DATA COMPRESSION USING A MULTI-RESIDUE SYSTEM (MRS)

Jyothy Melaedavattil Jaganathan, B.Tech

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

August 2012

APPROVED:

Oscar N. García, Major Professor
Shengli Fu, Committee Member
Xinrong Li, Committee Member
Murali Varanasi, Chair of the Department of Electrical Engineering
Costas Tsatsoulis, Dean of the College of Engineering
Mark Wardell, Dean of the Toulouse Graduate School

Melaedavattil Jaganathan, Jyothy. Data compression using a multi-residue system (MRS). Master of Science (Electrical Engineering), August 2012, 62 pp., 7 tables, 19 illustrations, bibliography, 40 titles.

This work presents a novel technique for data compression based on multi-residue number systems. The basic theorem is that an under-determined system of congruences could be solved to accomplish data compression for a signal satisfying continuity of its information content and bounded in peak-to -peak amplitude by the product of relatively prime moduli,. This thesis investigates this property and presents quantitative results along with MATLAB codes.

Chapter 1 is introductory in nature and Chapter 2 deals in more detail with the basic theorem. Chapter 3 explicitly mentions the assumptions made and chapter 4 shows alternative solutions to the Chinese remainder theorem. Chapter 5 explains the experiments in detail whose results are mentioned in chapter 6. Chapter 7 concludes with a summary and suggestions for future work.

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my advisor, Dr. Oscar N. García, for his continuous support during my Master's study and research and for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me throughout my research and writing of this thesis. I cannot imagine having a better advisor and mentor for my graduate study than Dr. García.

Besides my advisor, I would like to thank the rest of my thesis committee, Dr. Shengli Fu and Dr. Xinrong Li, for their encouragement, insightful comments, and hard questions.

I would also like to thank all of my friends at the University of North Texas.

Finally, I would like to thank my parents, Jaganathan MD and Sheela ME, for giving birth to me in the first place and for supporting me spiritually and providing me will all the good things I have needed throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Residue arithmetic systems, popularly known as clock arithmetic systems, are widely used in coding theory applications to error code correcting codes, cryptography, and digital signal processing utilizing the simplicity and parallelism in arithmetic calculations, then using this representation. The most interesting property of the residue number system (RNS) [1] is the absence of carrying addition and subtraction between residues and elimination of partial product formation in multiplication, which makes the residue number system [2] interesting in computing theory. Thus, mechanisms using this approach are mostly found in arithmetic operations, information theory, and signal processing applications.

1.1 Purpose

This work reports on an investigation of a novel technique of data compression using a multi residue system. Before we get into the fundamental theorem, we describe the origin and general concept of RNS.

The origin of the RNS is traced back to the first century AD by the Chinese scholar Sun Tzu in his book *Suan-Ching*. He developed the idea of the Chinese remainder theorem (CRT) that solves a system of linear congruence [3] defined by relatively prime numbers. The concept of modular arithmetic was revived by Carl Friedrich Gauss in 1801 in his book *Disquisitiones Arithmeticae*. Gauss defines congruence, residue, and modulus as follows [4]:

> If a number a divides the difference of numbers b and c, b and c are said to be congruent to a. The number a is called the modulus, and the numbers b and c are the residue of the other.

The RNS is in general defined by a modulus say m, which consists of elements from 0 to m-1, m>0. The residue of an integer X for the RNS defined by modulus m is defined by

$x \equiv X \bmod m$, where $x \in \{0, 1, \ldots.. m\text{-}1\}$ is the remainder resulting from division $\frac{X}{m}$ and the '≡' sign denotes the congruence relation between x and p. Thus, x is X congruent modulo m. Thus, using modulo m, any integer can be coded into the residue system as an integer between 0 and m-1. Fig. 1.1 shows the clock representation of RNS. The left-hand side of the figure shows *mod* 8, and the right-hand side shows general *mod M* representation.

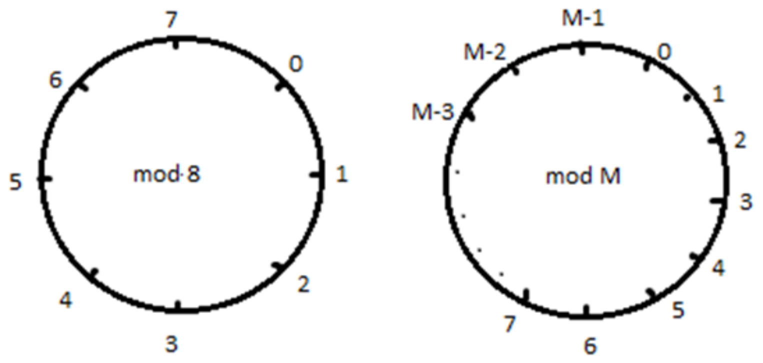


Fig. 1.1 Clock arithmetic representation

When more than one modulus is involved, the RNS is known as a multi-residue system (MRS) [5]. Table 1.1 shows a simple example of MRS. MRS can involve any number of moduli, but here for demonstration, only two moduli are used.

In abstract algebra, the set of integers defined by each of the congruence relations by modulo m refers to an equivalence class otherwise called congruence class or residue class. The congruence relation mentioned here does not need to be over prime numbers but rather over integers greater than or equal to 2. Hence, for generality, let's consider modulo m, $m \geq 3$. This forms a ring of integers denoted as Z_m^+. The sum of two integers in this ring is the remainder of the sum of integers divided by modulo m. In this equivalence class, when we say that two integers—say a and b—are equivalent, it means that they are congruent modulo m. This defines the general equivalence relation of congruence. When it comes to residue, they do not form an algebraic field unless $m = p^n$, where $n = 1,2,\ldots.n$ for finite n [6] [7].

Table 1.1 RNS encoding example

M = [3, 5] X_i	$x_1 \equiv X_1$ *mod* 3 $\in \{0, 1,2\}$	$x_2 \equiv X_2$ *mod* 5 $\in \{0, 1,2,3,4\}$
0	0	0
1	1	1
2	2	2
3	0	3
4	1	4
5	2	0
6	0	1
7	1	2
8	2	3
9	0	4
10	1	0
11	2	1
12	0	2
13	1	3
14	2	4
15	0	0

1.2 Solution of System of Congruences

Congruence therefore implies an equivalence relation between integers [8] X, x and m such as

$$x = X + m \times c,$$

where X is residue, x is integer, m is modulus, and c is any integer that solves the equation. Thus, the congruence relation between integers establishes a linear equation. The CRT by Sun Tzu solves the MRS of equations [9] provided that the moduli of the system are relatively prime.

The CRT can be summarized as follows: Let P ={p_1, p_2, ….. p_n} be n relatively prime modulus and let X = {X_1 ,X_2 ,…. X_n } be *n* arbitrary integers. Then, there exists a solution to the system of linear congruences [10] defined by these such that

$$\begin{aligned} x &\equiv X_1 \ mod\ p_1 \\ &\equiv X_2 \ mod\ p_2 \\ &\vdots \\ &\equiv X_n \ mod\ p_n \end{aligned}$$

where x is a unique residue modulo, $P = \prod_{i=1}^{n} p_i$.

Relatively prime moduli refer to moduli whose gcd = 1; i.e., *gcd*(p_1, p_2, ….. p_n) = 1. This is only a sufficient condition for CRT to work; there are exemptions to it. The solution to such a set of congruences is given by the following:

$$x \equiv X_1 \times a_1 \times \frac{P}{p_1} + X_2 \times a_2 \times \frac{P}{p_2} + \ldots\ldots\ldots + X_n \times a_n \times \frac{P}{p_n}\ (mod\ P),$$

where $P = p_1 \times p_2 \times \ldots\ldots \times p_n$ and a_i is obtained as $a_i \times \frac{P}{p_i} = 1\ mod\ p_i$.

