2002

Efficient convolvers using the Polynomial Residue Number System technique

Surendar Paruchuri
Louisiana State University and Agricultural and Mechanical College

EFFICIENT CONVOLVERS USING THE POLYNOMIAL RESIDUE NUMBER SYSTEM TECHNIQUE

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

in

The Department of Electrical and Computer Engineering

By
Surendar Paruchuri
B.E. E.C.E. .,1999
University of Madras
May, 2002

Acknowledgements

At the outset, I wish to thank my parents and sister for their abounding love, encouragement and support they have bestowed upon me.

I wish to express my sincere gratitude to my advisor Dr. Alexander Skavantzos for his tremendous guidance, support, and patience throughout my thesis work.

I wish to thank the faculty members Dr. Guoxiang Gu and
Dr. Jaganathan Ramanujam who accepted to guide me and be part of my committee.

I wish to thank Mr. Steven Nguyen for offering me a graduate assistantship position, without which it would not have been possible for me to be financially independent during my graduate studies at LSU.

Finally, I wish to thank all my friends for their support and help in times of need.

Table of Contents

List of Tables

List of Figures

Abstract

The problem of computing linear convolution is a very important one because with linear convolution we can mechanize digital filtering.

The linear convolution of two N-point sequences can be computed by the cyclic convolution of the following 2N-point sequences. The original sequence padded with N zero's each. The cyclic convolution of two N-point sequences requires N^2 multiplications and $N(N-1)$ additions for its computation.

A very efficient way of computing cyclic convolution of two sequences is by using the Polynomial Residue Number System (PRNS) technique. Using this technique the cyclic convolution of two N-point sequences can be computed using only N multiplications instead of N^2 multiplications. This can be achieved based on some forward and inverse PRNS transformation mappings. These mappings rely on additions, subtractions and many scaling operations (multiplications by constants). The PRNS technique would lose a lot in value if these many scaling operations were difficultly implemented. In this thesis we will show how to calculate cyclic convolution of two sequences using the PRNS technique based on forward and inverse transformation mapping which rely on complement operations (negations), additions and rotation operations. These rotation operations do not require any computational hardware. Therefore the complicated hardware required for the scaling operations has now been substituted by rotators, which do not require any computational hardware.

Chapter 1

Introduction

In this era of digital communications and information technology the area of Digital Signal Processing (DSP) assumes a very prominent place. The problem of computing linear convolution within this area is very important one because, we can mechanize digital filtering using linear convolution.

Consider two N-point sequences

$A = (a_0, a_1, a_2, ..., a_{N-1})$ and $B = (b_0, b_1, b_2, ..., b_{N-1})$.

Their linear convolution is $A(x)B(x)$ where

$A(x) = \sum_{i=0}^{N-1} a_i x^i$ and $B(x) = \sum_{i=0}^{N-1} b_i x^i$.

The above linear convolution can be computed by the cyclic convolution of the two 2N-point sequences $A' = (a_0, a_1, a_2, ..., a_{N-1}, 0,0,0,...,0)$ and $B' = (b_0, b_1, b_2, ..., b_{N-1}, 0,0,0,...,0)$.

The cyclic convolution of A' and B' is

$< A'(x)B'(x) >_{x^{2N}-1}$

Where $< P(x) >_{Q(x)}$ denotes $P(x) \bmod Q(x)$ and where

$A'(x) = \sum_{i=0}^{N-1} a_i x^i + \sum_{i=N}^{2N-1} o.x^i$ and

$B'(x) = \sum_{i=0}^{N-1} b_i x^i + \sum_{i=N}^{2N-1} o.x^i$.

Then $A(x)B(x) =< A'(x)B'(x) >_{x^{2N}-1}$

A very efficient way for computing the cyclic convolution is by using the Discrete Fourier Transform (DFT).

The Residue Number System (RNS)[1] has been gaining unparalleled importance in the area of Digital Signal Processing (DSP). This integer-based number system offers parallel, carry-free, high-speed arithmetic. Unlike the traditional system this integer-based number system also offers many properties like the error detection, error correction and fault tolerance, which makes it very attractive to be implemented in digital systems.

As based on the above-mentioned properties and features of RNS, this system is implemented in various areas of Digital Signal Processing like digital filters, convolution, correlation, DFT, FFT computations [2]-[13], and direct digital frequency synthesis [14]

The continuous and enduring work in the area of the RNS has led to the development of various other systems based on RNS like the Quadratic Residue Number System (QRNS) [15]-[16], [7], [17]-[18], the Quadratic Like Residue Number System (QLRNS) [19], the Modified Quadratic Residue Number System (MQRNS) [20], the Polynomial Residue Number System (PRNS) [21]-[22], [9]-[10], the Multi-Polynomial-Channel PRNS (MPCPRNS)[23] and the two-level binary RNS with pairs of conjugate moduli [24]. All these systems are being employed in the area of DSP to reduce the computational complexity while at the same time trying to achieve maximum parallelism.

A very efficient way of computing cyclic convolution of two sequences is by using the Polynomial Residue Number System (PRNS) technique. Using this technique the cyclic convolution of two N-point sequences can be computed using only N multiplications instead of N^2 multiplications. This can be achieved based on some forward and inverse PRNS transformation mappings. These mappings rely on additions,

subtractions and many scaling operations (multiplications by constants). The PRNS technique would lose a lot in value if these many scaling operations were difficultly implemented. In this thesis we will show how to calculate cyclic convolution of two sequences using the PRNS technique based on forward and inverse transformation mapping which rely on complement operations (negations), additions and rotation operations. These rotation operations do not require any computational hardware. Therefore the complicated hardware required for the scaling operations has now been substituted by rotators, which do not require any computational hardware.

1.1 Organization of the Thesis

Chapter 2 provides the basics of Residue Number System (RNS). Chapter 3 introduces the Polynomial Residue Number System (PRNS). Chapter 4, the main chapter of the thesis, presents the efficient cyclic convolves which are implemented using the PRNS technique which relies on forward and inverse transformation relying on complement operations, additions and rotations which do not require any computational hardware. This chapter also provides the background in diminished number system, which is extensively used in the forward and inverse transformations. Finally chapter 5 provides a conclusion.

Chapter 2

Residue Number System

In this chapter, the theoretical background for RNS[1] is presented. We will also discuss the representation, processing, conversion and RNS arithmetic in this chapter.

2.1 RNS Definition and Representation

The Residue Number System[1] is an integer system capable of supporting parallel, carry-free, high speed arithmetic.

Mathematically it can be defined by a set $S = \{m_1, m_2, m_3, ..., m_N\}$. The set S is called the set of moduli. The moduli $m_1, m_2, m_3, ..., m_N$ are positive integers such that $(m_i, m_j) = 1$ (where (a,b) indicates Greatest Common Divisor (GCD) of a and b) for $i \neq j$ (i.e. the moduli are pairwise relatively prime or relatively prime in pairs).

Let M be $M = \prod_{i=1}^{N} m_i = m_1.m_2.m_3.....m_N$ (2.1)

If X ($X \in Z_m$ where $Z_m = \{0,1,2,..., M-1\}$) is a number to be represented in RNS, then the N-tuple $(x_1, x_2, x_3, ..., x_N)$ is the representation of X in this system. This N-tuple consists of the integer remainders that result from dividing X by each one of the N moduli.

$x_i = X \bmod m_i$ (2.2)

2.1.1 Weighted to RNS Conversion

In order to convert the number X from the weighted system into the RNS, we divide X by each of the RNS moduli and obtain the remainders. The set of remainders represent the number X in the RNS.

$$x_i = < X >_{m_i} \quad \text{if} \quad X > 0 \tag{2.3}$$

$$x_i = < M - |X| >_{m_i} \quad \text{if} \quad X < 0 \tag{2.4}$$

where $< a >_b$ indicates a mod b

2.1.2 Dynamic Range of the RNS

Let M be $M = \prod_{i=1}^{N} m_i$, and then if the system supports only unsigned numbers, the dynamic range (DR) is,

$$DR = [0 \quad M-1]. \tag{2.5}$$

If the system supports signed numbers the dynamic range (DR) is

$$DR = [-M/2 \quad M/2 - 1] \text{ if } M \text{ is even} \tag{2.6}$$

$$DR = [-(M-1)/2 \quad (M-1)/2] \text{ if } M \text{ is odd} \tag{2.7}$$

2.1.3 RNS to Weighted Transformation

There are two basic techniques for converting from RNS to the weighted system.

- The Chinese remainder Theorem (CRT)
- The Mixed Radix Conversion (MRC)

2.1.3.1 Chinese Remainder Theorem

Let $X \in [0, M]$ be represented as

$$X \xrightarrow{RNS} (x_1, x_2, x_3, ..., x_N) \tag{2.8}$$

The CRT reconstruction of X from its residues is as follows

$$X =< X_1 M_1 L_1 + X_2 M_2 L_2 + ... + X_N M_N L_N >_M$$

or

$$X =< \sum_{i=1}^{N} X_i M_i L_i >_M \tag{2.9}$$

where $M = \prod_{i=1}^{N} m_i$; $M_i = \frac{M}{m_i}$; $L_i =< M_i^{-1} > m_i$

An alternate form of the CRT equation is

$$X =<< X_1 L_1 >_{m_1} M_1 + < X_2 L_2 >_{m_2} M_2 + ... + < X_N L_N >_{m_n} M_N >_M$$

or

$$X =< \sum_{i=1}^{N} < X_i L_i >_{m_i} M_i >_M \tag{2.10}$$

where $M = \prod_{i=1}^{N} m_i$; $M_i = \frac{M}{m_i}$; $L_i =< M_i^{-1} > m_i$

2.1.3.2 Mixed Radix Conversion Technique

The CRT conversion requires arithmetic mod M in obtaining the weighted result. This is an extra restriction and requires special hardware to perform the arithmetic operations mod M. The Mixed Radix Conversion (MRC), on the other hand, does not have this disadvantage. It also permits simpler magnitude comparison.

Let $X \in [0, M]$ be represented as

$$X \overset{RNS}{\rightarrow} (x_1, x_2, x_3, ..., x_N)$$

The Mixed Radix Conversion formula is the one shown below

$$X = X'_1 + m_1 . X'_2 + m_1 . m_2 . X'_3 + ... + m_1 . m_2 . m_3 .... m_{N-1} . X'_N \tag{2.11}$$

where $X'_1, X'_2, X'_3, ..., X'_N$ are called the mixed radix digits and $X'_i \in Z_{m_i}$.

The mixed radix digits can be expressed as functions of the residues. The mixed radix digits for a 4-moduli RNS are given by:

$$X'_1 = x_1$$

$$X'_2 = < m_1^{-1}(x_2 - x_1) >_{m_2} \quad (2.12)$$

$$X'_3 = < m_1^{-1} m_2^{-1}(x_3 - m_1 X'_2 - X'_1) >_{m_3}$$

$$X'_4 = < m_1^{-1} m_2^{-1} m_3^{-1}(x_4 - m_1 m_2 X'_3 - m_1 X'_2 - X'_1) >_{m_4}$$

2.2 Arithmetic Operations in RNS

2.2.1 Addition

The addition operation between two numbers X and Y in the RNS as defined by the moduli set S, is performed as follows.

Let

$$X \overset{RNS}{\rightarrow} (x_1, x_2, x_3, ..., x_N)$$

and

$$Y \overset{RNS}{\rightarrow} (y_1, y_2, y_3, ..., y_N)$$

The following equations describe the inputs X,Y in the N-channels

$$X = q_1 m_1 + x_1 \quad , \quad Y = q'_1 m_1 + y_1$$

$$X = q_2 m_2 + x_2 \quad , \quad Y = q'_2 m_2 + y_2$$

…….. (2.13)

$$X = q_N m_N + x_N \quad , \quad Y = q'_N m_N + y_N$$

The result of the addition is then

$$Z \overset{RNS}{\rightarrow} (z_1, z_2, z_3, ..., z_N)$$

and the components of the result in the RNS domain are given by $z_i =< x_i + y_i > m_i$,

$1 \le i \le N$.

The following equation describes the result Z in the N-channels

$$Z = (q_1 + q'_1)m_1 + (x_1 + y_1)$$

$$Z = (q_2 + q'_2)m_2 + (x_2 + y_2)$$

……. (2.14)

$$Z = (q_N + q'_N)m_N + (x_N + y_N)$$

2.2.2 Subtraction

In the case of performing subtraction, let

$$X \overset{RNS}{\rightarrow} (x_1, x_2, x_3, ..., x_N)$$

and

$$Y \overset{RNS}{\rightarrow} (y_1, y_2, y_3, ..., y_N)$$

The following equations describe the inputs *X,Y* in the N-channels

$$X = q_1 m_1 + x_1 \quad , \quad Y = q'_1 m_1 + y_1$$

$$X = q_2 m_2 + x_2 \quad , \quad Y = q'_2 m_2 + y_2$$

…….. (2.15)

$$X = q_N m_N + x_N \quad , \quad Y = q'_N m_N + y_N$$

The result Z is given by

$$Z \xrightarrow{RNS} (z_1, z_2, z_3, ..., z_N)$$

and the components of the result in the RNS domain are given by $z_i =< x_i - y_i >_{m_i}$ or

$z_i =< x_i + (-y_i) >_{m_i}$, $1 \le i \le N$.

2.2.3 Multiplication

The multiplication operation in the RNS is analogous to the addition and subtraction operations. The result $Z=X$ x Y can be expressed as

$$Z = q_i m_i q'_i \, m_i + q_i m_i y_i + q'_i \, m_i x_i + (x_i y_i) \qquad (2.16)$$

This is equivalent to $< Z >_{m_i} = z_i =< x_i X y_i >_{m_i}$, $1 \le i \le N$

2.2.4 Division

The division operation is not carried out directly in the RNS computations. This is because RNS is defined over integers while the division operation in not closed over integers.

2.2.5 Multiplicative Inverse

The multiplicative inverse of a number X with respect to the modulus is the number X^{-1} such that

$$< X^{-1} X >_m = 1 \qquad (2.17)$$

2.2.6 Fermat's Theorem

The fermat's theorem states that for a given prime number k $< a^k >_k =< a >_k$. (2.18)

If $a \neq 0$ which implies that inverse of a is defined and is finite then multiplying both sides of 2.18 by a^{-1} two times yields the following result.

$$< a^{-1} >_k =< a^{k-2} >_k \qquad (2.19)$$

Chapter 3

The Polynomial Residue Number System (PRNS)

The PRNS[21] provides an efficient and faster implementation of multiplication of two polynomials of degree $(N-1) \bmod (x^N \pm 1)$ over a modular ring Z_m. The multiplication is carried out in *N* parallel multiplications involving no additions instead of N^2 multiplications and *N(N-1)* additions.

Consider *P(m)* to be a finite structure having *(N-1)* order polynomials with coefficients in Z_m. A PRNS(N) isomorphic mapping f_N of *P(m)* onto $Z_m^N \stackrel{\Delta}{=} Z_m x Z_m x ... x Z_m$ exists if the polynomial $x^N \pm 1$ is factorizable into *N* distinct first degree factors in Z_m.

We now discuss the PRNS forward mapping which is used to transform coefficients of a polynomial into the PRNS domain.

3.1 Forward Mapping

$$f_N : A(x) = a_0 + a_1 x + ... + a_{N-1} x^{N-1} \xrightarrow{f_N} A^*(x) \qquad (3.1)$$

$$= (a_0^*, a_1^*, ..., a_{N-1}^*)$$

with $a_i^* = << A(x) >_{(x-r_i)} >_m = < A(r_i) >_m, i = 0,1,..., N-1$

where r_i are distinct roots of $x^N \pm 1 \equiv 0$ in Z_m.

$x^N \pm 1 \equiv (x - r_0)(x - r_1)...(x - r_{N-1}), r_i \in Z_m, i = 0,1,2,..., N-1$

We now discuss the PRNS inverse mapping which is used to transform the values in PRNS domain into the native form.

