(fid,'%d',ramout255(i,k));
    end
    fprintf(fid,'\n');
end

```



```

end
fclose(fid);

fid=fopen('ramout255l.mif','w');
for i=1:256,
    for k=17:32,
        fprintf(fid,'%d',ramout255(i,k));
    end
    fprintf(fid,'\n');
end
fclose(fid);

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```

%For mod127

```

```

clear i j out k
for j=1:128,
    out(j)=k127*(j-1)/m(4);
    ram127(j)=fix(2^32*(out(j)-fix(out(j))));
    ramout127(j,:)=binary(ram127(j),32,0);
end

```

```

fid=fopen('ramout127.mif','w');
for i=1:128,
    for k=1:32,
        fprintf(fid,'%d',ramout127(i,k));
    end
    fprintf(fid,'\n');
end
fclose(fid);

```

VITA 2

Kadambari Kaluri

Candidate for the Degree of

Master of Science

Thesis: **HARDWARE IMPLEMENTATION OF DIGITAL FIR FILTER IN
RESIDUE NUMBER SYSTEM**

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Srikakulam, India on August 8th, 1978, the daughter of Ranga Rao and Lavanya.

Education: Received a Bachelor of Engineering in Electronics and Communication Engineering from University of Madras in May 2000. Completed the requirements for the Master of Science degree with a major in Electrical Engineering at Oklahoma State University in December, 2002.

Experience:

Worked as a Research and Teaching Assistant for the Electrical and Computer Science Engineering Department for Oklahoma State University from August 2000 to present.