To demonstrate how CRT solves for a system of moduli, let's consider three moduli: 5, 7, and 9 and all the numbers from 1 to the product of 5, 7, and 9; i.e., 315.

$$x \equiv 2\ mod\ 3$$

$$x \equiv 3\ mod\ 5$$

$$x \equiv 2\ mod\ 7$$

Since *gcd*(3,5,7) = 1, they are relatively prime. According to CRT,

$$\begin{aligned} x &\equiv 2 \times a_1 \times \frac{105}{3} + 3 \times a_2 \times \frac{105}{5} + 2 \times a_3 \times \frac{105}{7}\ (mod\ 105) \\ &\equiv 2 \times a_1 \times 35 + 3 \times a_2 \times 21 + 2 \times a_3 \times 15\ (mod\ 105), \end{aligned}$$

where, $a_1 \times 35 \equiv 1\ mod\ 3$, $a_2 \times 21 \equiv 1\ mod\ 5$ and $a_3 \times 15 \equiv 1\ mod\ 7$;

i.e., $a_1 \equiv 35\ mod\ 3 \equiv 2$; $a_2 \equiv 21\ mod\ 5 \equiv 1$ and $a_3 \equiv 15\ mod\ 7 \equiv 1$.

Therefore, $x \equiv 2 \times 2\times 35 + 3 \times 1 \times 21 + 2 \times 1 \times 15 \ (mod\ 105) \equiv 233\ mod\ 105 \equiv 23$, which is the solution for the given set of congruences.

1.3 Other Methods of Solutions

Apart from the CRT, there are other methods for solving the linear system of congruences, the extended Euclidean algorithm and the Aryabhatta remainder theorem [11] [12]. The extended Euclidean algorithm involves long division and requires the computation of inverses, whereas the Aryabhatta remainder theorem is similar to the CRT in its approach. For this work, CRT was initially chosen because of its popularity and simplicity. Also, we propose a finite table look-up to solve the system of congruences, constructed using CRT.

CHAPTER 2

BASIC THEOREM

Data transmission, both analog and digital, refers to the transmission of information from source to destination. In today's world, data transmission occurs in every walk of life. Owing to advancement in the usage of computers, we need not only to pass binary-encoded text messages, as we used to during the 1920s with the telegraph system, but we also need to transmit and store other digitalized information such as potentially large audio and video files.

Data transmission and storage can be costly processes. The cost of transmission or storage is directly proportional to the amount of data. And hence it is important to use information encoding and decoding schemes such that only the most relevant data is transmitted. Thus, data encoding/decoding and data reduction play a key role in transmission.
Encoding for comparison refers to the process of putting symbolic data into a particular format utilizing the same symbols, and decoding refers to putting it back in the original format. The purpose of data encoding/decoding isn't limited to data reduction but may also include secure transmission.

2.1 Data Compression

Data compression or source coding can be of two kinds: lossless and lossy. In lossless compression, data is reduced by eliminating redundancy but preserving all information. Lossy compression selects the most important information needed for an "acceptable" decoded message and eliminates the rest. By "acceptable," we mean that the user of the decoded message is not likely to perceive the information lost such as in JPEG image encoding. Data compression usually reduces the usage of network resources in transmission or storage but comes with an additional cost of hardware for encoding at the source and decoding at the destination. Data compression dates back to 1948 when Shannon presented a criterion that guides data compression for source coding known as the Shannon-Nyquist criterion [13] [14].

Basically, encoding/decoding schemes involve solving a set of equation as in $Ax = B$, where A is the encoding scheme, x is the original signal/data involved, and B is the encoded form. To reconstruct and recover the data, this equation has to be solved for $x = BA^{-1}$. Thus, in general, the process of decoding implies solving a system of linear equations [15]. In the case of data compression, this problem involves the solution of an under-determined system of equations [16] [17]. The recent idea of "compressive sensing" [18] [19] [20] throws light into this issue in detail. Thus, the motivation of this work was derived from the idea of solving a system of under-determined equations [21] [22] [23] that would facilitate efficient compression and a simple encoding and decoding scheme.

2.2 Early Approach

This work presents a novel encoding/decoding scheme using MRSs introduced in Chapter 1. The system of equations was shifted to a new class of integers called residue class involving congruences. Thus, the linear equation [24] [25] transforms into the following:

$$\mathrm{Ax}\ (mod\ \mathrm{p}) \equiv \mathrm{B} \text{ where } p \text{ is the prime number.}$$

Since the original motivation for this work came from the consideration of compressive sensing [26] [27], the same signal used in Cleve Moler's magic reconstruction [28] problem was considered first as an example. This signal corresponded to the signal generated by the "1" key of a touch-tone telephone. The signal is the sum of sinusoids and continuous and small valued signals.

$$x(\mathrm{t}) = \frac{\sin(1394\pi t) + \sin(3266\pi t)}{2}$$

The continuously varying signal in the time domain was sampled to obtain the discrete values. Sampling frequency is irrelevant in this context since our aim was to successfully encode the signal in an MRS [29] and decode it using the CRT [9].

2.3 Multi-Residue Coding

In order to work on multi-residue coding [1], three relatively prime moduli were chosen. There are no restrictions to the number of moduli to be chosen, but three was thought to be optimum since two might be too few and four might be a little more than needed.

Before converting the signal to an MRS [30], the discrete values of the signal were analyzed and some pre-processing was done for efficient decoding. Table 1.1 shows how the residue system handles negative numbers. Once the negative values are converted to the residue system, it would be impossible to decode them to obtain the original values back. Hence, it was found necessary to shift the entire set of values to positive half by adding the peak-to-peak value to the signal.

On further analysis, the discrete signal values were found to be nominal. Converting these nominal values to MRS wouldn’t make much of a difference since the moduli chosen were much greater than these values. Thus, bringing the signal to the bounds of the product of the moduli was found to be the best. We know that in an MRS [31], the bounds are determined by the product of the moduli. We know that the solution to MRS is defined by prime moduli p_1, p_2, and p_3:

$$\begin{aligned} x &\equiv X_1 \ \mathit{mod}\ p_1 \\ &\equiv X_2 \ \mathit{mod}\ p_2 \\ &\equiv X_3 \ \mathit{mod}\ p_3 \end{aligned}$$

is $x \in \{0,1,2,\ldots. p_1 \times p_2 \times p_3\}$. Thus, the input signal was augmented to the limit of the product of the moduli by multiplying with a ratio of peak to peak of signal to the product of the moduli. Multiplying served not only to bring the signal to the bounds of moduli considered, but it also served to save the decimal values. The augmented signal was rounded off since the CRT cannot handle decimal values.

A sampled, bounded positive signal $x(\mathrm{t}) < \prod p_i$ i= {1,2,3} for all t and quantized as an integer $x(\mathrm{r})$ was converted into an MRS using three relative prime moduli p_1, p_2, and p_3. This yielded three sets of residues for each sample such as X_1, X_2 and X_3. Of these three sets of residue, one set, say X_3, was removed, resulting in the reduction of data to be sent and thus defining a set of an under-determined set of congruence equations. Thus, the two sets of moduli were sent to the decoder section.

2.4 Basic Theoretical Approach

The basic theorem by Oscar N Garcia states: for a signal that remains bounded in peak to peak amplitude within the product of relatively prime moduli $\prod p_i$ and satisfies continuity in its information content, we can solve an under determined system of congruences and accomplish data compression.

The basic idea presented here was to reconstruct the original signal from the reduced set of congruences while solving for them using the CRT using the previous value of the signal. Therefore, the first sample of the augmented signal was transferred to the decoder section without converting it into an MRS. The decoder consisted of a module to compute the solution to a system of equations defined above using the CRT and a module to bring the decoded signal back to the limits of the original signal. Thus, at the decoder at a given time t, residues of sample at t and previous sample at $t\text{-}\Delta t$ were processed.

To demonstrate the process, let's consider three moduli 2, 3, and 5 that are relatively prime and all odd values between 1 and the product of (2, 3, 5); i.e., 30.

Table 2.1: MRS encoding

x	X_1 *mod* 2	X_2 *mod* 3
1	-	-
3	1	0
5	1	2
7	1	1
9	1	0
11	1	2
13	1	1
15	1	0
17	1	2
19	1	1
21	1	0
23	1	2
25	1	1
27	1	0
29	1	2

To begin with, the initial value of sample, 1, was sent to the decoder without converting it into MRS. The decoder then received the first set of residue corresponding to the second sample 3; that is, 1 and 0. Given only two residues, we know from table 2.1 that the possible solutions are 3, 9, 15, 21, and 27. From this list of possible solutions, the correct answer is found taking into consideration the continuity of the transmitted signal. Thus, the correct answer was assumed to be the one closest to the previous sample 1; i.e., 3.