3.2 Inverse Mapping

$$f_N^{-1} : A^*(x) = (a_0^*, a_1^*, ..., a_{N-1}^*) \xrightarrow{f_N^{-1}} A(x) = a_0 + a_1 x + ... + a_{N-1} x^{N-1} \quad (3.2)$$

where $A(x) = \sum_{i=0}^{N-1} a_i^* Q_i(x)$

and $Q_i(x) = N^{-1}(1 + r_i^{-1} x + r_i^{-2} x^2 + ... + r_i^{-(N-2)} x^{N-2} + r_i^{-(N-1)} x^{N-1})$

The coefficients a_i are given by

$$a_i \equiv < N^{-1}(a_0^* r_0^{-i} + a_1^* r_1^{-i} + ... + a_{N-1}^* r_{N-1}^{-i}) >_m ; i = 0,1,2,..., N-1$$

In the PRNS domain the rules of addition remain unaffected but the multiplication is carried out as shown.

$$(a_0^*, a_1^*, a_2^*, ..., a_{N-1}^*)(b_0^*, b_1^*, b_2^*, ..., b_{N-1}^*)$$
$$= (< (a_0^* b_0^*) >_m, < (a_1^* b_1^*) >_m, < (a_2^* b_2^*) >_m, ..., < (a_{N-1}^* b_{N-1}^*) >_m)$$

(3.3)

The equation mentioned above demonstrates that product of two polynomials of *(N-1)* degree requires *N* multiplication mod *m* performed in parallel and there are no addition operations performed, if polynomial product is performed in Z_m^N. The same polynomial product $< A(x)B(x) >_{x^N \pm 1}$ would require N^2 multiplications and *N(N-1)* additions mod *m*, if performed in *P(m).*

The polynomial product is obtained in Z_m^N is in one level as opposed to multiple levels required in *P(m).*

Lemma 3.1:

$x^N + 1$ can be factorized into N distinct factors in Z_m as

$x^N + 1 \equiv < (x - r_0)(x - r_1)...(x - r_{N-1}) >_m$ if and only if $N \mid (p_i - 1)/2, i = 1,2,...,L$ where $a \mid b$ represents "a divides b"; where N and m are positive integers with a prime decomposition on m given in terms of powers e_i of its prime factors p_i, as

$m = p_1^{e_1} p_2^{e_2} p_3^{e_3} ... p_L^{e_L}$ with $N < p_i$.

Similarly for $x^N - 1$, the necessary and sufficient condition for its factorization becomes $N \mid (p_i - 1), i = 1,2,...,L$. For both cases the factorization is not unique. There are $(N!)^{L-1}$ different ways to factorize $x^N \pm 1$ into N distinct first-degree factors.

Theorem 3.1:

If the polynomial $x^N \pm 1$ can be factorized in N distinct factors in Z_m, then the mapping f_N of *P(m)* onto Z_m^N satisfies the following :

i) f_N is one –to-one and onto

ii) for $A, B \in P(m)$

$$f_N(A + B) = f_N(A) + f_N(B) \tag{3.4}$$

$$f_N(A.B) = f_N(A).f_N(B) \tag{3.5}$$

Example to illustrate $< A(x)B(x) >_{x_4+1}$ **in** Z_m

Let $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ and

$B(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3$

Here $A(x) = 2 + 3x + 4x^2 + 5x^3$

$B(x) = 6 + 7x + 8x^2 + 9x^3$

Let *m=17*

Roots of $x^4+1=<0>_{17}$ are $r_0=2; r_1=15; r_2=9; r_3=8$

$A(x) \xrightarrow{PRNS} (a_0^*, a_1^*, a_2^*, a_3^*)$

$a_0^* =< A(r_0) >_{17} =< 2+3*2+4*2^2+5*2^3 >_{17} = 13$

$a_1^* =< A(r_1) >_{17} =< 2+3*15+4*15^2+5*15^3 >_{17} = 6$

$a_2^* =< A(r_2) >_{17} =< 2+3*9+4*9^2+5*9^3 >_{17} = 3$

$a_3^* =< A(r_3) >_{17} =< 2+3*8+4*8^2+5*8^3 >_{17} = 3$

Therefore $A(x) \xrightarrow{PRNS} (a_0^*, a_1^*, a_2^*, a_3^*) = (13,6,3,3)$

$B(x) \xrightarrow{PRNS} (b_0^*, b_1^*, b_2^*, b_3^*)$

$b_0^* =< B(r_0) >_{17} =< 6+7*2+8*2^2+9*2^3 >_{17} = 5$

$b_1^* =< B(r_1) >_{17} =< 6+7*15+8*15^2+9*15^3 >_{17} = 3$

$b_2^* =< B(r_2) >_{17} =< 6+7*9+8*9^2+9*9^3 >_{17} = 2$

$b_3^* =< B(r_3) >_{17} =< 6+7*8+8*8^2+9*8^3 >_{17} = 14$

Therefore $B(x) \xrightarrow{PRNS} (b_0^*, b_1^*, b_2^*, b_3^*) = (5,3,2,14)$

$C(x) =<< A(x)B(x) >_{x^4+1} >_{17} \xrightarrow{PRNS} (c_0^*, c_1^*, c_2^*, c_3^*)$

$c_0^* =< a_0^*.b_0^* >_{17} =< 13*5 >_{17} = 14$

$c_1^* =< a_1^*.b_1^* >_{17} =< 6*3 >_{17} = 1$

$c_2^* =< a_2^*.b_2^* >_{17} =< 3*2 >_{17} = 6$

$c_3^* =< a_3^*.b_3^* >_{17} =< 3*14 >_{17} = 8$

Therefore $C(x) \xrightarrow{PRNS} (c_0^*, c_1^*, c_2^*, c_3^*) = (14,1,6,8)$

Inverse transformation to get (c_0, c_1, c_2, c_3)

The inverse roots are given by

$r_0^{-1} = 2^{-1} = 9$

Verification:

$< 2 * 9 >_{17} = < 18 >_{17} = 1$

$r_1^{-1} = 15^{-1} = 8$

Verification:

$< 15 * 8 >_{17} = < 120 >_{17} = 1$

$r_2^{-1} = 9^{-1} = 2$

Verification:

$< 9 * 2 >_{17} = < 18 >_{17} = 1$

$r_3^{-1} = 8^{-1} = 15$

Verification:

$< 8 * 15 >_{17} = < 120 >_{17} = 1$

The value of 4^{-1} is 13

Verification:

$< 4 * 13 >_{17} = < 52 >_{17} = 1$

$c_0 = < 4^{-1}(c_0^* r_0^{-0} + c_1^* r_1^{-0} + c_2^* r_2^{-0} + c_3^* r_3^{-0}) >_{17}$

$= < 13(14 + 1 + 6 + 8) >_{17} = 3$

$c_1 = < 4^{-1}(c_0^* r_0^{-1} + c_1^* r_1^{-1} + c_2^* r_2^{-1} + c_3^* r_3^{-1}) >_{17}$

$= < 13(14 * 9 + 1 * 8 + 6 * 2 + 8 * 15) >_{17} = 7$

$c_2 = < 4^{-1}(c_0^* r_0^{-2} + c_1^* r_1^{-2} + c_2^* r_2^{-2} + c_3^* r_3^{-2}) >_{17}$

$$=<13(14*9^2+1*8^2+6*2^2+8*15^2)>_{17}=16$$

$$c_3=<4^{-1}(c_0^*r_0^{-3}+c_1^*r_1^{-3}+c_2^*r_2^{-3}+c_3^*r_3^{-3})>_{17}$$

$$=<13(14*9^3+1*8^3+6*2^3+8*15^3)>_{17}=15$$

Therefore $C(x)=3+7x+16x^2+15x^3$

Verification:

Finding the product of $<<A(x)B(x)>_{x^4+1}>_{17}$ using the traditional multiplication technique

$$\begin{array}{r}
2+3x+4x^2+5x^3 \\
6+7x+8x^2+9x^3 \\
\hline
12+18x+24x^2+30x^3 \\
-35+14x+21x^2+28x^3 \\
-32-40x+16x^2+24x^3 \\
-27-36x-45x^2+18x^3 \\
\hline
3+7x+16x^2+15x^3 \\
\hline
\end{array}$$

Chapter 4

Efficient Convolvers Using The Polynomial Residue Number System Technique

In this chapter we will discuss the implementation of efficient convolvers using the PRNS technique. These cyclic convolvers are based on PRNS forward and inverse transformation mappings, which rely on complement operations(negations), additions and rotation operations. These rotation operations do not require any computational hardware. This way the complicated hardware that would otherwise be required to perform the scaling operations has now been substituted by rotators that do not require any computational hardware.

At first the problem of performing $< A(x)B(x) >_{x^8-1}$ using a 3-moduli RNS using the set $\{m_1, m_{2,} m_3\}$ and PRNS techniques is addressed.

Consider the polynomials $A(x)$ and $B(x)$ where,

$A(x) = a_0 + a_1 x + a_2 x^2 + ... + a_7 x^7$

$B(x) = b_0 + b_1 x + b_2 x^2 + ... + b_7 x^7$

$< A(x)B(x) >_{x^8-1}$ represents cyclic convolutions of sequences $(a_0, a_1, ..., a_7)$ and $(b_0, b_1, ..., b_7)$

For $x^8 - 1$ to have eight distinct roots in Z_m, each prime factor p of m must be such so that $8 \mid p-1$ or $p - 1 = 8k$ where k is a integer. Therefore $p = 8k + 1$;(by lemma 3.1)

We are going to investigate moduli of form $2^n, 2^n + 1, 2^n - 1$. We are using these moduli because the processing using these moduli is simple and efficient.

Moduli of the form 2^n are not appropriate because the only prime factor is *2* but $2 \neq 8k+1$.

Moduli of the $2^n - 1$ would be appropriate if each prime factor is of the form *8k+1*, thereby making number $2^n - 1$ of form 8k+1.

If $2^n - 1$ was of the form of 8k+1 then

$$2^n - 1 = 8k + 1$$

$$2^n = 8k + 2$$

$$k = \frac{2^n - 2}{8}$$

$$= \frac{2^n - 2}{2^3}$$

$$= 2^{n-3} - \frac{1}{4}$$

= not an integer or k is not an integer

Hence moduli of the form $2^n - 1$ are also not appropriate

Regarding the moduli of form $2^n + 1$ consider the following sets [24]

$$S_1 = \{2^1 + 1, 2^3 + 1, 2^5 + 1, 2^7 + 1, 2^9 + 1, \ldots\}$$

$$S_2 = \{2^2 + 1, 2^6 + 1, 2^{10} + 1, 2^{14} + 1, 2^{18} + 1, \ldots\}$$

$$S_3 = \{2^4 + 1, 2^{12} + 1, 2^{20} + 1, 2^{28} + 1, 2^{36} + 1, \ldots\}$$

$$S_4 = \{2^8 + 1, 2^{24} + 1, 2^{40} + 1, 2^{56} + 1, 2^{72} + 1, \ldots\}$$

$$S_5 = \{2^{16} + 1, 2^{48} + 1, 2^{80} + 1, 2^{112} + 1, 2^{144} + 1, \ldots\}$$

In general

S_d = { all numbers Nk,d such that $N_{k,d} = 2^{2^d k + 2^{d-1}} + 1; k = 0, 1, 2, 3, \ldots\}$, where d=1,2,3,…

The exponents of two of the numbers of the S_d set posses the following properties.

Property 4.1 : Exponents of adjacent numbers in any set set S_d differ by 2^d.

This means that exponents of adjacent numbers in sets $S_1, S_2, S_3, S_4, S_5, \ldots$ differ by 2,4,8,16,32,…

Property 4.2 : A number $2^n + 1$ belongs to the set S_d if and only if $< n >_{2^d} = 2^{d-1}$.

This means that the binary exponents (i.e. the exponents of 2) for numbers of the sets $S_1, S_2, S_3, S_4, S_5, \ldots$, are of the form *2k+1,4k+2,8k+4, 16k+8, 32k+16,….*

Lemma 4.1: The first number in any set S_d set is a common divisor of all the numbers in the same set.

Theorem 4.1: Any two numbers of the form $2^n + 1$ are relatively prime if and only if they belong to two different S_d sets.

We see that each element of the set S_1 has $2^1 + 1 = 3$ as a prime factor but $3 \neq 8k + 1$.

Hence, it is not possible to choose moduli from set S_1.

In the similar fashion each element of set S_2 has $2^2 + 1 = 5$ as a prime factor but $5 \neq 8k + 1$. Thus, it is not possible to choose moduli from set S_2.

Therefore we are now left with sets S_d with $d \geq 3$.

The polynomial $x^8 - 1$ can be factorized as

$$x^8 - 1 = (x^4 - 1)(x^4 + 1)$$

$$= (x^2 - 1)(x^2 + 1)(x^4 + 1)$$

$$= (x - 1)(x + 1)(x^2 + 1)(x^4 + 1) \qquad (4.1)$$

We will now calculate the roots of the polynomial $x^8 - 1$.

Roots of x^2+1 in Z_{2^n+1} are $2^{n/2}$ and $-2^{n/2}$

Verification:

$$<(2^{n/2})^2+1>_{2^n+1}=<2^n+1>_{2^n+1}=0$$

$$<(-2^{n/2})^2+1>_{2^n+1}=<(-1)^2(2^{n/2})^2+1>_{2^n+1}=<1.2^n+1>_{2^n+1}=0$$

Another form of roots for x^2+1 in Z_{2^n+1} are $2^{n/2}$ and $2^{3n/2}$

Verification:

$$<(2^{n/2})^2+1>_{2^n+1}=<2^n+1>_{2^n+1}=0$$

$$<(2^{3n/2})^2+1>_{2^n+1}=<2^{3n}+1>_{2^n+1}=<(2^n)^3+1>_{2^n+1}=<(-1)^3+1>_{2^n+1}=<-1+1>_{2^n+1}=0$$

Roots of x^4+1 in Z_{2^n+1} are $2^{n/4}, -2^{n/4}, 2^{3n/4}$ and $-2^{3n/4}$

Verification:

$$<(2^{n/4})^4+1>_{2^n+1}=<2^n+1>_{2^n+1}=0$$

$$<(-2^{n/4})^4+1>_{2^n+1}=<(-1)^4(2^{n/4})^4+1>_{2^n+1}=<1.2^n+1>_{2^n+1}=0$$

$$<(2^{3n/4})^4+1>_{2^n+1}=<2^{3n}+1>_{2^n+1}=<(2^n)^3+1>_{2^n+1}=<(-1)^3+1>_{2^n+1}=<-1+1>_{2^n+1}=0$$

$$<(-2^{3n/4})^4+1>_{2^n+1}=<(-1)^4 2^{3n}+1>_{2^n+1}=<1.(2^n)^3+1>_{2^n+1}=<(-1)^3+1>_{2^n+1}=<-1+1>_{2^n+1}=0$$

Another form of roots for x^4+1 in Z_{2^n+1} are $2^{n/4}, 2^{3n/4}, 2^{5n/4}$ and $2^{7n/4}$

Verification:

$$<(2^{n/4})^4+1>_{2^n+1}=<2^n+1>_{2^n+1}=0$$

$$<(2^{3n/4})^4+1>_{2^n+1}=<2^{3n}+1>_{2^n+1}=<(2^n)^3+1>_{2^n+1}=<(-1)^3+1>_{2^n+1}=<-1+1>_{2^n+1}=0$$

$$<(2^{5n/4})^4+1>_{2^n+1}=<2^{5n}+1>_{2^n+1}=<(2^n)^5+1>_{2^n+1}=<(-1)^5+1>_{2^n+1}=<-1+1>_{2^n+1}=0$$

$< (2^{7n/4})^4 + 1 >_{2^n+1} = < 2^{7n} + 1 >_{2^n+1} = < (2^n)^7 + 1 >_{2^n+1} = < (-1)^7 + 1 >_{2^n+1} = < -1 + 1 >_{2^n+1} = 0$

Hence the roots of $x^8 - 1$ are 1, -1, $2^{n/2}$, $2^{3n/2}$, $2^{n/4}$, $2^{3n/4}$, $2^{5n/4}$ and $2^{7n/4}$.

$$x^8 - 1 = (x - r_0)(x - r_1)(x - r_2)(x - r_3)(x - r_4)(x - r_5)(x - r_6)(x - r_7)$$

$$= (x-1)(x-(-1)(x-2^{n/2})(x-2^{3n/2})(x-2^{n/4})(x-2^{3n/4})(x-2^{5n/4})(x-2^{7n/4}) \quad (4.2)$$

One such appropriate multimoduli PRNS set is the set

$$R = \{m_1, m_2, m_3\}$$

$$= \{2^{n-4} + 1, 2^n + 1, 2^{n+4} + 1\}, n = 8k + 4, k = 1,2,3,... \quad (4.3)$$

Verification:

Since $n = 8k + 4$, $m_2 \in S_3$

Now $n - 4 = 8k$ and n-4 not odd $\Rightarrow m_1 \notin S_1$.