The scheme used here depends on the continuity of the signal and is not applicable to any random signal. The extent of continuity or permissible rate of change of signal value is discussed in Chapter 3 under background considerations.

The decoded signal is brought back to the vicinity of the original signal by dividing the decoded value by the ratio mentioned above and then subtracting the peak of the original signal from it. Thus, the expected error in this process comes from rounding off.

To sum up, an encoding and decoding scheme is proposed here which uses the properties of a continuously varying signal, RNS [3], and CRT. The signal is sampled, made positive, brought to the bounds of product of moduli, quantized, and converted to the system of residue using the first two moduli. The initial sample is send to decoder without encoding it. At the decoder at a given time *t*, sample at is found with the help of two residues of sample at *t* and the previous sample at $t\text{-}\Delta t$ using the CRT. It is assumed that the signal is continuous enough such that the sample at *t* is the closest one to sample at $t\text{-}\Delta t$ from a list of possible solutions for the first two residues at *t*.

The application of the theorem basically transforms the integers from integer domain to z complete finite residue class. To further explain the process that takes place during the execution of algorithm, let's consider the solution at time *t*. The solution at time *t* lies in a two dimensional plane, say with $m_1 \times m_2$ on its *x* axis and m_3 on its y axis. The solution at time t projects the solution at time $t+\Delta t$ onto another plane whose dimensions are same. This projection would be exactly the intersection of two circles whose radii are m_1 and m_2. Fig. 2.1 gives a pictorial view of process explained.

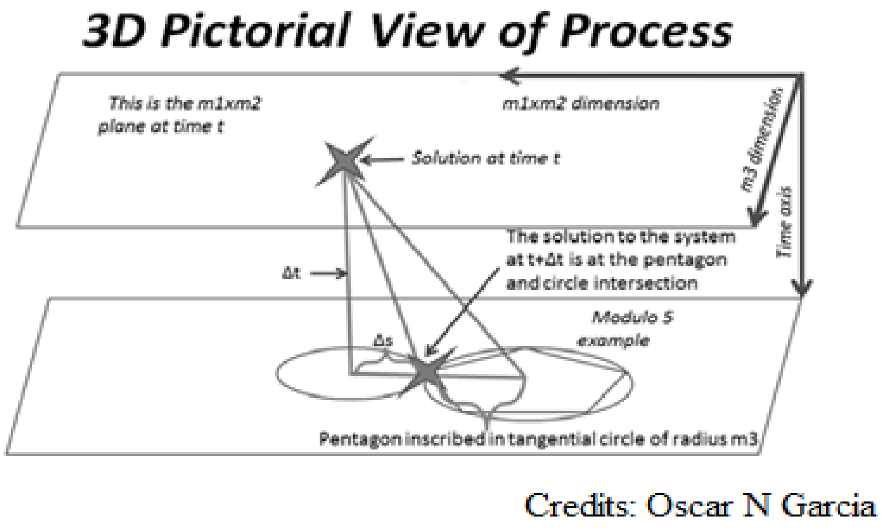

3D Pictorial View of Process
This is the m1xm2 plane at time t
m1xm2 dimension
m3 dimension
Time axis
Solution at time t
Δt
The solution to the system at t+Δt is at the pentagon and circle intersection
Modulo 5 example
Δs
Pentagon inscribed in tangential circle of radius m3
Credits: Oscar N Garcia

Fig.2.1 3D pictoral view of algorithm.

As seen in figure, if the modulus is 5, a pentagon would be inscribed in the circle whose one edge would be at the intersection of two circles or solution at time *t +Δt*.

CHAPTER 3

BACKGROUND CONSIDERATIONS

Chapter 2 presented a novel data encoding/decoding mechanism using an MRS. This chapter deals with the details of the assumptions presented in Chapter 2, justifying each of them.

3.1 Assumptions

Following are the three assumptions for the proposed algorithm:

1) The moduli selected must be relatively prime and odd.
2) The signal must have a functional relation.
3) The signal must be bound by the product of the relatively prime moduli selected.

Let's consider the first condition. The moduli selected should be relatively prime; i.e., the gcd of any two of them be one, because the CRT guarantees a unique solution when moduli are relatively prime. There are exceptions to these conditions; but to ensure that the algorithm works, we consider only relatively prime moduli. We know that the decoding program chooses the correct next sample from a list of all possible solutions such that the closest one to the previous sample would be the correct choice. This is possible only if the moduli selected are odd. If they are even, there will be two numbers that will be equidistant to the given previous sample, which would force the system to take a decision at 0.5 probability that could risk failure of the entire process.

Let's consider the second condition: the entire algorithm is based on the fact that the signal is continuous and has a slow rate of change of amplitude; i.e., the amplitude of the signal doesn't vary abruptly. In other words, the signal should have a functional relation and cannot be random. So to test the algorithm, we considered smooth continuous signals like the sum of sinusoids and speech signal, such that the next sample is the closest one to the present sample.

Moving on to the third condition that the boundary of the signal should be within the limits of the product of moduli. Consider the sampled signal x(n), according to the condition:

i.e., $0 < x(\mathrm{n}) < \prod p_i$.

We know MRS for the defined primes, say p_1, p_2 and p_3 comprises of integers 0 to $P = p_1 \times p_2 \times p_3$. And integer greater than P will be converted to an integer within this bound.

Now consider a signal whose boundaries are very much larger than the product of prime.

i.e., $x(\mathrm{n}) >> \prod p_i$where i={1,2,3}

At the encoding end, the signal values get converted into an MRS, where its values get reduced to within the range of the product of moduli. During decoding, the CRT would give all possible solutions with the range mentioned above, and the result would be an incorrect output.

Now let's consider a signal whose range is very much less than product of moduli.

i.e., $x(\mathrm{n}) << \prod p_i$ where I ={1,2,3}

In this case, since the values are very much smaller, conversion into multi-residue wouldn't bring much changes and moreover the difference between consecutive samples would be much smaller such that the nearest neighbor prediction has higher chances of being error.

Thus, the above-mentioned are the three necessary conditions for the proposed algorithm to work correctly.

CHAPTER 4

RESIDUE TABLE

The new algorithm discussed here is so far based on the CRT. However, on further investigation, we were convinced by the fact that the CRT could be substituted by a table of all integers ranging from 0 to the product of the moduli arranged in a manner respective to their moduli.

4.1 Rationale for the Use of a Table

The idea was to save the time the process took in order to compute the possible solutions by the CRT each time. The table was meant to act as a look-up table for CRT solutions. Thus, the dimension of the two-dimensional table would be $m_1 \times m_2$ for column and m_3 for row or vice versa, either way being correct. At a given time t, the column (or row) of sample at t would be selected using the product of two residues out of the three that we have been using, say $m_1 \times m_2$, to designate the column and then m_3 to determine the row. The solution would be selected as the one giving the minimum absolute difference with sample at $t\text{-}\Delta t$. Fig. 4.1 gives a pictorial representation of this process.

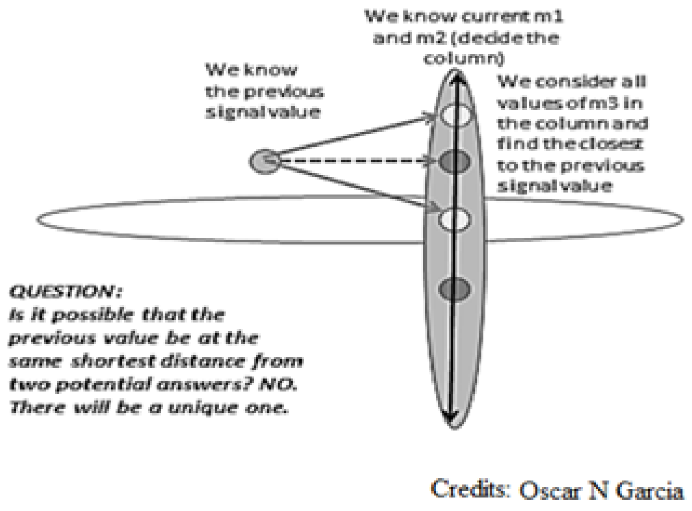

We know current m1 and m2 (decide the column)
We know the previous signal value
We consider all values of m3 in the column and find the closest to the previous signal value
QUESTION: Is it possible that the previous value be at the same shortest distance from two potential answers? NO. There will be a unique one.
Credits: Oscar N Garcia

Fig. 4.1 Pictorial representation of decoding process using table

4.2 Different Tables

The table discussed above isn't a unique one for the given moduli. It varies based on the way moduli and corresponding remainders are used in the rows and columns. Tables 4.1, 4.2, and 4.3 show three possible residue tables for moduli 3, 5, and 7.

Table 4.1 Table style 1 for P = [3 5 7]

m_3\ $r(m_1 \times m_2)$	1,1	2,2	0,3	1,4	2,0	0,1	1,2	2,3	0,4	1,0	2,1	0,2	1,3	2,4	0,0
1	1	92	78	64	50	36	22	8	99	85	71	57	43	29	15
2	16	2	93	79	65	51	37	23	9	100	86	72	58	44	30
3	31	17	3	94	80	66	52	38	24	10	101	87	73	59	45
4	46	32	18	4	95	81	67	53	39	25	11	102	88	74	60
5	61	47	33	19	5	96	82	68	54	40	26	12	103	89	75
6	76	62	48	34	20	6	97	83	69	55	41	27	13	104	90
0	91	77	63	49	35	21	7	98	84	70	56	42	28	14	0

Table 4.2 Table style 2 for P= [3 5 7]

m_3\ $r(m_1 \times m_2)$	0,0	0,1	0,2	0,3	0,4	1,0	1,1	1,2	1,3	1,4	2,0	2,1	2,2	2,3	2,4
0	0	21	42	63	84	70	91	7	28	49	35	56	77	98	14
1	15	36	57	78	99	85	1	22	43	64	50	71	92	8	29
2	30	51	72	93	9	100	16	37	58	79	65	86	2	23	44
3	45	66	87	3	24	10	31	52	73	94	80	101	17	38	59
4	60	81	102	18	39	25	46	67	88	4	95	11	32	53	74
5	75	96	12	33	54	40	61	82	103	19	5	26	47	68	89
6	90	6	27	48	69	55	76	97	13	34	20	41	62	83	104

Table 4.3 Table style 3 for P= [3 5 7]

m_3\ $r(m_1,m_2)$	0,0	0,1	0,2	0,3	0,4	1,0	1,1	1,2	1,3	1,4	2,0	2,1	2,2	2,3	2,4
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
3	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
4	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
5	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
6	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
0	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104

Thus, clearly. these tables differ.

4.3 No Search at the Expense of More Memory

Though this two-dimensional table saves calculation time and is 12 times faster in this simple example when compared to solving for CRT for every single sample, we found that a modification of a two-dimensional table would work faster. The idea was to use a three-dimensional table with elements prearranged, not only in the order of moduli but also with respect to the distance from one another, thus avoiding search time for the shortest distance element in the column. In short, a third dimension based on distance between elements of other columns was built on top of the two dimensional table as a modification. Fig. 4.2 explains the idea of three-dimensional table pictorially.

The three axes for the three-dimensional table are residues from $m_1 \times m_2$ on x-axis, residue from m_3 on y-axis (or vice versa) and closest element with respect to elements in residue ($m_1 \times m_2$) with a length of $m_1 \times m_2$. In other words, the two-dimensional table forms the base of the three-dimensional table as shown in Fig. 4.2. The three-dimensional table arranges the elements in a two-dimensional table with the closest element in the next row when counting clockwise, as the third dimension, i.e., height of the three-dimensional table. Since the table which forms base for this table isn't a table, the three-dimensional table also isn't unique. While using the three-dimensional table at the decoding end, the residues of the first two moduli of the previous sample at $t - \Delta t$ give the row number; the residue of the third moduli of the previous sample at $t - \Delta t$ gives the column number; and the residues of the first two moduli of the present sample at t give the height component.

Therefore sample at t = 3D table $\{r(m_3)_{t-\Delta t}, r(m_1 \times m_2)_{t-\Delta t}, r(m_1 \times m_2)_t\}$

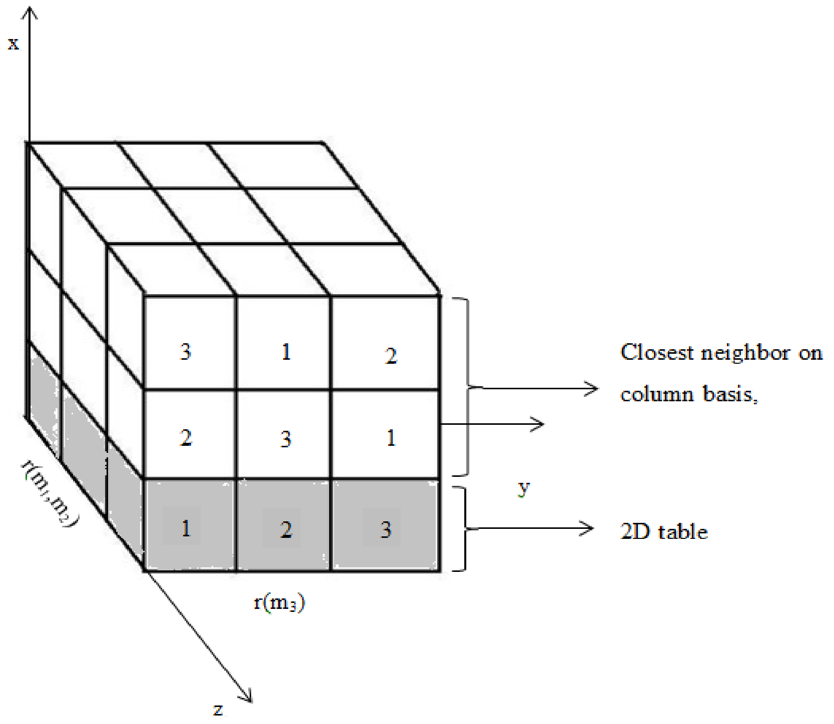
The addressing varies according to the way the two-dimensional table is arranged. The size of the three-dimensional table would be m_3 by $m_1 \times m_2$ by $m_1 \times m_2$ (or) $m_1 \times m_2$ by m_3 by m_1

$\times\ m_2$. The three-dimensional table, as mentioned before, saves execution time because there isn't any time involved in searching or calculating results as before, at the expense of memory to build and retain the three-dimensional table for given moduli, thus consuming more power and area. This could be a serious issue when large moduli are selected.

4.4 Two-Dimensional Canonical Table

Later on, the idea of using only two moduli was considered. The table for the same was constructed as shown in Table 4.4.

Table 4.4 Two-dimensional canonical table for P = [5 7] using two moduli



Col no.\r(m₁)	0	1	2	3	4	5	6
1	0	1	2	3	4	5	6
2	7	8	9	10	11	12	13
3	14	15	16	17	18	19	20
4	21	22	23	24	25	26	27
5	28	29	30	31	32	33	34

Fig. 4.2 Pictorial representation of 3D table

The new two-dimensional table was derived from Table 4.3 by replacing two moduli with two. This new table was named “canonical residue table” and provided the best direct addressing performance and lowest memory requirements of all the tables. Having the elements arranged in ascending order eliminated the need for search and made it easier to use a new decoding scheme, which addressed the elements directly. It should be noticed that this table is part of a family of tables whose columns have the same arithmetic difference between corresponding elements as between the values of the residues of modulo m_2.

CHAPTER 5

SEQUENCE OF EXPERIMENTS

The algorithm and a different residue table proposed for data compression were subjected to experimentation with different inputs. Many modifications of the original algorithm and experiments with different data were carried out, which resulted in changes in the algorithm and results as mentioned in this thesis. This chapter deals with the experiences and lessons learned during this aspect of the research.