Also $n - 4 = 8k$ and $< 8k >_4 = 0 \neq 2 \Rightarrow m_1 \notin S_2$ (using property 4.2) and

$< 8k >_8 = 0 \neq 4 \Rightarrow m_1 \notin S_3$ (using property 4.2).

Therefore m_1 belongs to some S_d set with $d \geq 4$.

We also see the since $n + 4 = 8k + 8$ and n+4 not odd $\Rightarrow m_3 \notin S_1$ (using property 4.2) and

$< 8k + 8 >_4 = 0 \neq 2 \Rightarrow m_3 \notin S_2$ and $< 8k + 8 >_8 = 0 \neq 4 \Rightarrow m_3 \notin S_3$ (using property 4.2).

Therefore m_3 belongs to some S_d set with $d \geq 4$.

The exponent difference in m_1 and m_3 is n+4-(n-4) = 8. The two moduli m_1, m_3 do not belong to the same S_d set since the difference in adjacent exponents in any of the S_d set with $d \geq 4$ is greater than 8 (using property 4.1). Hence it can be concluded that m_1, m_3 belong to two different S_d sets with $d \geq 4$.

The modulii m_1, m_2, m_3 satisfy the requirements of theorem 4.1 i.e. m_1, m_2, m_3 belong to different sets, thus we can conclude that m_1, m_2, m_3 are pair wise prime.

Three such sets are

$$R_1 = \{2^8 + 1, 2^{12} + 1, 2^{16} + 1\} \tag{4.4}$$

The dynamic range is 36 bits this is of medium range.

$$R_2 = \{2^{16} + 1, 2^{20} + 1, 2^{24} + 1\} \tag{4.5}$$

The dynamic range is 60 bits which is of large range.

$$R_3 = \{2^{24} + 1, 2^{28} + 1, 2^{32} + 1\} \tag{4.6}$$

The dynamic range is 84 bits which is considered very large.

4.1 Diminished –1 System

We now present the basics of the diminished-1 system for processing mod $2^N + 1$. This system is going to be used in the later part of the thesis.

Let A be an integer such that $A \in Z_{2^n+1}$ where $Z_{2^n+1} = \{0,1,2,....2^n\}$ (the ring of integers mod$(2^n + 1)$). Then A needs n+1 bits for its representation.

The following shows the correspondence between normal and diminished-1 representation [25]-[26] .

Table 1 Diminished -1 System

A	Diminished –1 of A (A-1)
0=000…000(n+1 bits)	100…000= 2^n (n+1 bits)
1=000…001	00…000=0
2=000…010	00…001=1
3=000…011	00…010=2
4=000…100	00…011=3
.	.
.	.
.	.
2^n =100…000	11…111= 2^n -1

As seen from the table on previous page, the diminished –1 representation of the number 0 is 2^n and requires n+1 bits for the same. For non-zero numbers we require only n bits for representation.

4.2.1 Arithmetic Mod $(2^n + 1)$

When performing arithmetic mod $(2^n + 1)$ by using the diminished –1 system, all input operands are in diminished –1 form and the results are returned in diminished –1 form as well.

Addition mod $(2^n + 1)$

Let A,B be integers such that $A, B \in Z_{2^n+1}$. If $A = B = 0$ then $< A + B >_{2^n+1} = 0$;

If $A = 0$ and $B \neq 0$ then $< A + B >_{2^n+1} = B$;

If $A \neq 0$ and $B = 0$ then $< A + B >_{2^n+1} = A$;

If $A \neq 0$ and $B \neq 0$ then $< A + B >_{2^n+1}$ is carried out as follows

Compute the diminished – 1 form of *A,B*

$A \xrightarrow{\text{dim–1}} A - 1$

$B \xrightarrow{\text{dim–1}} B - 1$

Add the n-bit numbers *A-1* and *B-1*; Complement (negate) the carry-out, end it around and add it back. The obtained result will be the diminished – 1 form of $< A + B >_{2^n+1}$.

Example:

Let $A = (10)_{10}$, $B = (13)_{10}$

To perform $< A + B >_{2^4+1}$

$A = 10 = (1010)_2 \xrightarrow{\text{dim–1}} (1001)_2 = A - 1$

$B = 13 = (1101)_2 \xrightarrow{\text{dim}-1} (1100)_2 = B - 1$

Adding A-1 and B-1 we get

$$\begin{array}{cccccc} & & 1 & 0 & 0 & 1 \\ & & 1 & 1 & 0 & 0 \\ \hline 1 & & 0 & 1 & 0 & 1 \\ \llcorner & & & & & 0 \\ \hline & & 0 & 1 & 0 & 1 \end{array}$$

$(0101)_2 = 5 = \text{dim} - 1 \text{ of } 6.$

Negation mod $(2^n + 1)$

Let A belong to Z_{2^n+1}. If $A = 0$ then $< -A >_{2^n+1} = 0$. For the general case when $A \neq 0$ the computation of $< -A >_{2^n+1}$ is carried out as shown.

$A \xrightarrow{\text{dim}-1} A - 1$

Take the 1's complement of A-1. The obtained result will be the diminished –1 form of $< -A >_{2^n+1}$.

Example.

Let $A = (10)_{10}$

To perform $< -A >_{2^4+1}$

$A = 10 = (1010)_2 \xrightarrow{\text{dim}-1} (1001)_2 = A - 1$

$\overline{A-1} = (0110)_2 = 6 = \text{dim} - 1 \text{ form of } 7.$

Scaling by power of 2 mod $(2^n + 1)$

Let A belong to Z_{2^n+1}. If $A = 0$ then $< 2^k A >_{2^n+1} = 0$. For the general case when $A \neq 0$ the computation of $< 2^k A >_{2^n+1}$ is carried out as shown.

$A \xrightarrow{\text{dim}-1} A - 1$

Rotate the number A-1 k bits to the left where the bits shifted out of the left end and shifted into the right end are complemented. The obtained result will be the diminished -1 form of $< 2^k A >_{2^n+1}$.

Example. Let $A = (10)_{10}$

To perform $< A * 2^3 >_{2^4+1}$

$A = 10 = (1010)_2 \xrightarrow{\text{dim}-1} (1001)_2 = A - 1$

$1001 \xrightarrow{rotate\_3\_bit\_left} 1100 \xrightarrow{complement\_rest} 1\overline{\overline{100}} = (1011)_2 = 11$ (diminished –1 of 12).

4.2 Carry Save Adders and Carry Propagate Adders

We now present the basics of carry save and carry propagate adders. These adders are used in the later part of this chapter.

Carry Save Adders (CSA) are also called 3-2 counters. They are implemented for addition of three n bit numbers to produce a sum and a carry output. The CSA does not propagate carry bits. An n-bit CSA consists of n independent full adders.

The carry and sum bits are added using the carry propagate adder(CPA).

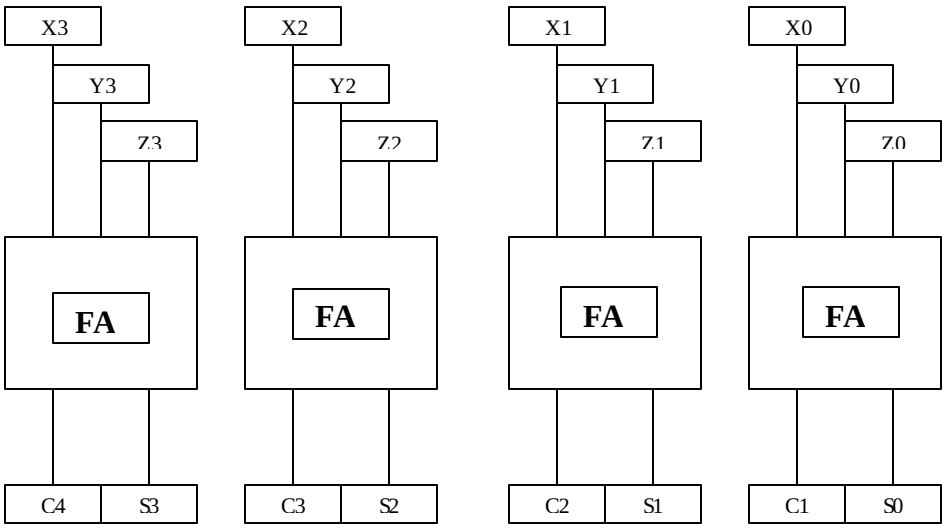


Figure 1. Carry Save Adders and Carry Propagate Adders

4.3 PRNS Transformation in Z_{2^n+1} with $n = 4k$

4.3.1 Forward Mapping

We will now present the forward mapping using the diminished –1 system.

Let $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$

Applying eq 3.1 to the above equation we have,

$$a_0^* =< a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 >_{2^n+1} \tag{4.7}$$

$$a_1^* =< a_0 - a_1 + a_2 - a_3 + a_4 - a_5 + a_6 - a_7 >_{2^n+1} \tag{4.8}$$

$$a_2^* =< a_0 + a_1.2^{n/2} + a_2.2^n + a_3.2^{3n/2} + a_4.2^{2n} + a_5.2^{5n/2} + a_6.2^{3n} + a_7.2^{7n/2} >_{2^n+1} \tag{4.9}$$

$$a_3^* =< a_0 + a_1.2^{3n/2} + a_2.2^{3n} + a_3.2^{9n/2} + a_4.2^{6n} + a_5.2^{15n/2} + a_6.2^{9n} + a_7.2^{21n/2} >_{2^n+1} \tag{4.10}$$

$$a_4^* =< a_0 + a_1.2^{n/4} + a_2.2^{n/2} + a_3.2^{3n/4} + a_4.2^n + a_5.2^{5n/4} + a_6.2^{3n/2} + a_7.2^{7n/4} >_{2^n+1} \tag{4.11}$$

$$a_5^* =< a_0 + a_1.2^{3n/4} + a_2.2^{3n/2} + a_3.2^{9n/4} + a_4.2^{3n} + a_5.2^{15n/4} + a_6.2^{9n/2} + a_7.2^{21n/4} >_{2^n+1} \tag{4.12}$$

$$a_6^* =< a_0 + a_1.2^{5n/4} + a_2.2^{5n/2} + a_3.2^{15n/4} + a_4.2^{5n} + a_5.2^{25n/4} + a_6.2^{15n/2} + a_7.2^{35n/4} >_{2^n+1} \tag{4.13}$$

$$a_7^* =< a_0 + a_1.2^{7n/4} + a_2.2^{7n/2} + a_3.2^{21n/4} + a_4.2^{7n} + a_5.2^{35n/4} + a_6.2^{21n/2} + a_7.2^{49n/4} >_{2^n+1} \tag{4.14}$$

The above set can be re-written as

$$a_0^* =< a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 >_{2^n+1} \tag{4.15}$$

$$a_1^* =< a_0 - a_1 + a_2 - a_3 + a_4 - a_5 + a_6 - a_7 >_{2^n+1} \tag{4.16}$$

$$a_2^* =< a_0 + a_1.2^{n/2} + a_2.2^n + a_3.2^n.2^{n/2} + a_4.2^{2n} + a_5.2^{2n}.2^{n/2} + a_6.2^{3n} + a_7.2^{3n}.2^{n/2} >_{2^n+1} \quad (4.17)$$

$$a_3^* =< a_0 + a_1.2^n.2^{n/2} + a_2.2^{3n} + a_3.2^{4n}.2^{n/2} + a_4.2^{6n} + a_5.2^{7n}.2^{n/2} + a_6.2^{9n} + a_7.2^{10n}.2^{n/2} >_{2^n+1} \quad (4.18)$$

$$a_4^* =< a_0 + a_1.2^{n/4} + a_2.2^{n/2} + a_3.2^{3n/4} + a_4.2^n + a_5.2^n.2^{n/4} + a_6.2^n.2^{n/2} + a_7.2^n.2^{3n/4} >_{2^n+1} \quad (4.19)$$

$$a_5^* =< a_0 + a_1.2^{3n/4} + a_2.2^n.2^{n/2} + a_3.2^{2n}.2^{n/4} + a_4.2^{3n} + a_5.2^{3n}.2^{3n/4} + a_6.2^{4n}.2^{n/2} + a_7.2^{5n}.2^{n/4} >_{2^n+1} \quad (4.20)$$

$$a_6^* =< a_0 + a_1.2^n.2^{n/4} + a_2.2^{2n}.2^{n/2} + a_3.2^{3n}.2^{3n/4} + a_4.2^{5n} + a_5.2^{6n}.2^{n/4} + a_6.2^{7n}.2^{n/2} + a_7.2^{8n}.2^{n/4} >_{2^n+1} \quad (4.21)$$

$$a_7^* =< a_0 + a_1.2^n.2^{3n/4} + a_2.2^{3n}.2^{n/2} + a_3.2^{5n}.2^{n/4} + a_4.2^{7n} + a_5.2^{8n}.2^{3n/4} + a_6.2^{10n}.2^{n/2} + a_7.2^{12n}.2^{n/4} >_{2^n+1} \quad (4.22)$$

In the diminished –1 system

$-a = \bar{a}$ where $\bar{a}$ represents a bit by bit complement of a.

Also

$< 2^{kn} >_{2^n+1} = 1$ if $k = even$

$< 2^{kn} >_{2^n+1} = -1$ if $k = odd$

Using the above two results the set of equations are rewritten as follows

$$a_0^* =< a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 >_{2^n+1} \quad (4.23)$$

$$a_1^* =< a_0 + \bar{a}_1 + a_2 + \bar{a}_3 + a_4 + \bar{a}_5 + a_6 + \bar{a}_7 >_{2^n+1} \quad (4.24)$$

$$a_2^* =< a_0 + a_1.2^{n/2} + \bar{a}_2 + \bar{a}_3.2^{n/2} + a_4 + a_5.2^{n/2} + \bar{a}_6 + \bar{a}_7.2^{n/2} >_{2^n+1} \quad (4.25)$$

$$a_3^* =< a_0 + \bar{a}_1.2^{n/2} + \bar{a}_2 + a_3.2^{n/2} + a_4 + \bar{a}_5.2^{n/2} + \bar{a}_6 + a_7.2^{n/2} >_{2^n+1} \quad (4.26)$$

$$a_4^* =< a_0 + a_1.2^{n/4} + a_2.2^{n/2} + a_3.2^{3n/4} + \bar{a}_4 + \bar{a}_5.2^{n/4} + \bar{a}_6.2^{n/2} + \bar{a}_7.2^{3n/4} >_{2^n+1} \quad (4.27)$$

$$a_5^* =< a_0 + a_1.2^{3n/4} + \bar{a}_2.2^{n/2} + a_3.2^{n/4} + \bar{a}_4 + \bar{a}_5.2^{3n/4} + a_6.2^{n/2} + \bar{a}_7.2^{n/4} >_{2^n+1} \quad (4.28)$$

$$a_6^* =< a_0 + \bar{a}_1.2^{n/4} + a_2.2^{n/2} + \bar{a}_3.2^{3n/4} + \bar{a}_4 + a_5.2^{n/4} + \bar{a}_6.2^{n/2} + a_7.2^{3n/4} >_{2^n+1} \quad (4.29)$$

$$a_7^* =< a_0 + \bar{a}_1.2^{3n/4} + \bar{a}_2.2^{n/2} + \bar{a}_3.2^{n/4} + \bar{a}_4 + a_5.2^{3n/4} + a_6.2^{n/2} + a_7.2^{n/4} >_{2^n+1} \quad (4.30)$$

Effects of multiplying a variable with powers of 2.

We rotate the equivalent number of bits as many positions as exponent of 2 indicates to the left and complement the other shifted bits.