5.1 Platform

The experiments mentioned below were carried out on a MATLAB platform, which is a linear algebraic computing language. To solve the CRT, initially the Symbolic Math toolbox in MATLAB was used. The rest of the experiments were carried out in the basic platform of MATLAB.

The Symbolic Math MATLAB Toolbox provides tools for solving and manipulating symbolic math expressions and performing arithmetic with variable precision. The toolbox contains hundreds of MATLAB symbolic functions that leverage the MuPAD engine for tasks such as symbolic differentiation, integration, simplification, transforms, and solving system of equations. The Symbolic Math toolbox also includes the MuPAD programming language, which is optimized for handling and operating user-created symbolic math expressions. Programming in MuPAD can include a broad set of libraries of MuPAD functions in frequently used mathematical areas, such as calculus and linear algebra, as well as specialized areas, such as number theory and combinatory.

5.2 Initial Motivation: Case 1

As mentioned in Chapter 1, the work began with an effort to find alternatives to solve an underdetermined system of equations involved that appeared in compressive sensing [32] [33] for data compression [34], and then the idea of taking the entire system to a multi-residue framework came up. Since tutorial work began by reading about "compressive sensing" [35] [36] [37], the sinusoidal signal used in Cleve Moler's "Magic Reconstruction" [28] was first used to see the feasibility of a multi-residue approach. The signal corresponded to the "A" key of a touch-tone telephone. The basic algorithm of converting signals into residue classes and reconstructing them using the CRT was tested successfully on this signal; this result encouraged us to follow this research line.

What follows is part of the code from the encoding section. Here, the augmented input data is converted to MRS using the '*mod*' function in MATLAB:

```
for id = 1 : dM-1
    mod_dat(:, id) = mod(dat_aug, M(id));
end
```

The following snippet of code shows how CRT was called on MATLAB using Symbolic Math toolbox in this program.

```
command = ['numlib::ichrem(' z_str ',[5,7,11])'];
p(i) = evalin(symengine, command);
```

Initially, the parameter passed onto the Symbolic Math toolbox was converted into symbols/string, and then the CRT command *numlib::ichrem* was executed.

5.3 Speed Up: Cases 2 and 3

In an effort to utilize the signal for a real-world signal, a sound file was considered. For the purpose of a uniform comparison, a sound file, SA1.wav file (ID: 55092), was downloaded from the TIMIT database and used in all the following experiments reported here. The sound file had a sampling frequency of 16k Hz, duration of 2.9248 seconds with 16 bits/sample by default. The *.*wav* file was read using *wavread* command on MATLAB as given below.

```
[din,fs]= wavread('timi_in');
```

The signal was tested successfully using the same algorithm just by replacing the sampled sinusoidal signal with the new one. The output was saved as another sound (*.*wav*) file using the *wavwrite* function.

```
wavwrite(dout ,fs ,'timi_out');
```

The acceptability of the result was confirmed by computing the root mean square (RMS) error between input and output signals and by listening to the output sound signal. The RMS error was in the range of 10^{-6}, which we considered practically lossless. The following code shows the RMS error calculation:

```
err = dout(2:end) - din(2:end) %does not include first sample
err_rms = sqrt((sum(err.^2))/length(err));
```

However, solving the CRT by using the Symbolic Math toolbox was a long, time-consuming approach since each time there was a call for it, data from the basic MATLAB environment had to be converted to symbolic form. After the computation of the result, the answer was converted back from symbolic form to the MATLAB environment. This command and conversion call between MATLAB and the Symbolic Math toolbox was found to be a limitation to the usage of this algorithm in real-time applications. Hence, CRT was written in the MATLAB platform to avoid using the slower Symbolic Math toolbox process. This proved to be a good decision since it sped up the execution of the program by 12 times. Below is a snippet of the CRT code (see Appendix A for the complete code).

```
for j=1:length(a)
   temp = invmodn(M/m(j),m(j));
   x=x+ a(j)*(M/m(j))*temp;
   x=mod(x,M);
end;
```

where *invmodn(*x,m*)* found the modular inverse of argument *x* with respect to modulus *m*.

The idea for compression was to project the signal to a residue class that would reduce the number of bits per sample for storage and transmission by using all but one of the remainders. For example, the TIMIT sentence, by default, had 16 bits per sample; but compressing it with moduli 7, 31, and 127 and transmitting the residues from 7 and 31 (3 and 5 bits, respectively) reduced the number of bits per sample by 50%. Later on, a provision for bit shift was done to enforce 8 bits per sample. This approach was done by multiplying the second higher-order residue by 2^n, where n is the number of bits in the first lower-order moduli. In the case of 7, 31, and 127, the residue was multiplied by 2^3, 3 corresponding to number of bits in 7 at the encoding end. Fig. 5.1 depicts the process. The algorithm written in MATLAB is as follows:

```
for i=1:length(dat)
   mod_dat(i) =  mod_dat(i,1)+(8*mod_dat(i,2));
end
```

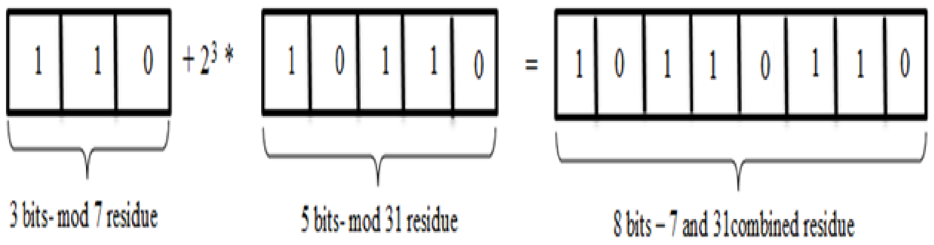


Fig. 5.1 Combining two residues to single byte

We transmit the residues of *mod* 7 and *mod* 31 and, by combining those to a single byte, obtain 50% compression. At the decoder, the encoded samples were first converted to an 8-bit binary representation and then the first lower-order 5 bits were considered to be residues of *mod* 31 and the higher-order 3 bits were considered to be the residues of *mod* 7.

Later on, a thought to encode the signal initially using differential pulse-code modulation (DPCM) was suggested. DPCM was used at the encoder prior to coding it in the RNS [8] and was decoded at other end after decoding it by using CRT. This, however, was found not to be very effective since in the minimum number of bits to be sent was 8 bits, the same as the case without using DPCM. Hence, DPCM was dropped to reduce the numerical computation complexity of the algorithm. Encoding two consecutive DPCM bytes was considered but not pursued at this time.

5.4 Further Speed-Up: Cases 4 and 5

In the beginning, the main intent of this work was to come up with a methodology for data compression using multi-residue codes [38] [39]. After working on it with sound waves, sinusoidal signals, and implementing CRT module, we realized it was desirable to operate in real time and to take a note of timing for the algorithm; therefore, timing plots were included in the programs. This gave information on the time taken by segments of the program and allowed their selective optimization. The decoding segment of the process was found to be consuming two-thirds of the total execution time. The time taken was later reduced by the inclusion of two-dimensional residue table decoder instead of CRT (see Appendix B for the code); nevertheless, the decoding end dominated the execution time owing to the search process. This was again improved by using a three-dimensional table where the result of converting the multi-residue to an integer was done by direct addressing. This eliminated the search. As mentioned before, we saved execution time at the expense of using more memory space for the three-dimensional table (see Appendix C for code). The final approach used a canonical two-dimensional table that did not require a search, allowed direct addressing, and had lower memory requirements.