Consider the following $k * 2^{n/2}$

Let k=1101 and n=4.

1	1	0	1
0	1	$\bar{1}$	$\bar{1}$
0	1	0	0

Consider the following $k * 2^{n/4}$
Let k=1101 and n=4.

1	1	0	1
1	0	1	$\bar{1}$
1	0	1	0

Consider the following $k * 2^{3n/4}$
Let k=1101 and n=4.

1	1	0	1
1	1	$\bar{1}$	$\bar{0}$
1	0	0	1

Figure2. Effects of Multiplying a Variable by Powers of 2

We now run an example to illustrate the forward mapping using carry save adder trees, carry propagate adder and diminished-1 system.

Let $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$

$= 2 + 3x + 4x^2 + 5x^3 + 6x^4 + 7x^5 + 8x^6 + 9x^7$ in Z_{2^4+1} domain

Here $2^n + 1 = 2^4 + 1$ so $n = 4$.

The values of coefficients of powers of *x* are

$a_0 = 2 = 0010$

$a_1 = 3 = 0011$

$a_2 = 4 = 0100$

$a_3 = 5 = 0101$

$a_4 = 6 = 0110$

$a_5 = 7 = 0111$

$a_6 = 8 = 1000$

$a_7 = 9 = 1001$

The diminished –1 values of the above coefficients are

$a_0 = 1 = 0001$

$a_1 = 2 = 0010$

$a_2 = 3 = 0011$

$a_3 = 4 = 0100$

$a_4 = 5 = 0101$

$a_5 = 6 = 0110$

$a_6 = 7 = 0111$

$a_7 = 8 = 1000$

4.3.1.1 Methodology Adopted in Performing the Calculations of Coefficient Values Using Carry Save Adder (CSA) Tree/ Carry Propagate Adder (CPA)

The 3-to-2 counter calculates the sum and the carry of each row of selected bits indicated by a box. The sum is placed in the same bit position but the carry is placed in the next higher bit position. The carry generated by the highest order bits is complemented and placed in the lowest bit position (this is indicated by the arrow in the diagram).

In the initial state there are 8 rows. Using two sets of four carry save adders we transform then into 6 rows. These 6 rows are reduced to 4 rows using two sets of four carry save adders. The 4 rows are then reduced to 3 rows using one set of carry save adders. The 3 rows are transformed into 2 rows using one set of carry save adders. These two rows contain the sum and the carry bits.

The final sum and the carry are added using the CPA. Here again the end around carry generated is complemented and added to the lowest order bit.

Detailed description of calculation of a_5^*.

$$a_5^* = < a_0 + a_1.2^{3n/4} + \bar{a}_2.2^{n/2} + a_3.2^{n/4} + \bar{a}_4 + \bar{a}_5.2^{3n/4} + a_6.2^{n/2} + \bar{a}_7.2^{n/4} >_{2^n+1}$$

Calculation of $a_1\, 2^{3n/4}$

$a_1 = 0010$

$2^{3n/4} = 2^{3*4/4} = 2^3$

We rotate the right most bit to the left and complement the other bits.

Therefore $a_1\, 2^{3n/4} = 0110$

Calculation of $\bar{a}_2\, 2^{n/2}$

$a_2 = 0011$

$\bar{a}_2 = 1100$

$2^{n/2} = 2^{4/2} = 2^2$

We rotate the last two right most bits to the left and complement the left most bits, which are placed in the right.

Therefore $\bar{a}_2 2^{n/2} = 0000$

Calculation of $a_3 2^{n/4}$

$a_3 = 0100$

$2^{n/4} = 2^{4/4} = 2^1$

We rotate the last three right most bits to the left and complement the left most bit, which is placed in the right.

Therefore $a_3 2^{n/4} = 1001$

Calculation of $\bar{a}_4$

$a_4 = 0101$

We calculate $\bar{a}_4$ by performing bit-by-bit complement.

Therefore $\bar{a}_4 = 1010$.

Calculation of $\bar{a}_5 2^{3n/4}$

$a_5 = 0110$

$\bar{a}_5 = 1001$

$2^{3n/4} = 2^{3*4/4} = 2^3$

We rotate the right most bit to the left and complement the other bits shifted to the right.

Therefore $\bar{a}_5 2^{3n/4} = 1011$

Calculation of $a_6 2^{n/2}$

$a_6 = 0111$

$2^{n/2} = 2^{4/2} = 2^2$

We rotate the last two right most bits to the left and complement the left most bits, which are placed in the right.

Therefore $a_6 2^{n/2} = 1110$

Calculation of $\bar{a}_7 2^{n/4}$

$a_{71} = 1000$

$\bar{a}_7 = 0111$

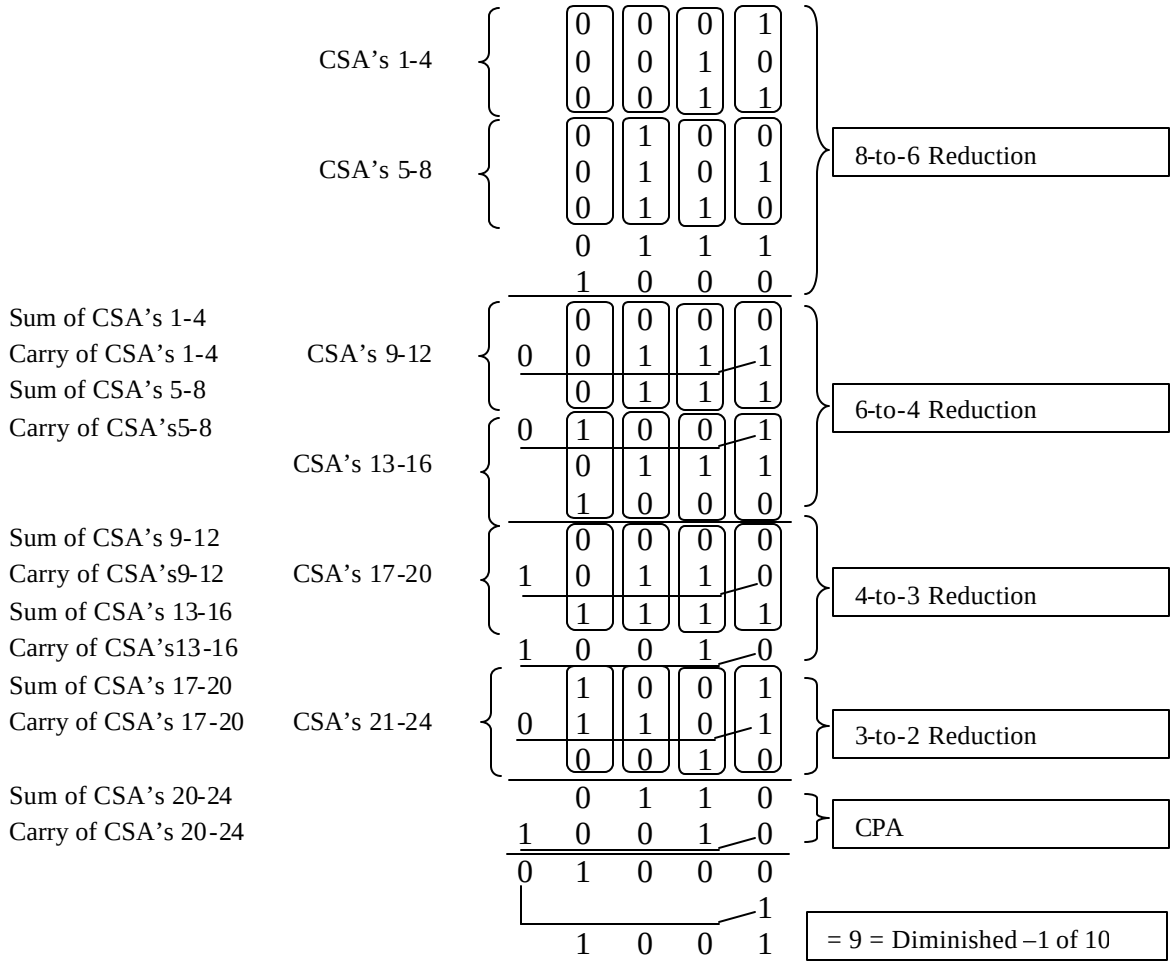
$2^{n/4} = 2^{4/4} = 2^1$

We rotate the last three right most bits to the left and complement the left most bit, which is placed in the right.

Therefore $\bar{a}_7 2^{n/4} = 1111$

These values are used in the implementation using the CSA tree/CPA shown on page 37

We evaluate the value of a_0^* (eq. 4.23) using the CSA tree/CPA implementation with a set of 3-to-2 counters.



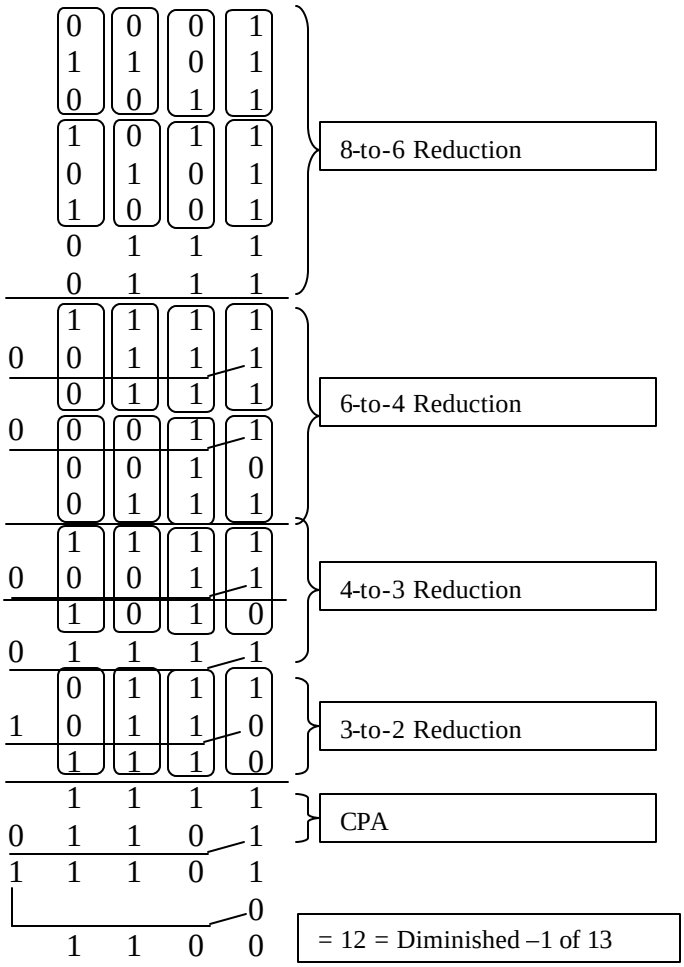
$$a_0^* =< 2+3+4+5+6+7+8+9 >_{17} =< 27 >_{17} = 10$$

The result we have from CSA tree reduction is the diminished –1 form of 10 i.e. 9

Figure 3. Forward Mapping

(Figure Continued)

We evaluate the value of a_1^* (eq. 4.24) using the CSA tree/CPA implementation with a set of 3-to-2 counters.

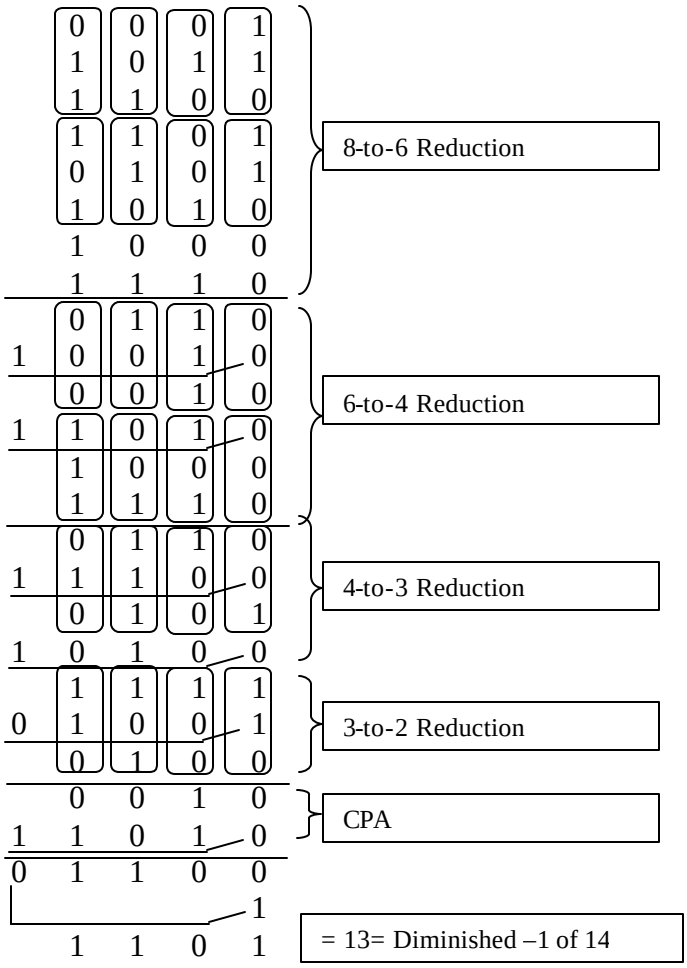


$a_1^* = <2-3+4-5+6-7+8-9>_{17} = <-4>_{17} = 13$

The result we have from CSA tree reduction is the diminished –1 form of 13 i.e. 12

(Figure Continued)

We evaluate the value of a_2^* (eq.4.25) using the CSA tree/CPA implementation with a set of 3-to-2 counters.

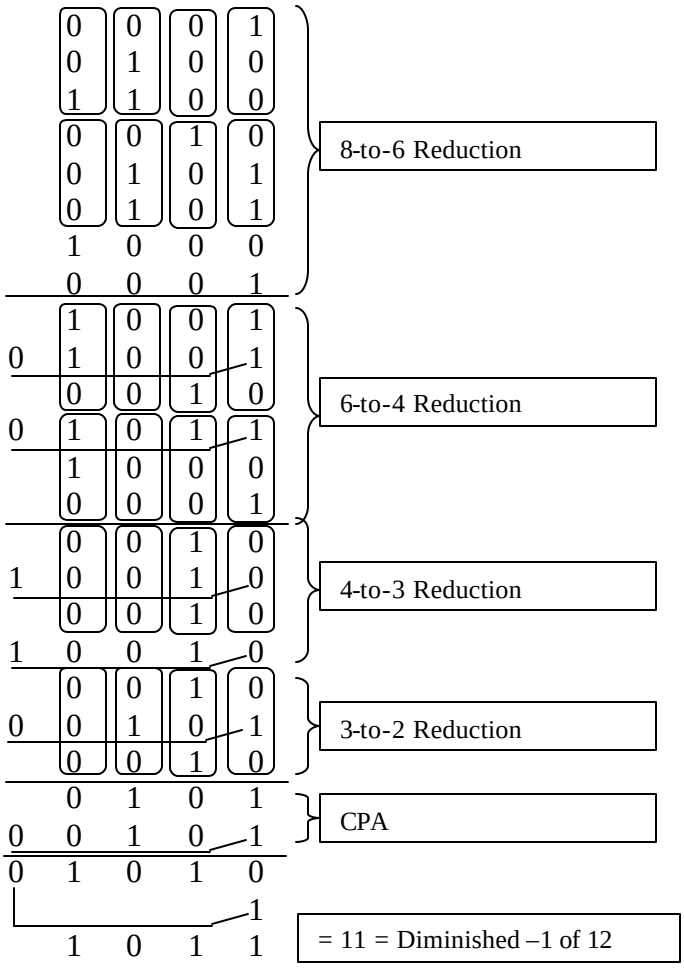


$a_2^* =< 2 + 3*4 + 4*4^2 + 5*4^3 + 6*4^4 + 7*4^5 + 8*4^6 + 9*4^7 >_{17} =< 189326 >_{17} = 14$

The result we have from CSA tree reduction is the diminished –1 form of 14i.e. 13

(Figure Continued)

We evaluate the value of a_3^* (eq.4.26) using the CSA tree/CPA implementation with a set of 3-to-2 counters.

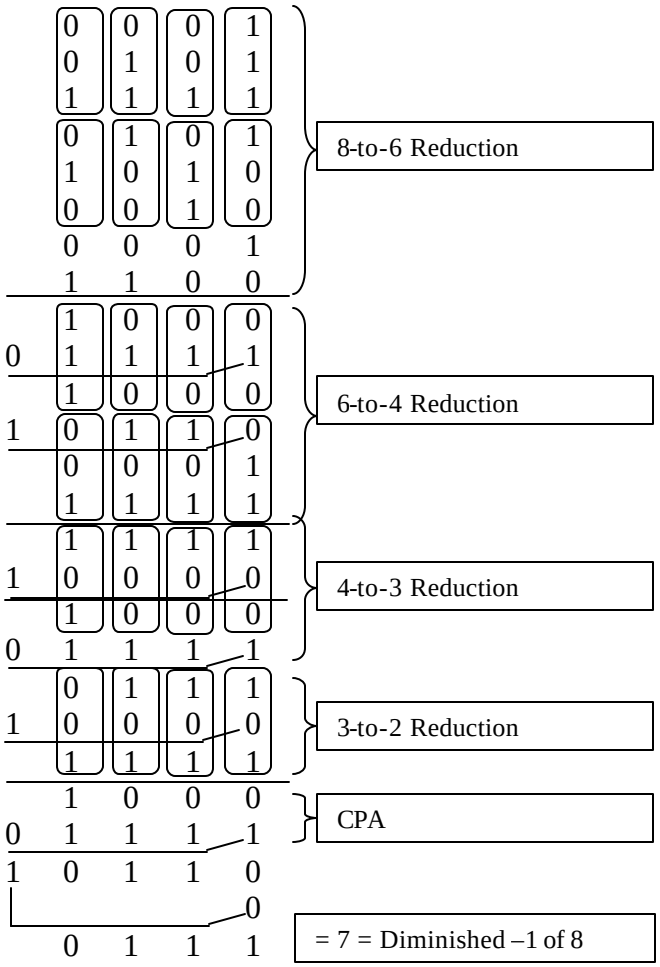


$a_3^* = < 2 - 3*4 + 4*4^2 - 5*4^3 + 6*4^4 - 7*4^5 + 8*4^6 - 9*4^7 >_{17} = < -120586 >_{17} = 12$

The result we have from CSA tree reduction is the diminished –1 form of 12 i.e. 11

(Figure Continued)

We evaluate the value of a_4^* (eq.4.27) using the CSA tree/CPA implementation with a set of 3-to-2 counters.

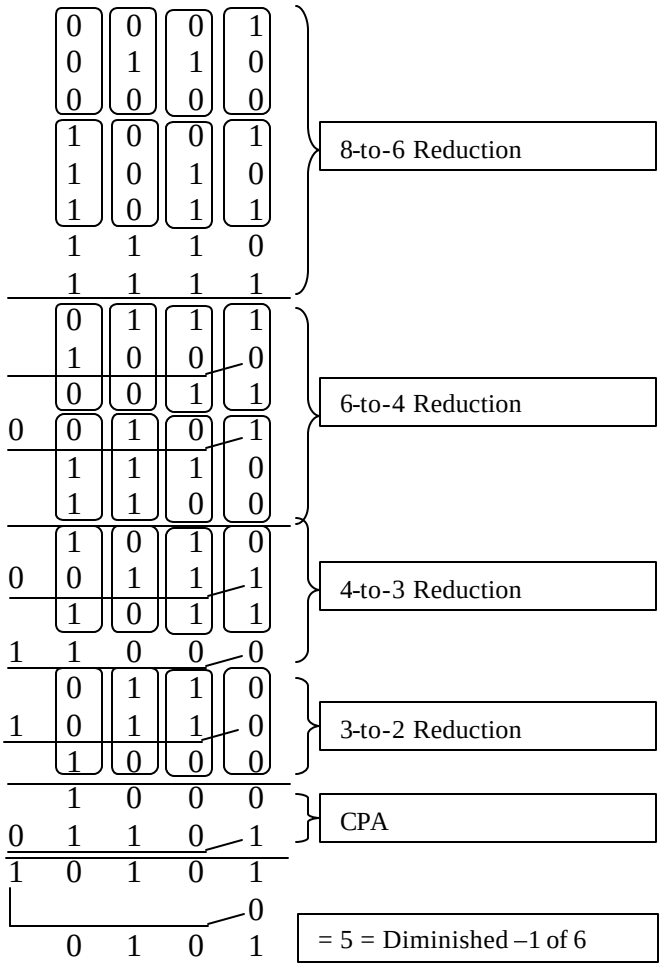


$a_4^* = <2 + 3*2 + 4*2^2 + 5*2^3 + 6*2^4 + 7*2^5 + 8*2^6 + 9*2^7>_{17} = <2048>_{17} = 8$

The result we have from CSA tree reduction is the diminished –1 form of 8 i.e. 7

(Figure Continued)

We evaluate the value of a_5^* (eq.4.28) using the CSA tree/CPA implementation with a set of 3-to-2 counters.

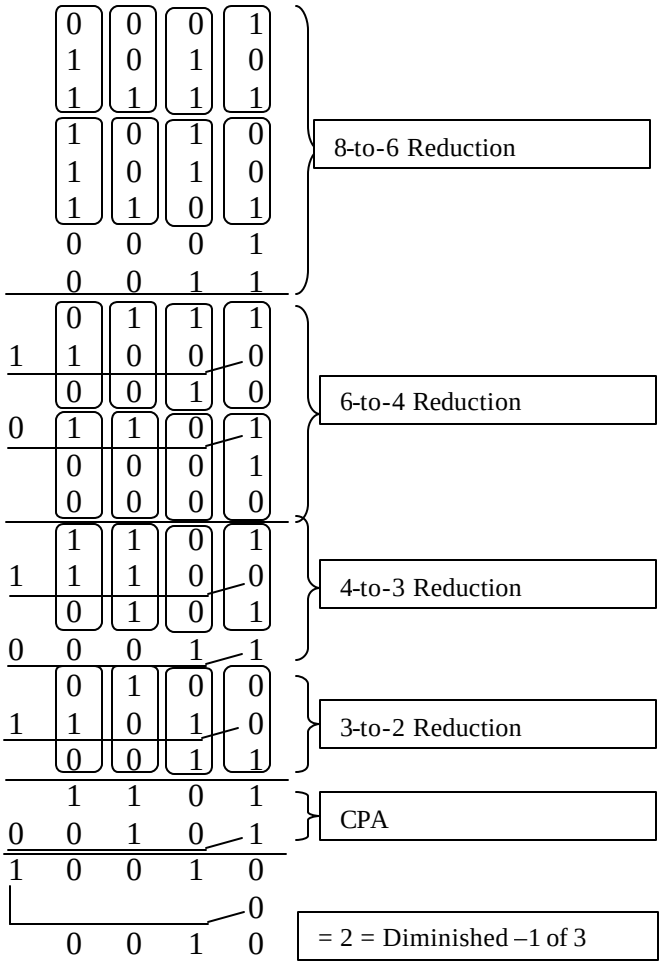


$a_5^* = <2+3*8+4*8^2+5*8^3+6*8^4+7*8^5+8*8^6+9*8^7>_{17} = <21228314>_{17} = 6$

The result we have from CSA tree reduction is the diminished –1 form of 6 i.e. 5

(Figure Continued)

We evaluate the value of a_6^* (eq.4.29) using the CSA tree/CPA implementation with a set of 3-to-2 counters.

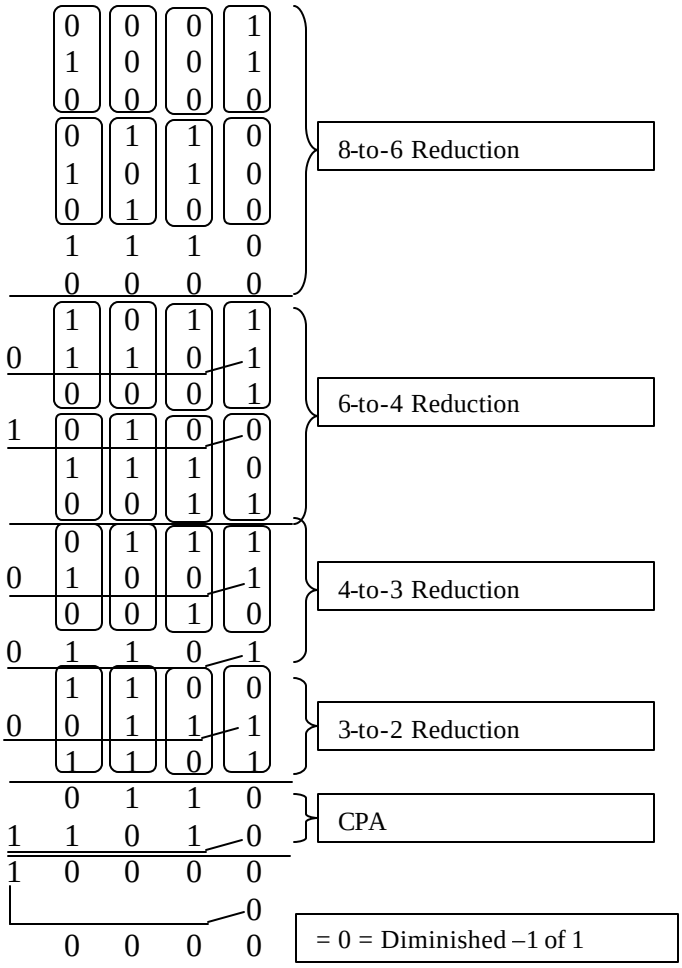


$a_6^* = <2 - 3*2 + 4*2^2 - 5*2^3 + 6*2^4 - 7*2^5 + 8*2^6 - 9*2^7>_{17} = <-796>_{17} = 3$

The result we have from CSA tree reduction is the diminished –1 form of 3 i.e. 2

(Figure Continued)

We evaluate the value of a_7^* (eq. 4.30) using the CSA tree/CPA implementation with a set of 3-to-2 counters.



$a_7^* = < 2 - 3*8 + 4*8^2 - 5*8^3 + 6*8^4 - 7*8^5 + 8*8^6 - 9*8^7 >_{17} = < -16984342 >_{17} = 1$

The result we have from CSA tree reduction is the diminished –1 form of 1 i.e. 0

4.3.2 Inverse Mapping

We now present the inverse mapping based on the result obtained in chapter 3 section 2

$$a_0 = <8^{-1}(a_0^* r_0^{-0} + a_1^* r_1^{-0} + a_2^* r_2^{-0} + a_3^* r_3^{-0} + a_4^* r_4^{-0} + a_5^* r_5^{-0} + a_6^* r_6^{-0} + a_7^* r_7^{-0})>_{2^n+1} \quad (4.31)$$

$$a_1 = <8^{-1}(a_0^* r_0^{-1} + a_1^* r_1^{-1} + a_2^* r_2^{-1} + a_3^* r_3^{-1} + a_4^* r_4^{-1} + a_5^* r_5^{-1} + a_6^* r_6^{-1} + a_7^* r_7^{-1})>_{2^n+1} \quad (4.32)$$

$$a_2 = <8^{-1}(a_0^* r_0^{-2} + a_1^* r_1^{-2} + a_2^* r_2^{-2} + a_3^* r_3^{-2} + a_4^* r_4^{-2} + a_5^* r_5^{-2} + a_6^* r_6^{-2} + a_7^* r_7^{-2})>_{2^n+1} \quad (4.33)$$

$$a_3 = <8^{-1}(a_0^* r_0^{-3} + a_1^* r_1^{-3} + a_2^* r_2^{-3} + a_3^* r_3^{-3} + a_4^* r_4^{-3} + a_5^* r_5^{-3} + a_6^* r_6^{-3} + a_7^* r_7^{-3})>_{2^n+1} \quad (4.34)$$

$$a_4 = <8^{-1}(a_0^* r_0^{-4} + a_1^* r_1^{-4} + a_2^* r_2^{-4} + a_3^* r_3^{-4} + a_4^* r_4^{-4} + a_5^* r_5^{-4} + a_6^* r_6^{-4} + a_7^* r_7^{-4})>_{2^n+1} \quad (4.35)$$

$$a_5 = <8^{-1}(a_0^* r_0^{-5} + a_1^* r_1^{-5} + a_2^* r_2^{-5} + a_3^* r_3^{-5} + a_4^* r_4^{-5} + a_5^* r_5^{-5} + a_6^* r_6^{-5} + a_7^* r_7^{-5})>_{2^n+1} \quad (4.36)$$

$$a_6 = <8^{-1}(a_0^* r_0^{-6} + a_1^* r_1^{-6} + a_2^* r_2^{-6} + a_3^* r_3^{-6} + a_4^* r_4^{-6} + a_5^* r_5^{-6} + a_6^* r_6^{-6} + a_7^* r_7^{-6})>_{2^n+1} \quad (4.37)$$

$$a_7 = <8^{-1}(a_0^* r_0^{-7} + a_1^* r_1^{-7} + a_2^* r_2^{-7} + a_3^* r_3^{-7} + a_4^* r_4^{-7} + a_5^* r_5^{-7} + a_6^* r_6^{-7} + a_7^* r_7^{-7})>_{2^n+1} \quad (4.38)$$

Calculation of Inverses of the roots

$$<r_0^{-1}>_{2^n+1} = <1^{-1}>_{2^n+1} = 1$$

$$<r_1^{-1}>_{2^n+1} = <(-1)^{-1}>_{2^n+1} = -1$$

$$<r_2^{-1}>_{2^n+1} = <(2^{n/2})^{-1}>_{2^n+1} = 2^{3n/2}$$

Verification:

$$<2^{n/2}.2^{3n/2}>_{2^n+1} = <2^{4n/2}>_{2^n+1} = <2^{2n}>_{2^n+1} = <(2^n)^2>_{2^n+1} = <(-1)^2>_{2^n+1} = 1$$

$$<r_3^{-1}>_{2^n+1} = <(2^{3n/2})^{-1}>_{2^n+1} = 2^{n/2}$$

Verification:

$$<2^{3n/2}.2^{n/2}>_{2^n+1} = <2^{4n/2}>_{2^n+1} = <2^{2n}>_{2^n+1} = <(2^n)^2>_{2^n+1} = <(-1)^2>_{2^n+1} = 1$$

$$<r_4^{-1}>_{2^n+1} = <(2^{n/4})^{-1}>_{2^n+1} = 2^{7n/4}$$

Verification:

$$< 2^{n/4}.2^{7n/4} >_{2^n+1} =< 2^{8n/4} >_{2^n+1} =< 2^{2n} >_{2^n+1} =< (2^n)^2 >_{2^n+1} =< (-1)^2 >_{2^n+1} = 1$$

$$< r_5^{-1} >_{2^n+1} =< (2^{3n/4})^{-1} >_{2^n+1} = 2^{5n/4}$$

Verification:

$$< 2^{3n/4}.2^{5n/4} >_{2^n+1} =< 2^{8n/4} >_{2^n+1} =< 2^{2n} >_{2^n+1} =< (2^n)^2 >_{2^n+1} =< (-1)^2 >_{2^n+1} = 1$$

$$< r_6^{-1} >_{2^n+1} =< (2^{5n/4})^{-1} >_{2^n+1} = 2^{3n/4}$$

Verification:

$$< 2^{5n/4}.2^{3n/4} >_{2^n+1} =< 2^{8n/4} >_{2^n+1} =< 2^{2n} >_{2^n+1} =< (2^n)^2 >_{2^n+1} =< (-1)^2 >_{2^n+1} = 1$$

$$< r_7^{-1} >_{2^n+1} =< (2^{7n/4})^{-1} >_{2^n+1} = 2^{n/4}$$

Verification:

$$< 2^{7n/4}.2^{n/4} >_{2^n+1} =< 2^{8n/4} >_{2^n+1} =< 2^{2n} >_{2^n+1} =< (2^n)^2 >_{2^n+1} =< (-1)^2 >_{2^n+1} = 1$$

Calculation of $< 8^{-1} >_{2^n+1}$

$$< 2^{-1} >_{2^n+1} = \frac{2^n + 2}{2} = 2^{n-1} + 1 = -(2^n + 1 - 2^{n-1} - 1) = -2^{n-1}$$