5.5 Best Table with Zero Error: Cases 6 and 7

We introduced a new decoding scheme that addresses the elements of the table, thus eliminating a search. This gave us a speedier process and also fewer computations. In an effort to reduce computations, we also used two moduli instead of three. Thus, the two-dimensional table was reconstructed as shown in Table 4.4. The following is a snippet of code from encoding and decoding part of the new scheme.

```
%---Encoding-----
a = mod(f2(:,:,1),M(2))+1;
Isave = a;

Isend=((f2(:,:,1)-a)./M(2))+1;
%---Decoding--------

for j=1:n
  for i=1:m
    r(i,j)=Table(Isend(i,j),Isave(i,j));
  end
end
```

As can be seen, different variables named Isend and Isave were created that address the table directly, avoiding search. Thus, the scheme totally eliminated the search process and hence improved timing. On further analysis, we realized that the error from the two-dimensional table and the three-dimensional table was due to rounding off the sampling data. The representation of the sampled value in MATLAB was in double precision floating point format, which necessitated rounding off values before encoding. Therefore, in the new program, we read data using int16 format, thus avoiding usage of the rounding-off function and thereby achieving zero RMS error. The following part of the MATLAB code (See Appendix E for code) shows the same.

```
y = wavread('timi_in.wav','native');
```

We have considered how we achieved sequential improvements in computation time, complexity, and memory utilization with a two-dimensional canonical table and direct addressing without search and eliminated the error within the length (16 bits) of the data when it was read into the workspace by using the ‘native’ MATLAB option. We give quantitative results in the next chapter.

CHAPTER 6

RESULTS

This chapter presents the results obtained in each experimental case studied and with each of the associated experiments. The first two experiments used the example of Cleve Moler's compressive sensing problem [28], and the others used TIMIT's *.wav file in raw format after eliminating the header.

6.1 Case 1: Use of the Symbolic Math Toolbox on the Sinusoid Signal

Input: $x(t) = \frac{\sin(1394\pi t) + \sin(3266\pi t)}{2}$ Signal corresponding to dial tone of '1'

Moduli M = [5 7 11]

Output was calculated by using the CRT from the Symbolic Math toolbox in MATLAB. Fig. 6.1 shows the input and output signals. The RMS error obtained was 4.0626e^{-5}.

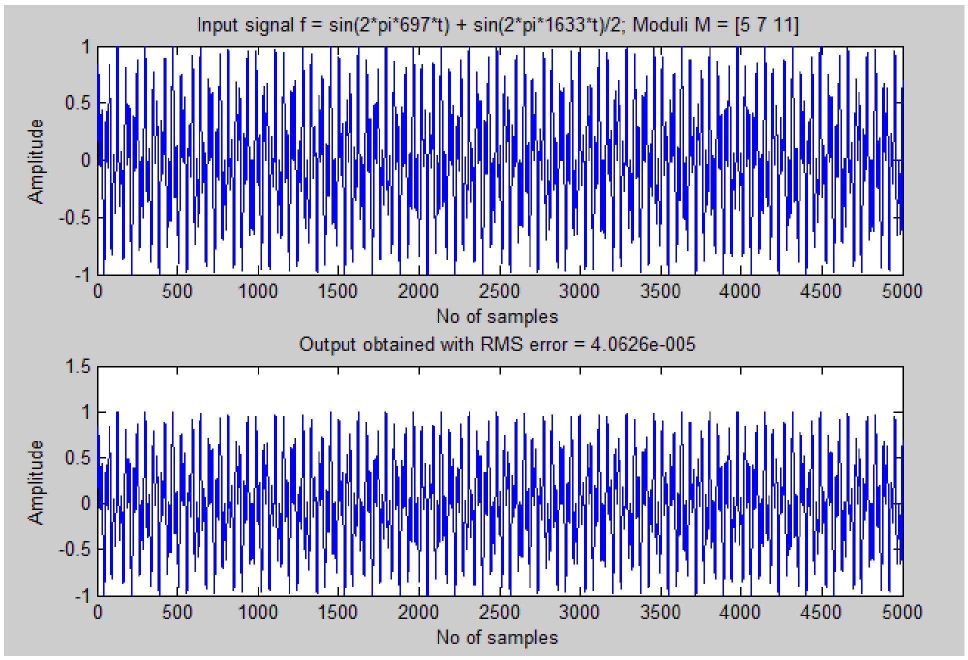


Fig. 6.1 Input and output figures for Case 1

Fig. 6.2 is the timing bar plot for case 1 above. As can be seen, out of total 23 seconds of run time, 75% of the time was consumed by the decoder section with 17 seconds. From our evaluation, it was understood that the time taken in decoder was in calls and switches between MATLAB and the Symbolic Math toolbox. To avoid these calls, a MATLAB code for CRT was written, and timing was evaluated with same signal in the next experiment.

With the RMS error as low as 4.0626e^{-5}, it was estimated as noiseless compression for all practical purposes.

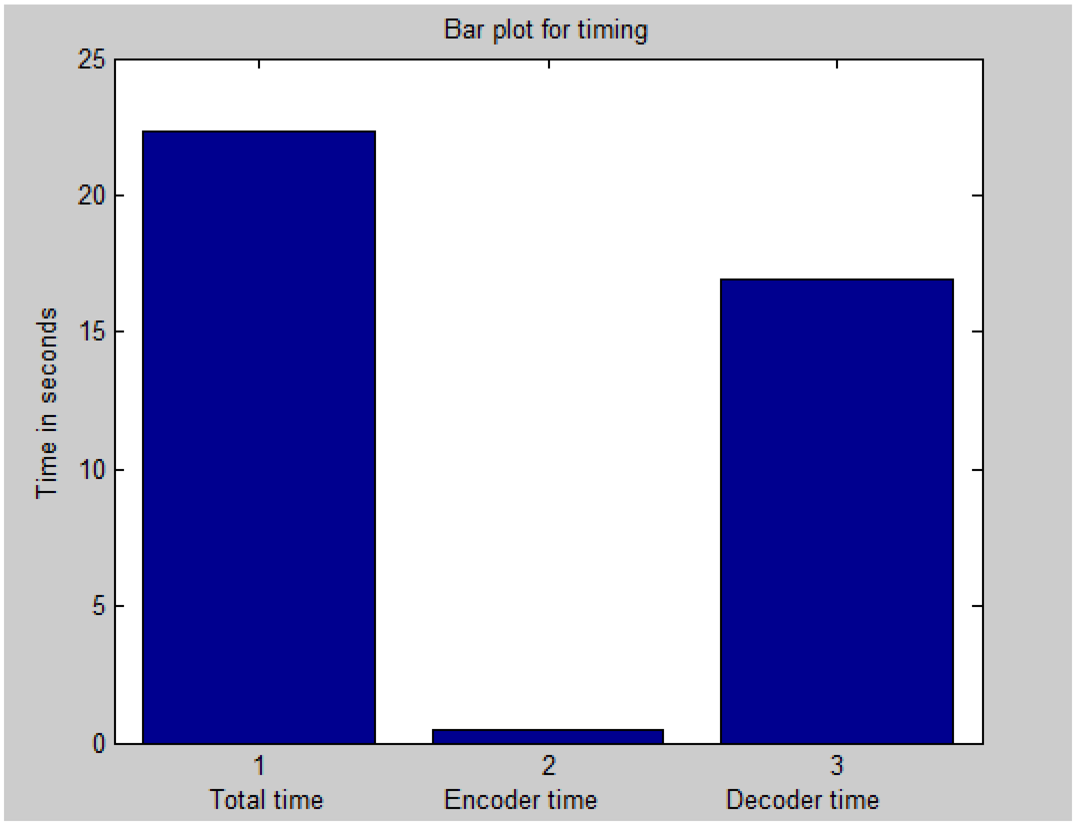


Fig. 6.2 Timing bar chart for Case 1

6.2 Case 2: Use of the CRT on the Sinusoid Signal

Input same as Case 1.

Moduli used, M = [7 31 127]

Programmed using the new CRT code, written in MATLAB

RMS error obtained = $2.9e^{-4}$

Fig. 6.3 shows the MATLAB figure for input and output signals.

We noticed the slight loss in precision in the decoded signal, probably due to the numerical approximations in the MATLAB code.

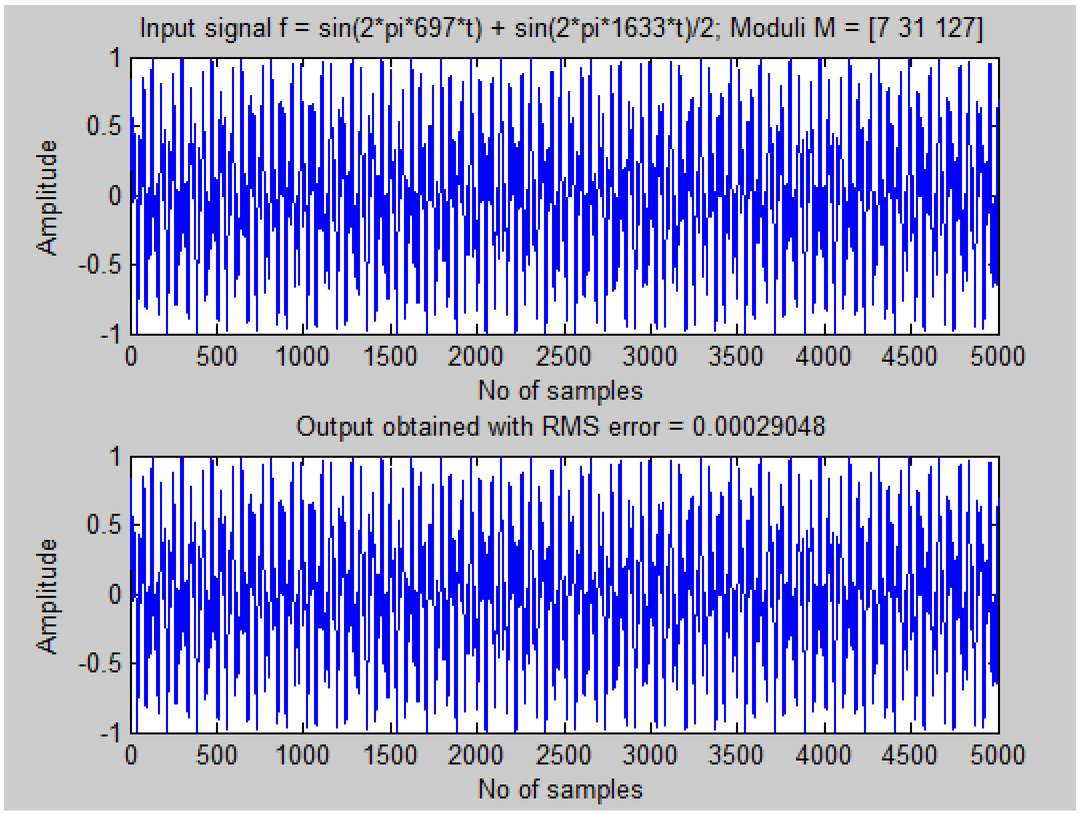


Fig. 6.3 Input and output figures for Case 2

Fig. 6.4 is the timing bar plot for case 2 discussed. As can be seen, the total time was reduced from 20 seconds to 1.65 seconds, proving that time taken by writing CRT code was better than using the Symbolic Math toolbox by 12 times. The decoder continued to use a major portion of the time, but it was no longer because of calls between MATLAB and the Symbolic Math toolbox. Instead, it was the solution of the system of congruences due to the time taken in calculating solutions using CRT for each signal and searching for the approximate answer. Hence came the decision to use the two-dimensional table. Before dealing with the two-dimensional table, case 3 presents the result of the same experiment with speech data.

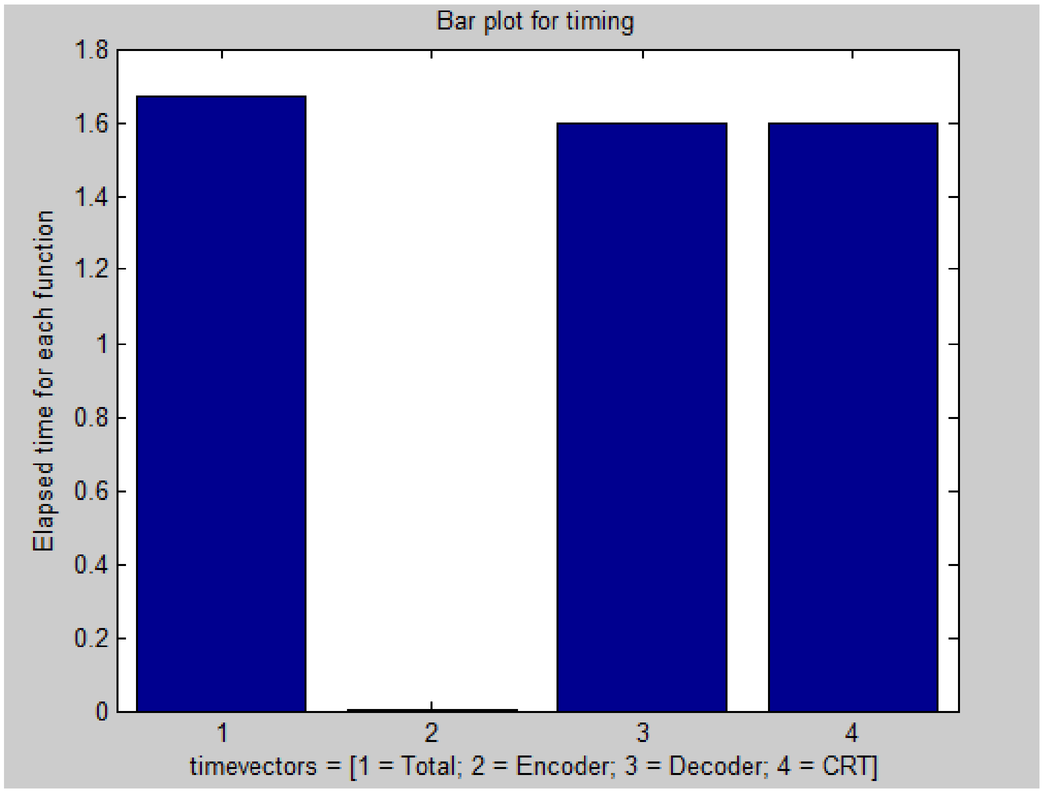


Fig. 6.4 Timing bar plot for Case 2

6.3 Case 3: Use of the CRT on the TIMIT Sentence

Input: The TIMIT sentence was downloaded from the TIMIT database.

Programmed using new CRT code, written in MATLAB

Moduli M = [7 31 127]

RMS error obtained = 2.766e^{-4} (see Appendix A for code)