Therefore

$$< 8^{-1} >_{2^n+1} =< (2^{-1})^3 >_{2^n+1} = (-2^{n-1})^3 = -2^{3n-3} = -2^{2n}.2^{n-3}$$

Summarizing the above results

$$< r_0^{-1} >_{2^n+1} = 1$$

$$< r_1^{-1} >_{2^n+1} = -1$$

$$< r_2^{-1} >_{2^n+1} = 2^{3n/2}$$

$$< r_3^{-1} >_{2^n+1} = 2^{n/2}$$

$< r_4^{-1} >_{2^n+1} = 2^{7n/4}$

$< r_5^{-1} >_{2^n+1} = 2^{5n/4}$

$< r_6^{-1} >_{2^n+1} = 2^{3n/4}$

$< r_7^{-1} >_{2^n+1} = 2^{n/4}$

$< 8^{-1} >_{2^n+1} = -2^{2n}.2^{n-3}$

Applying these set of results to the inverse transform equations in diminished –1 system.

$$a_0 =< -2^{2n}.2^{n-3}(a_0^* + a_1^* + a_2^* + a_3^* + a_4^* + a_5^* + a_6^* + a_7^*) >_{2^n+1} \quad (4.39)$$

$$a_1 =< -2^{2n}.2^{n-3}(a_0^* - a_1^* + a_2^*.2^{3n/2} + a_3^*.2^{n/2} + a_4^*.2^{7n/4} + a_5^*.2^{5n/4} + a_6^*.2^{3n/4} + a_7^*.2^{n/4}) >_{2^n+1} \quad (4.40)$$

$$a_2 =< -2^{2n}.2^{n-3}(a_0^* + a_1^* + a_2^*.2^{3n} + a_3^*.2^{n} + a_4^*.2^{7n/2} + a_5^*.2^{5n/2} + a_6^*.2^{3n/2} + a_7^*.2^{n/2}) >_{2^n+1} \quad (4.41)$$

$$a_3 =< -2^{2n}.2^{n-3}(a_0^* - a_1^* + a_2^*.2^{9n/2} + a_3^*.2^{3n/2} + a_4^*.2^{21n/4} + a_5^*.2^{15n/4} + a_6^*.2^{9n/4} + a_7^*.2^{3n/4}) >_{2^n+1} \quad (4.42)$$

$$a_4 =< -2^{2n}.2^{n-3}(a_0^* + a_1^* + a_2^*.2^{6n} + a_3^*.2^{2n} + a_4^*.2^{7n} + a_5^*.2^{5n} + a_6^*.2^{3n} + a_7^*.2^{n}) >_{2^n+1} \quad (4.43)$$

$$a_5 =< -2^{2n}.2^{n-3}(a_0^* - a_1^* + a_2^*.2^{15n/2} + a_3^*.2^{5n/2} + a_4^*.2^{35n/4} + a_5^*.2^{25n/4} + a_6^*.2^{15n/4} + a_7^*.2^{5n/4}) >_{2^n+1} \quad (4.44)$$

$$a_6 =< -2^{2n}.2^{n-3}(a_0^* + a_1^* + a_2^*.2^{9n} + a_3^*.2^{3n} + a_4^*.2^{21n/2} + a_5^*.2^{15n/2} + a_6^*.2^{9n/2} + a_7^*.2^{3n/2}) >_{2^n+1} \quad (4.45)$$

$$a_7 =< -2^{2n}.2^{n-3}(a_0^* - a_1^* + a_2^*.2^{21n/2} + a_3^*.2^{7n/2} + a_4^*.2^{49n/4} + a_5^*.2^{35n/4} + a_6^*.2^{21n/4} + a_7^*.2^{7n/4}) >_{2^n+1} \quad (4.46)$$

Applying the following two results to the above set of equations we get

$< -b >_{2^n+1} =< \bar{b} >_{2^n+1}$ where $\bar{b}$ represents a bit by bit complement of b.

$< 2^{kn} >_{2^n+1} = 1$ if k is even

$= -1$ if k is odd

$$a_0 =< 2^{n-3}(\bar{a}_0^* + \bar{a}_1^* + \bar{a}_2^* + \bar{a}_3^* + \bar{a}_4^* + \bar{a}_5^* + \bar{a}_6^* + \bar{a}_7^*) >_{2^n+1} \tag{4.47}$$

$$a_1 =< 2^{n-3}(\bar{a}_0^* + a_1^* + a_2^*.2^{n/2} + \bar{a}_3^*.2^{n/2} + a_4^*.2^{3n/4} + a_5^*.2^{n/4} + \bar{a}_6^*.2^{3n/4} + \bar{a}_7^*.2^{n/4}) >_{2^n+1} \tag{4.48}$$

$$a_2 =< 2^{n-3}(\bar{a}_0^* + \bar{a}_1^* + a_2^* + a_3^* + a_4^*.2^{n/2} + \bar{a}_5^*.2^{n/2} + a_6^*.2^{n/2} + \bar{a}_7^*.2^{n/2}) >_{2^n+1} \tag{4.49}$$

$$a_3 =< 2^{n-3}(\bar{a}_0^* + a_1^* + \bar{a}_2^*.2^{n/2} + a_3^*.2^{n/2} + a_4^*.2^{n/4} + a_5^*.2^{3n/4} + \bar{a}_6^*.2^{n/4} + \bar{a}_7^*.2^{3n/4}) >_{2^n+1} \tag{4.50}$$

$$a_4 =< 2^{n-3}(\bar{a}_0^* + \bar{a}_1^* + \bar{a}_2^* + \bar{a}_3^* + a_4^* + a_5^* + a_6^* + a_7^*) >_{2^n+1} \tag{4.51}$$

$$a_5 =< 2^{n-3}(\bar{a}_0^* + a_1^* + a_2^*.2^{n/2} + \bar{a}_3^*.2^{n/2} + \bar{a}_4^*.2^{3n/4} + \bar{a}_5^*.2^{n/4} + a_6^*.2^{3n/4} + a_7^*.2^{n/4}) >_{2^n+1} \tag{4.52}$$

$$a_6 =< 2^{n-3}(\bar{a}_0^* + \bar{a}_1^* + a_2^* + a_3^* + \bar{a}_4^*.2^{n/2} + a_5^*.2^{n/2} + \bar{a}_6^*.2^{n/2} + a_7^*.2^{n/2}) >_{2^n+1} \tag{4.53}$$

$$a_7 =< 2^{n-3}(\bar{a}_0^* + a_1^* + \bar{a}_2^*.2^{n/2} + a_3^*.2^{n/2} + \bar{a}_4^*.2^{n/4} + \bar{a}_5^*.2^{3n/4} + a_6^*.2^{n/4} + a_7^*.2^{3n/4}) >_{2^n+1} \tag{4.54}$$

The a_i^* coefficients *(i=0,1,2,…,7)* from the example carried for forward transformation in diminished –1 system.

$a_0^* = 1001$

$a_1^* = 1100$

$a_2^* = 1101$

$a_3^* = 1011$

$a_4^* = 0111$

$a_5^* = 0101$

$a_6^* = 0010$

$a_7^* = 0000$

4.3.2.1 Methodology Adopted in Performing the Calculations of Coefficient Values Using Carry Save Adder (CSA) Tree/ Carry Propagate Adder (CPA)

The 3-to-2 counter calculates the sum and the carry of each row of selected bits indicated by a box. The sum is placed in the same bit position but the carry is placed in the next higher bit position. The carry generated by the highest order bits is complemented and placed in the lowest bit position (this is indicated by the arrow in the diagram).

In the initial state there are 8 rows. Using two sets of four carry save adders we transform then into 6 rows. These 6 rows are reduced to 4 rows using two sets of four carry save adders. The 4 rows are then reduced to 3 rows using one set of carry save adders. The 3 rows are transformed into 2 rows using one set of carry save adders. These two rows contain the sum and the carry bits.

The final sum and the carry are added using the CPA. Here again the end around carry generated is complemented and added to the lowest order bit.

Detailed description of calculation of a_5.

$$a_5 = <2^{n-3}(\overline{a}_0^* + a_1^* + a_2^*.2^{n/2} + \overline{a}_3^*.2^{n/2} + \overline{a}_4^*.2^{3n/4} + \overline{a}_5^*.2^{n/4} + a_6^*.2^{3n/4} + a_7^*.2^{n/4}) >_{2^n+1}$$

Calculation of $\overline{a}_0^*$

$a_0^* = 1001$

We calculate $\overline{a}_0^*$ by performing bit-by-bit complement.

Therefore $\overline{a}_0^* = 0110$.

Calculation of $a_2^* 2^{n/2}$

$a_2^* = 1101$

$2^{n/2} = 2^{4/2} = 2^2$

We rotate the last two right most bits to the left and complement the left most bits, which are placed in the right.

Therefore $a_2^* 2^{n/2} = 0100$

Calculation of $\overline{a}_3^* 2^{n/2}$

$a_3^* = 1011$

$\overline{a}_3^* = 0100$

Therefore $\overline{a}_3^* 2^{n/2} = 0010$

Calculation of $\overline{a}_4^* 2^{3n/4}$

$a_4^* = 0111$

$\overline{a}_4^* = 1000$

$2^{3n/4} = 2^{3*4/4} = 2^3$

We rotate the right most bit to the left and complement the other bits shifted to the right.

Therefore $\overline{a}_4^* 2^{3n/4} = 0011$

Calculation of $\overline{a}_5^* 2^{n/4}$

$a_5^* = 0101$

$\overline{a}_5^* = 1010$

$2^{n/4} = 2^{4/4} = 2^1$

We rotate the last three right most bits to the left and complement the left most bit which is placed in the right.

Therefore $\bar{a}_5^* 2^{n/4} = 0100$

Calculation of $a_6^* 2^{3n/4}$

$a_6^* = 0010$

$2^{3n/4} = 2^{3*4/4} = 2^3$

We rotate the right most bit to the left and complement the other bits shifted to the right.

Therefore $a_6^* 2^{3n/4} = 0110$

Calculation of $a_7^* 2^{n/4}$

$a_7^* = 0000$

$2^{n/4} = 2^{4/4} = 2^1$

We rotate the last three right most bits to the left and complement the left most bit which is placed in the right.

Therefore $a_7^* 2^{n/4} = 0001$

Summarizing the above results

$\bar{a}_0^* = 0110$

$a_1^* = 1100$

$a_2^* 2^{n/2} = 1101$

$\bar{a}_3^* 2^{n/2} = 0010$

$\bar{a}_4^* 2^{3n/4} = 0011$

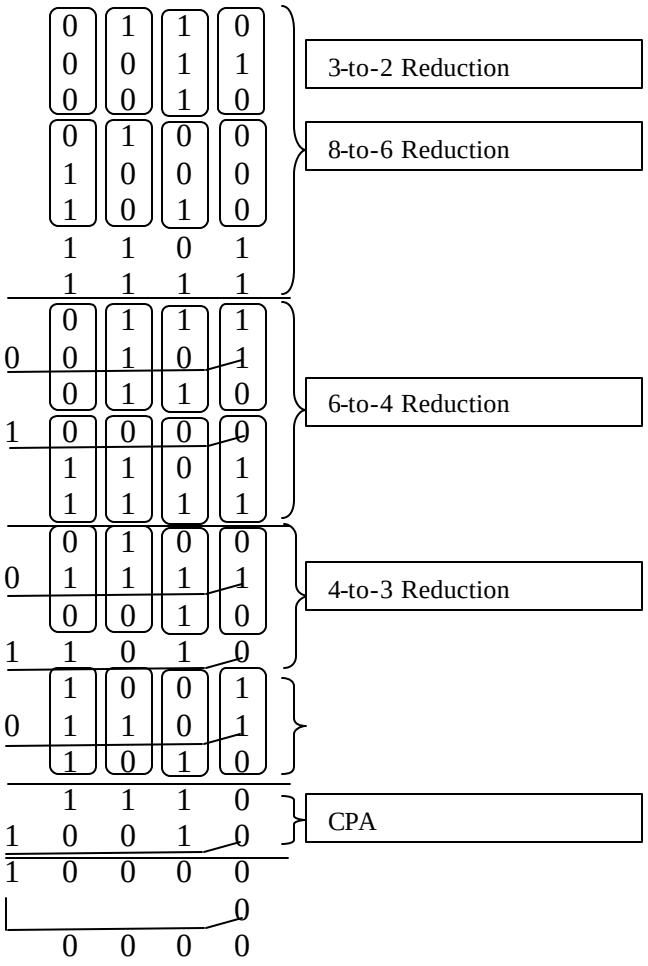
$\overline{a}_5^* 2^{n/4} = 0100$

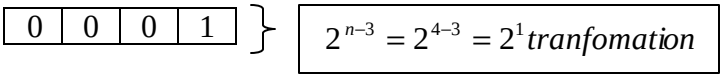
$a_6^* 2^{3n/4} = 0110$

$a_7^* 2^{n/4} = 0001$

After performing the CSA tree/CPA implementation (shown on page 53) using the above calculated values the final result is obtained after we multiply the answer obtained by $2^1 (2^{n-3} = 2^{4-3} = 2^1)$ in a similar method as shown above.

We evaluate the value of a_0 (eq. 4.47) using the CSA tree/CPA implementation with a set of 3-to-2 counters.



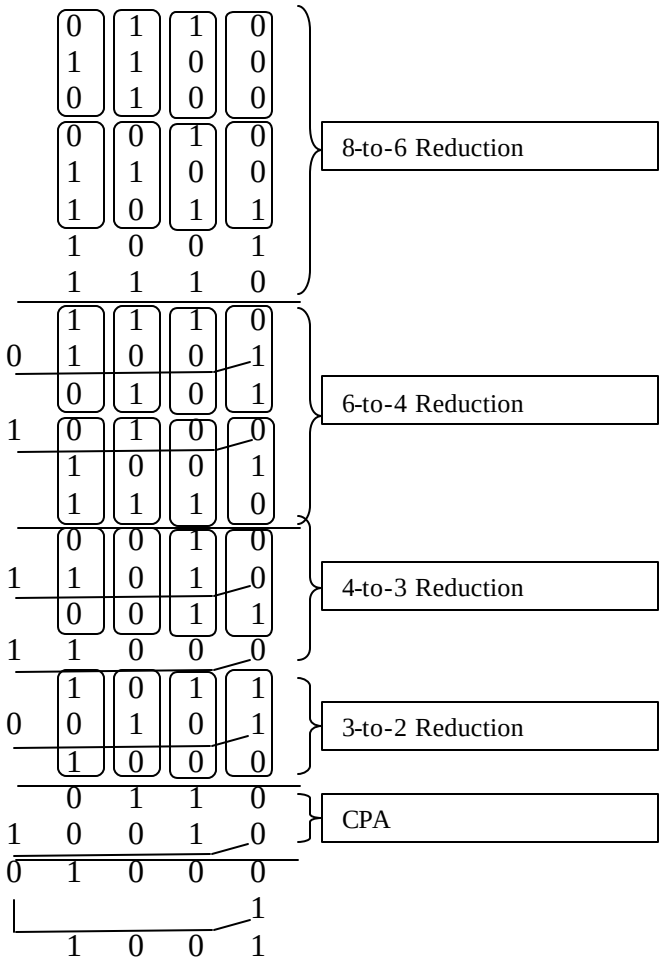


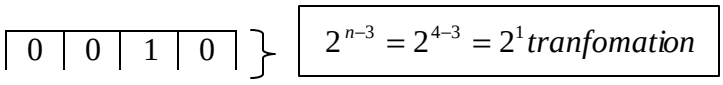
0001 = 1 = diminished –1 of 2.

Figure 4. Inverse Mapping

(Figure Continued)

We evaluate the value of a_1 (eq.4.48) using the CSA tree/CPA implementation with a set of 3-to-2 counters.

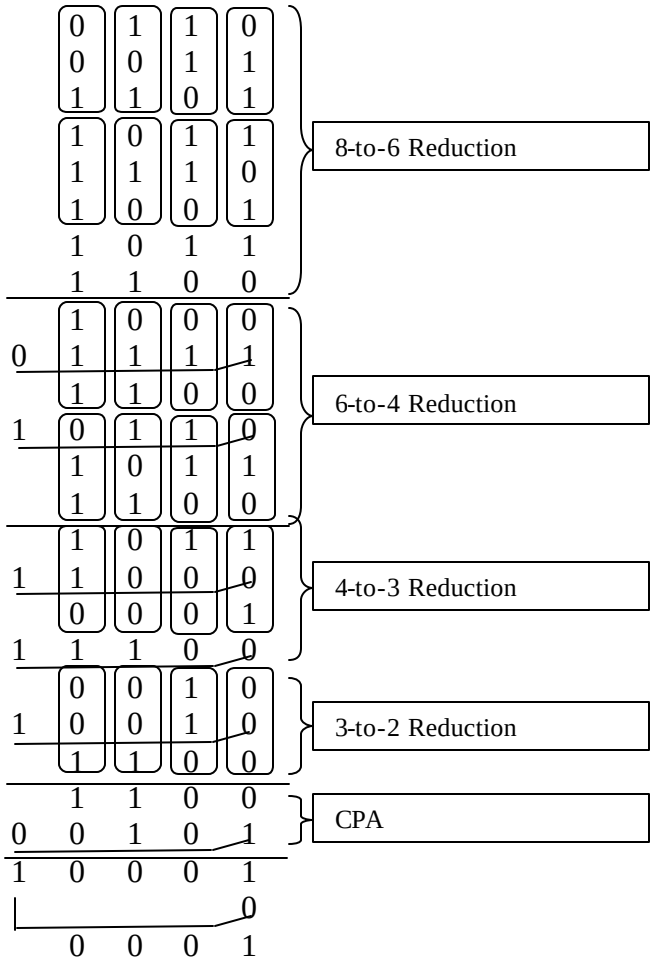


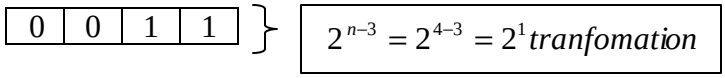


0010= 2= diminished –1 of 3.

(Figure Continued)

We evaluate the value of a_2 (eq. 4.49) using the CSA tree/CPA implementation with a set of 3-to-2 counters.

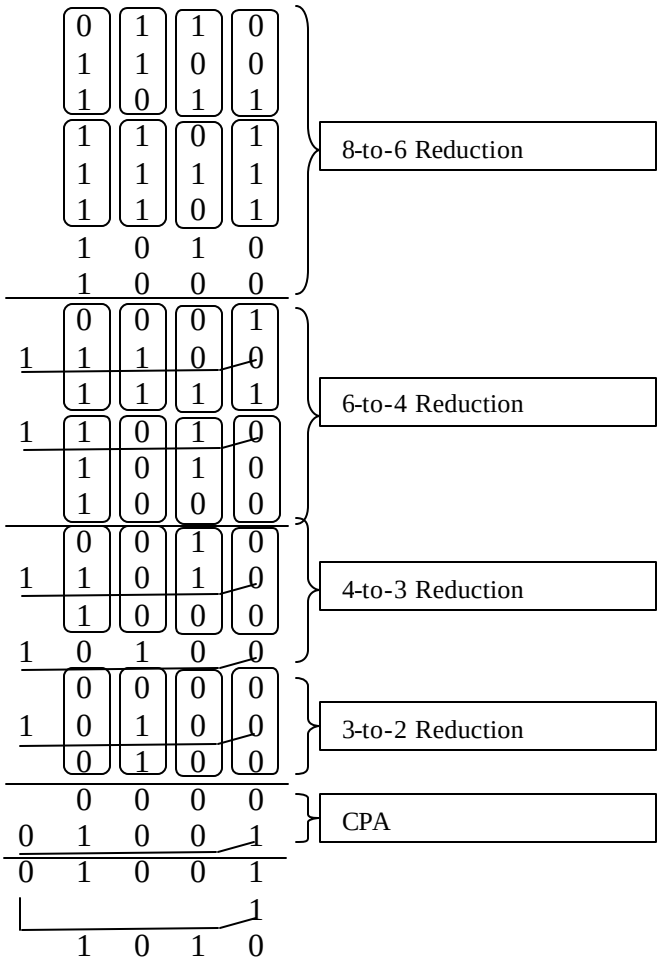




0011 = 3 = diminished –1 of 4.

(Figure Continued)

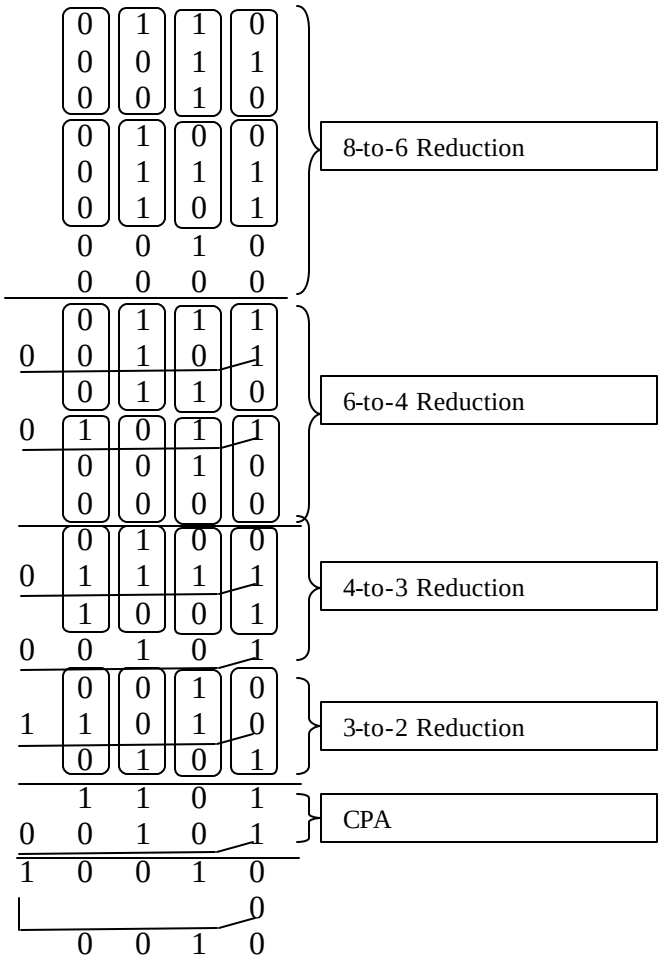
We evaluate the value of a_3 (eq. 4.50) using the CSA tree/CPA implementation with a set of 3-to-2 counters.



0100 = 4 = diminished –1 of 5

(Figure Continued)

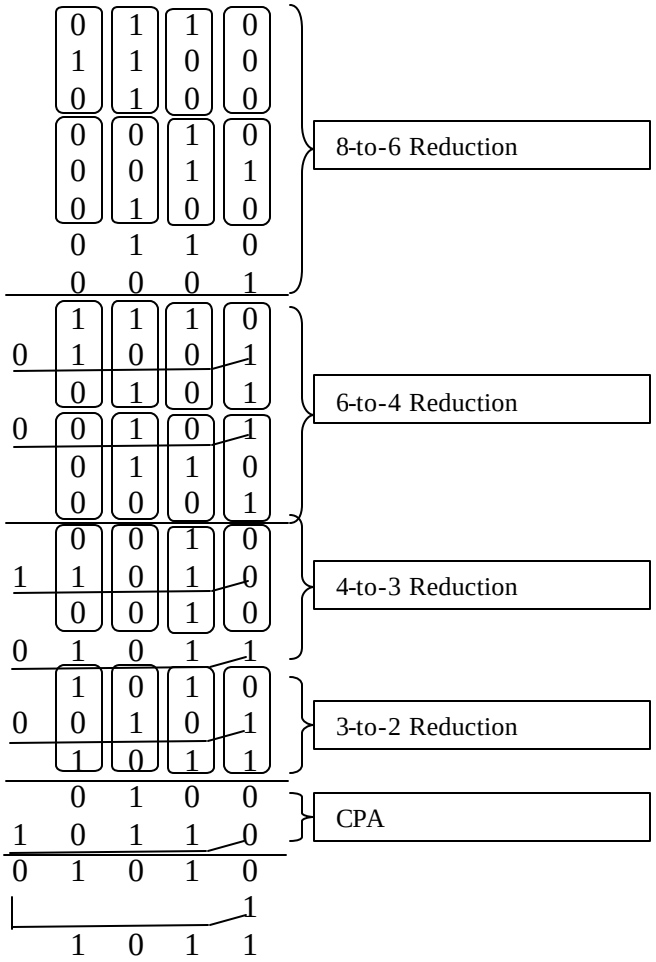
We evaluate the value of a_4 (eq. 4.51) using the CSA tree/CPA implementation with a set of 3-to-2 counters.

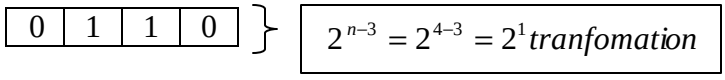


0101 = 5 = diminished –1 of 6

(Figure Continued)

We evaluate the value of a_5 (eq.4.52) using the CSA tree/CPA implementation with a set of 3-to-2 counters.

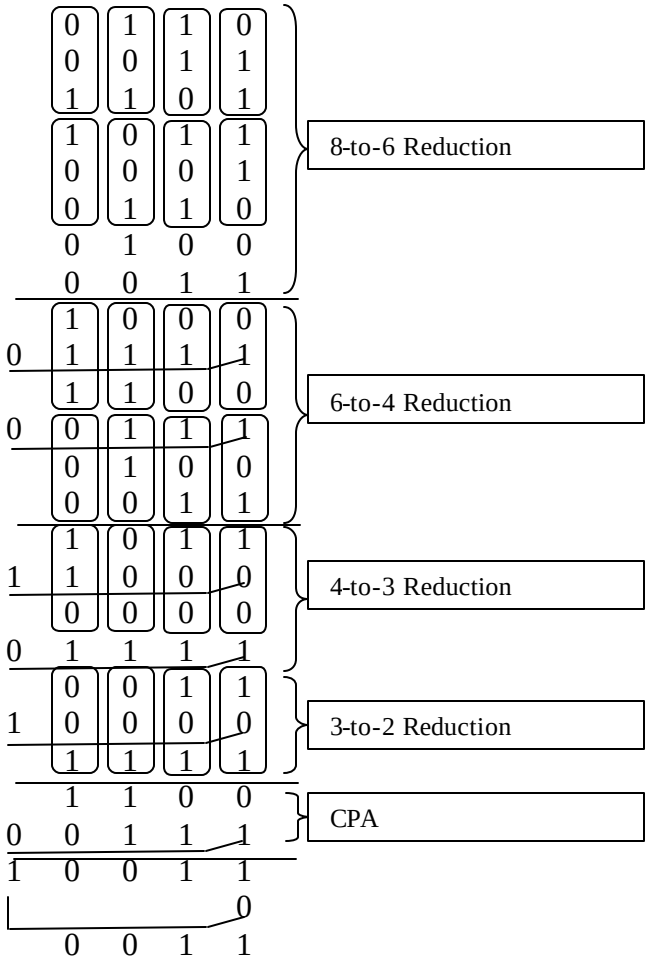


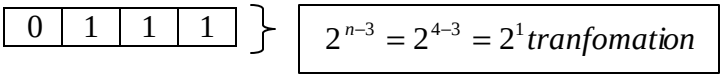


0110 = 6 =diminished –1 of 7

(Figure Continued)

We evaluate the value of a_6 (eq. 4.53) using the CSA tree/CPA implementation with a set of 3-to-2 counters.

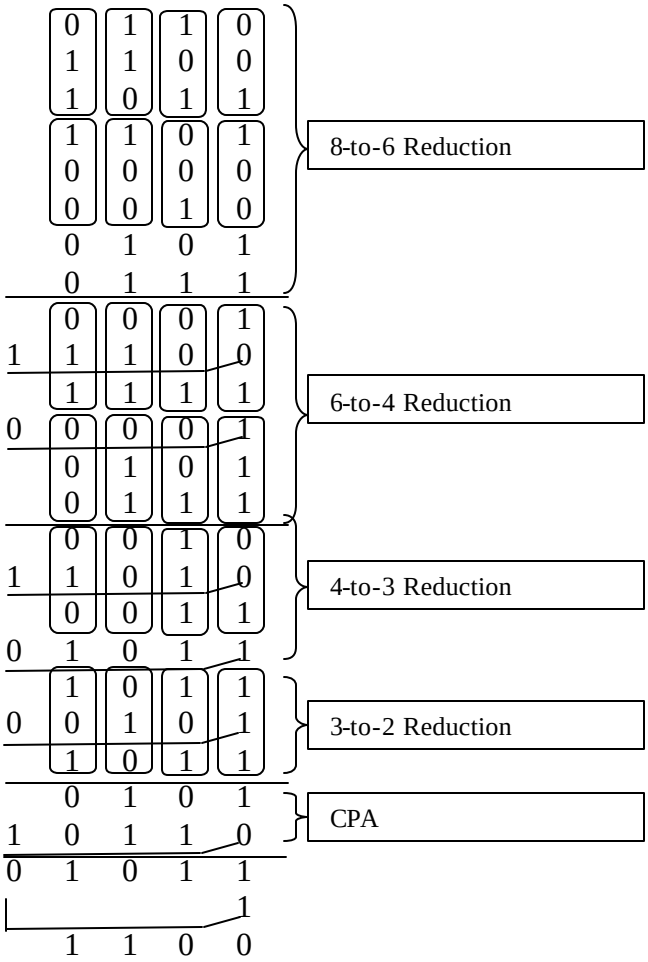




0111 = 7 = diminished –1 of 8

(Figure Continued)

We evaluate the value of a_7 (eq. 4.54) using the CSA tree/CPA implementation with a set of 3-to-2 counters.



1000 = 8 = diminished –1 of 9

We now demonstrate an example to perform a cyclic convolution of two sequences.

Consider the following two polynomials.

$A(x) = 2 + 3x + 4x^2 + 5x^3 + 6x^4 + 7x^5 + 8x^6 + 9x^7$

and $B(x) = 2 + 7x + 11x^2 + 3x^3 + 6x^4 + 8x^5 + 14x^6 + 9x^7$

We will calculate $< A(x)B(x) >_{x^8-1}$ in the ring Z_{2^n+1} where *n=4*

Step 1.

Calculation of $a_i^*, where\ i = 0,1,2,3,...7$

From the calculations performed in pg. nos. 35-42 we have the following results (in undiminished form)

$a_0^* = 10$

$a_1^* = 13$

$a_2^* = 14$

$a_3^* = 12$

$a_4^* = 8$

$a_5^* = 6$

$a_6^* = 3$

$a_7^* = 1$

Step 2

Calculation of $b_i^*, where\ i = 0,1,2,3,...7$

Here

$b_0 = 2$

$b_1 = 7$

$b_2 = 11$

$b_3 = 3$

$b_4 = 6$

$b_5 = 8$

$b_6 = 14$

$b_7 = 9$

Calculation of b_0^* using eq.4.23

$$b_0^* = <2 + 7 + 11 + 3 + 6 + 8 + 14 + 9>_{17}$$

$$= 9$$

Calculation of b_1^* using eq.4.24

$$b_1^* = <2 - 7 + 11 - 3 + 6 - 8 + 14 - 9>_{17}$$

$$= 6$$

Calculation of b_2^* using eq.4.25

$$b_2^* = <2 + (7 * 2^2) - 11 - (3 * 2^2) + 6 + (8 * 2^2) - 14 - (9 * 2^2)>_{17}$$

$$= <2 + 28 - 11 - 12 + 6 + 32 - 14 - 36>_{17}$$

$$= 12$$

Calculation of b_3^* using eq.4.26

$$b_3^* = <2 - (7 * 2^2) - 11 + (3 * 2^2) + 6 - (8 * 2^2) - 14 + (9 * 2^2)>_{17}$$

$$= <2 - 28 - 11 + 12 + 6 - 32 - 14 + 36>_{17}$$

$$= 5$$

Calculation of b_4^* using eq.4.27

$$b_4^* =< 2 + (7 * 2^1) + (11 * 2^2) + (3 * 2^3) - 6 - (8 * 2^1) - (14 * 2^2) - (9 * 2^3) >_{17}$$

$$=< 2 + 14 + 44 + 24 - 6 - 16 - 56 - 72 >_{17}$$
$$= 2$$

Calculation of b_5^* using eq.4.28

$$b_5^* =< 2 + (7 * 2^3) - (11 * 2^2) + (3 * 2^1) - 6 - (8 * 2^3) + (14 * 2^2) - (9 * 2^1) >_{17}$$

$$=< 2 + 56 - 44 + 6 - 6 - 64 + 56 - 18 >_{17}$$
$$= 5$$

Calculation of b_6^* using eq.4.29

$$b_6^* =< 2 - (7 * 2^1) + (11 * 2^2) - (3 * 2^3) - 6 + (8 * 2^1) - (14 * 2^2) + (9 * 2^3) >_{17}$$

$$=< 2 - 14 + 44 - 24 - 6 + 16 - 56 + 72 >_{17}$$
$$= 0$$

Calculation of b_7^* using eq.4.30

$$b_7^* =< 2 - (7 * 2^3) - (11 * 2^2) - (3 * 2^1) - 6 + (8 * 2^3) + (14 * 2^2) + (9 * 2^1) >_{17}$$

$$=< 2 - 56 - 44 - 6 - 6 + 64 + 56 + 18 >_{17}$$
$$= 11$$

Summarizing the above results

$$b_0^* = 9$$

$$b_1^* = 6$$

$$b_2^* = 12$$

$$b_3^* = 5$$

$$b_4^* = 2$$

$$b_5^* = 5$$

$b_6^* = 0$

$b_7^* = 11$

Step 3.

Calculation of $c_i^* =< a_i^* * b_i^* >_{17}, where\ i = 0,1,2,3,...7$

$c_0^* =< 10 * 9 >_{17}$

$= 5$

$c_1^* =< 13 * 6 >_{17}$

$= 10$

$c_2^* =< 14 * 12 >_{17}$

$= 15$

$c_3^* =< 12 * 5 >_{17}$

$= 9$

$c_4^* =< 8 * 2 >_{17}$

$= 16$

$c_5^* =< 6 * 5 >_{17}$

$= 13$

$c_6^* =< 3 * 0 >_{17}$

$= 0$

$c_7^* =< 1 * 11 >_{17}$

$= 11$

Step 4 .

Calculation of c_i, *where* $i = 0,1,2,3,...7$

Calculation of c_0 using eq.4.47

$$c_0 =< 2^1(-5-10-15-9-16-13-0-11) >_{17}$$

$$= 12$$

Calculation of c_1 using eq.4.48

$$c_1 =< 2^1(-5+10+(15*2^2)-(9*2^2)+(16*2^3)+(13*2^1)-(0*2^3)-(11*2^1) >_{17}$$

$$=< 2(-5+10+60-36+128+26-0-22 >_{17}$$
$$= 16$$

Calculation of c_2 using eq.4.49

$$c_2 =< 2^1(-5-10+15+9+(16*2^2)-(13*2^2)+(0*2^2)-(11*2^2) >_{17}$$

$$=< 2(-5-10+15+9+64-52+0-44 >_{17}$$
$$= 5$$

Calculation of c_3 using eq.4.50

$$c_3 =< 2^1(-5+10-(15*2^2)+(9*2^2)+(16*2^1)+(13*2^3)-(0*2^1)-(11*2^3) >_{17}$$

$$=< 2(-5+10-60+36+32+104-0-88 >_{17}$$
$$= 7$$

Calculation of c_4 using eq.4.51

$$c_4 =< 2^1(-5-10-15-9+16+13+0+11) >_{17}$$

$$= 2$$

Calculation of c_5 using eq.4.52

$$c_5 =< 2^1(-5+10+(15*2^2)-(9*2^2)-(16*2^3)-(13*2^1)+(0*2^3)+(11*2^1)>_{17}$$

$$=< 2(-5+10+60-36-128-26+0+22>_{17}$$
$$=15$$

Calculation of c_6 using eq.4.53

$$c_6 =< 2^1(-5-10+15+9-(16*2^2)+(13*2^2)-(0*2^2)+(11*2^2)>_{17}$$

$$=< 2(-5-10+15+9-64+52-0+44>_{17}$$
$$=14$$

Calculation of c_7 using eq.4.54

$$c_7 =< 2^1(-5+10-(15*2^2)+(9*2^2)-(16*2^1)-(13*2^3)+(0*2^1)+(11*2^3)>_{17}$$

$$=< 2(-5+10-60+36-32-104+0+88>_{17}$$
$$=2$$

Therefore $< A(x)B(x) >_{x^8-1} = 12+16x+5x^2+7x^3+2x^4+15x^5+14x^6+2x^7$ in the ring

Z_{2^n+1}

Verification of the result obtained

$$2+3x+4x^2+5x^3+6x^4+7x^5+8x^6+9x^7$$
$$2+7x+11x^2+3x^3+6x^4+8x^5+14x^6+9x^7$$

$$4+6x+8x^2+10x^3+12x^4+14x^5+16x^6+18x^7$$
$$14x+21x^2+28x^3+35x^4+42x^5+49x^6+56x^7+63x^8$$
$$22x^2+33x^3+44x^4+55x^5+66x^6+77x^7+88x^8+99x^9$$
$$6x^3+9x^4+12x^5+15x^6+18x^7+21x^8+24x^9+27x^{10}$$
$$12x^4+18x^5+24x^6+30x^7+36x^8+42x^9+48x^{10}+54x^{11}$$
$$16x^5+24x^6+32x^7+40x^8+48x^9+56x^{10}+64x^{11}+72x^{12}$$
$$28x^6+42x^7+56x^8+70x^9+84x^{10}+98x^{11}+112x^{12}+126x^{13}$$
$$18x^7+27x^8+36x^9+45x^{10}+54x^{11}+63x^{12}+72x^{13}+81x^{14}$$

$$4+20x+51x^2+77x^3+112x^4+157x^5+222x^6+291x^7+331x^8+319x^9+260x^{10}+270x^{11}+247x^{12}+198x^{13}+81x^{14}$$

$$<4+20x+51x^2+77x^3+112x^4+157x^5+222x^6+291x^7+331x^8+319x^9+260x^{10}+270x^{11}+247x^{12}+198x^{13}+81x^{14}>_{x^8-1}$$
$$=335+339x+311x^2+347x^3+359x^4+355x^5+303x^6+291x^7$$

$$<335+339x+311x^2+347x^3+359x^4+355x^5+303x^6+291x^7>_{17}$$
$$=12+16x+5x^2+7x^3+2x^4+15x^5+14x^6+2x^7$$

Hence the result.

Chapter 5

Conclusion

In this thesis we presented an approach to calculate the cyclic convolution of two N-point sequences using the PRNS techniques relying on the forward and inverse mappings. The approach relied on the fact that by keeping the roots and inverse roots of $x^8 - 1$ strictly as powers of 2 we were able to substitute the complicated scaling computations by addition and rotation operations. The rotation operations do not require any computational hardware.

The cost of computing the cyclic convolution of the sequences

$A = (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$ and $B = (b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$ is the following.

If the computation is to be performed in the ring of the integers mod $2^n + 1$, every mapped coefficient a_i^* requires *6n* full adders or (3,2) counters and a carry propagate adder (CPA). Since the sequence has eight such coefficients the total cost will be *48n* full adders and *8* CPA's. Another *48n* full adders and *8* CPA's will be required for mapping the sequence $B = (b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$ into $(b_0^*, b_1^*, b_2^*, b_3^*, b_4^*, b_5^*, b_6^*, b_7^*)$. We would require *8* multipliers mod $2^n + 1$ for performing the multiplication's in the PRNS domain i.e. $c_i^* = < a_i^* . b_i^* >_{2^n+1}$ where $i = 0,1,2,3,...,7$. Following this we would have to have the inverse transformation transforming the sequence $(c_0^*, c_1^*, c_2^*, c_3^*, c_4^*, c_5^*, c_6^*, c_7^*)$ into the sequence $(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$. The computation of every coefficient c_i requires *6n* full adders and one CPA. Since there are 8 such c_i coefficients the total cost

would be *48n* full adders and *8* CPA's. The overall cost would now be *144n* full adders, *24* CPA's and *8* multipliers modulo $2^n + 1$.

The obtained terms $c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7$ are the terms of the cyclic convolution. If the same computation was performed without using the PRNS technique the hardware requirement would be *64* multipliers modulo $2^n + 1$ and following them each term of the cyclic convolution c_i is the summation of 8 terms. A CSA tree/CPA realization of each such term requires $6n$ full adders and one n-bit CPA. Since there are 8 such terms the total hardware cost would be *48n* full adders and *8*-n bit CPA's. Therefore the total cost of the non-PRNS technique is *64* multipliers modulo $2^n + 1$, *48n* full adders and *8*-n bit CPA's.

Considering that every modulo $2^n + 1$ multiplier consists of approximately n^2 full adders and every n-bit CPA consists of n full adders, the total cost in terms of full adders for the two techniques are.

For the PRNS technique we need $(144n + 24n + 8n^2)$ full adders = $(8n^2 + 168n)$ full adders.

For the non-PRNS technique the requirements becomes $(48n + 8n + 64n^2)$ full adders = $(64n^2 + 56n)$ full adders.

For any value of $n \geq 3$

$(64n^2 + 56n) > (8n^2 + 168n)$

Therefore the PRNS technique provides significant savings over the traditional one.

The following table shows the cost in terms of full adders of computing the cyclic convolution of the two *8*-point sequences using each one of the techniques for various values of *n*.

Table 2. Comparison of PRNS and non-PRNS Technique

n	Number of full adders required for non-PRNS technique.	Number of full adders required for PRNS technique.
4	1248	800
8	4544	1856
12	11232	3168
16	17280	4736
20	26720	6560

From the above table it is evident that as the value of *n* increases the full adders requirement in case of PRNS based technique is quite less compared to the non-PRNS based technique.

References

[1] M.A.Sodersft and, W.K.Jenkins, G.A.Jultien and F.J.Taylor eds., *"Residue Number System Arithmetic: Modern Applications in Digital Signal Processing,"* New York: BME Press, 1986.

[2] W.K. Jenkins and B.J. Leon, *"The use of residue number systems in the design of finite impulse response digital filters,"* IEEE Transactions on Circuits and Systems, Vol. CAS-24, pp. 191-201, April 1977.

[3] M.A. Soderstand, *"A high-speed low-cost recursive digital filter using residue number arithmetic,"* Proceedings of IEEE, Vol. 65, pp. 1065-1067, July 1977.

[4] W.K. Jenkins, *"Recent advances in residue number techniques for recursive digital filtering," IEEE Transactions on Acoustics, Speech, and Signal Processing,"* Vol.ASSP-27, pp. 19-30, February 1979.

[5] H.K. Nagpal, G.A. Jullien and W.C. Nfiller, *"Processor architectures for two-dimensional convolvers using a single multiplexed computational element with finite field arithmetic,"* IEEE Transactions on Computers, Vol. C-32, pp. 989-1000,November 1983.

[6] M.A. Soderstrmd and B.Sinha, *"A pipelined recursive residue number system digital filter," IEEE Transactions on Circuits and Systems,"* Vol. CAS-31, pp. 415-417, April 1984.

[7] F.J. Taylor, G. Papadourakis, A. Skavantzos and A. Stouraitis, *"A radix-4 FFT using complex RNS arithmetic,"* IEEE Transactions on Computers, vol. C-34, pp. 573-576, June 1985.

[8] A. Skavantzos, *"Using quadratic residue arithmetic for computing skew cyclic convolutions,"* Electronics Letters, vol. 27, no. 23, pp. 2140-2141, November 1991.

[9] A. Skavantzos and N. Mitash, *"Implementation issues of 2-dimensional polynomial multipliers for signal processing using residue arithmetic,"* IEE Proceedings-E, vol.140, no. 1, pp. 45-53, January 1993.

[10] A. Skavantzos and T. Stouraitis, *"Polynomial residue complex signal processing,* "IEEE Transactions on Circuits and Systems-11, vol. 40, no. 5, pp. 342-344, May 1993.

[11] C.-L. Wang, *"New bit serial VLSI implementation of RNS FIR digital filters,"* IEEE Transactions on Circuits and Systenw-II, vol. 41, no. 11, pp. 768-772, November 1994.

[12] J.C. Smith and FJ. Taylor, *"A fault-tolerant GEQRNS processing element for linear systolic array DSP applications,"* IEEE Transactions on Computers, vol. 44, no. 9,pp. 1121-1130, September 1995.

[13] M.A. Bayoumi, *"A high speed VLSI complex digital signal processor based on quadratic residue number system,"* VLSI Signal Processing II, pp. 200-211,IEEE Press, 1986.
[14] W.A. Chren, *"RNS-based enhancements for direct digital frequency synthesis," IEEE Transactions on Circuits and Systems-II,"* vol. 42, no. 8, pp. 516-524, August 1995.

[15] J.V. Krogmeier and W.K. Jenldns, *"Error Detection and Correction in Quadratic Residue Number System,"* Proceedings of the 26th Midwest Symposium on Circuits and Systems, (Puebla, MX), pp. 408-41 1, August 1983.

[16] S.H. Leung, *"Application of Residue Number systems to Complex Digital Filters," Proceedings of Fifteenth Asilomar Conference on Circuits, Systems, and computers,"* (Pacific Grove, CA), pp. 70-74, November 1981.

[17] M. Abdallah and A. Skavantzos, *"New multi-moduli residue and quadratic residue systems for large dynamic ranges,"* Proceedings of 29th Asilomar Conference on Signals, Systems, and Computers, (Pacific Grove, CA, October 1995), pp. 961-965.

[18] M. Abdallah and A. Skavantzos, *"On the binary quadratic residue system with non coprime moduli, "* IEEE Transactions on Signal Processing, vol. 45, no. 8, pp. 2085-2091, August 1997.

[19] M.A. Soderstrand and G.D. Poe, *"Applications of Quadratic Like Complex Residue Number System Arithmetic to Utrasonics,"* Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, (San Diego, CA), pp.28A.5.1-28A.5.4, March 1984.

[20] R. Krishnan, G.A. Jullien and W.C. Miller, *"The Modified Quadratic Residue Number System (MQRNS) for Complex Ifigh-Speed Signal Processing,"* IEEE Trans. on Circuits and Systems, Vol. CAS-33, No. 3, pp. 325-327, March 1986.

[21] A. Skavantzos and F.J. Taylor, *"On the Polynomial Residue Number System," IEEE Transactions on SignalProcessing,"* Vol. 39, No. 2, pp. 376-382, February 1991.

[22] T. Stouraitis and A. Skavantzos, *"Multiplication of complex numbers encoded as polynomials,"* Journal of VLSI Signal Processing, vol. 3, no. 4, pp. 319-328, October 1991.

[23] M. Abdallah and A. Skavantzos, *"The multi polynomial channel polynomial residue arithmetic system,"* IEEE Transactions on Circuits and Systems-II.- Analog and Digital Signal Processing, vol. 46, no. 2, pp. 165-171, February 1999.

[24] A. Skavantzos and M. Abdallah, *"Implementation issues of the two-level residue number system with pairs of conjugate moduli,"* IEEE Transactions on Signal Processing, vol. 47, no. 3, pp. 826-838, March 1999.

[25] L. M. Leibowitz, *"A simplified binary arithmetic for the fermat number transform,"* IEEE Trans. Acoustics, Speech, and Signal Processing, vol. ASSP-24, no. 5, pp. 356-359, Oct. 1976.

[26] Z. Wang, G.A. Jullien and W.C. Miller, *"An algorithm for multiplication modulo(2^n+1),"* in Proceedings of 29th Asilomar Conference on Signals, Systems and Computers, (Pacific Grove, CA, October 1995), pp. 956-960.

Vita

Surendar Paruchuri was born in 1977, in India. He received his bachelor's degree in Electronics and Communications from the University of Madras at Chennai, India, in April 1999. He joined Louisiana State University in Fall 2000, to pursue a master's degree in Electrical and Computer Engineering.

Presently he is a candidate for the degree of Master of Science degree in Electrical Engineering.