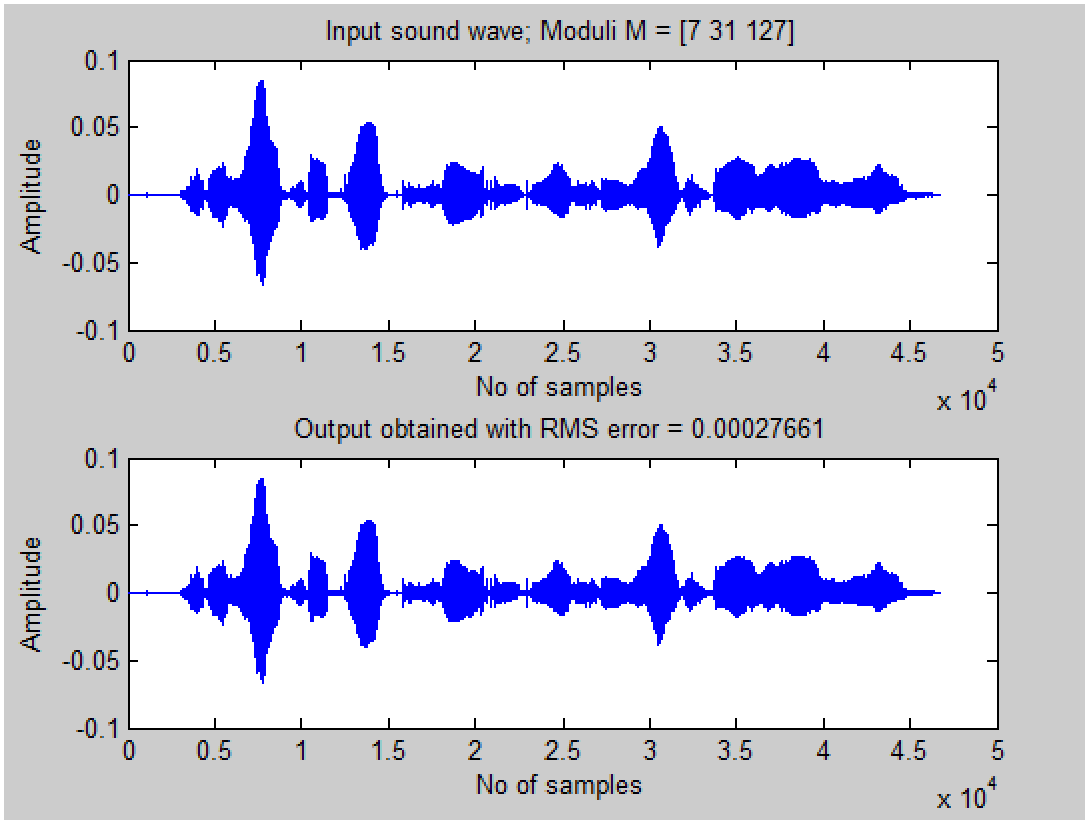


Fig. 6.5 Input and output figures with TIMIT sentence for Case 3

Fig. 6.6 shows the timing plot for Case 3 using the CRT MATLAB code as discussed. The total execution time was 220 seconds, out of which the decoder consumed most of the time. Though we saved on time by writing CRT code, the process wasn't fast enough to be used in a real-time environment. Case 4 presents the results obtained with the two-dimensional table.

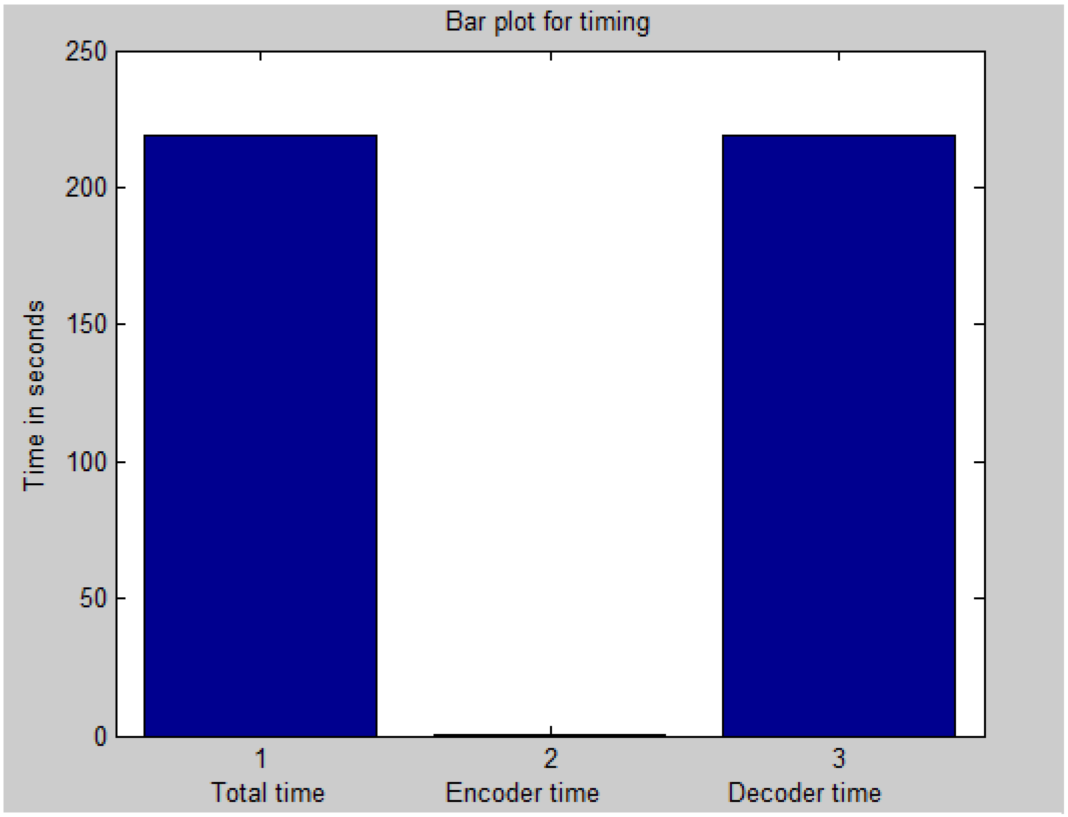


Fig. 6.6 Timing bar plot for Case 3

6.4 Case 4: Use of the Two-Dimensional Table with Three Prime Moduli

Input: The TIMIT sentence was downloaded from the TIMIT database.

Moduli M = [7 31 127]

A two-dimensional residue table (table style as in Table 4.1) was used in place of the CRT.

RMS error obtained = 2.766e^{-4}

Fig. 6.7 shows the input and output figures obtained during this experiment (see Appendix B for the code).

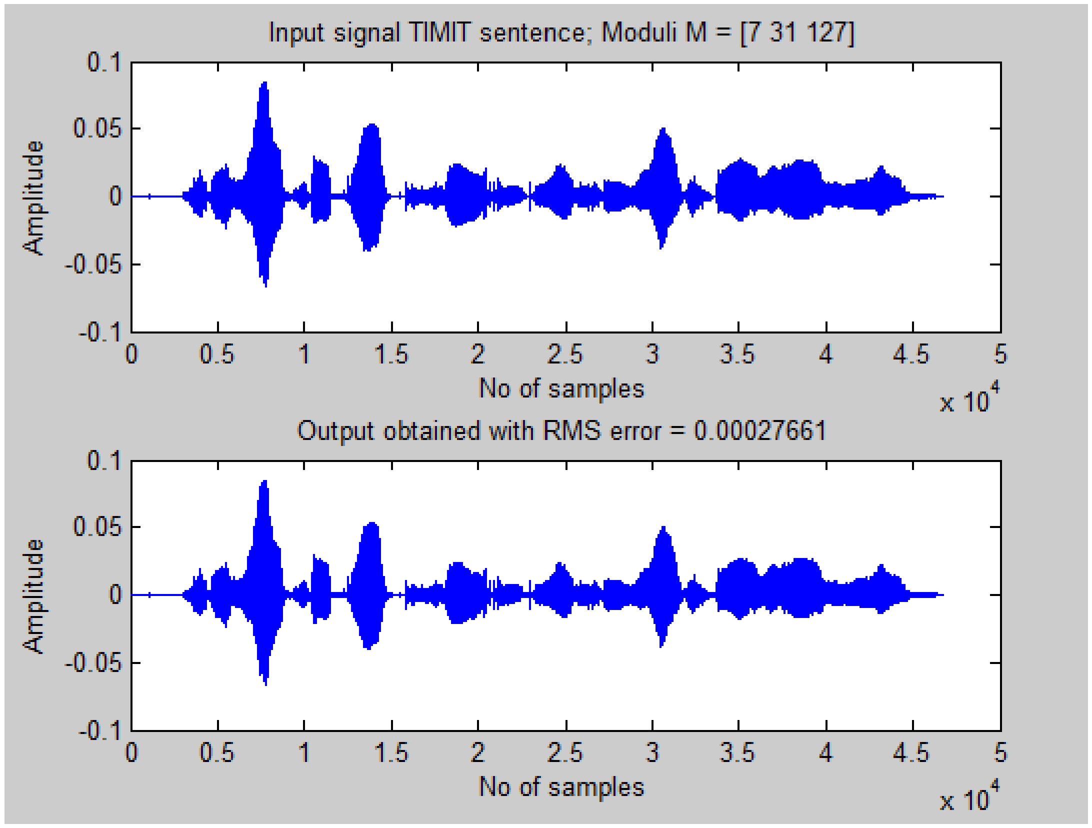


Fig. 6.7 Input and output figures with TIMIT sentence for Case 4

Fig. 6.8 shows the timing plot for Case 4. As can be seen, the execution times slashed down to 15 seconds from 220 seconds in Case 3. It shows that the idea of implementing a two-dimensional residue table makes the entire process 14 times faster than solving for CRT every single time.

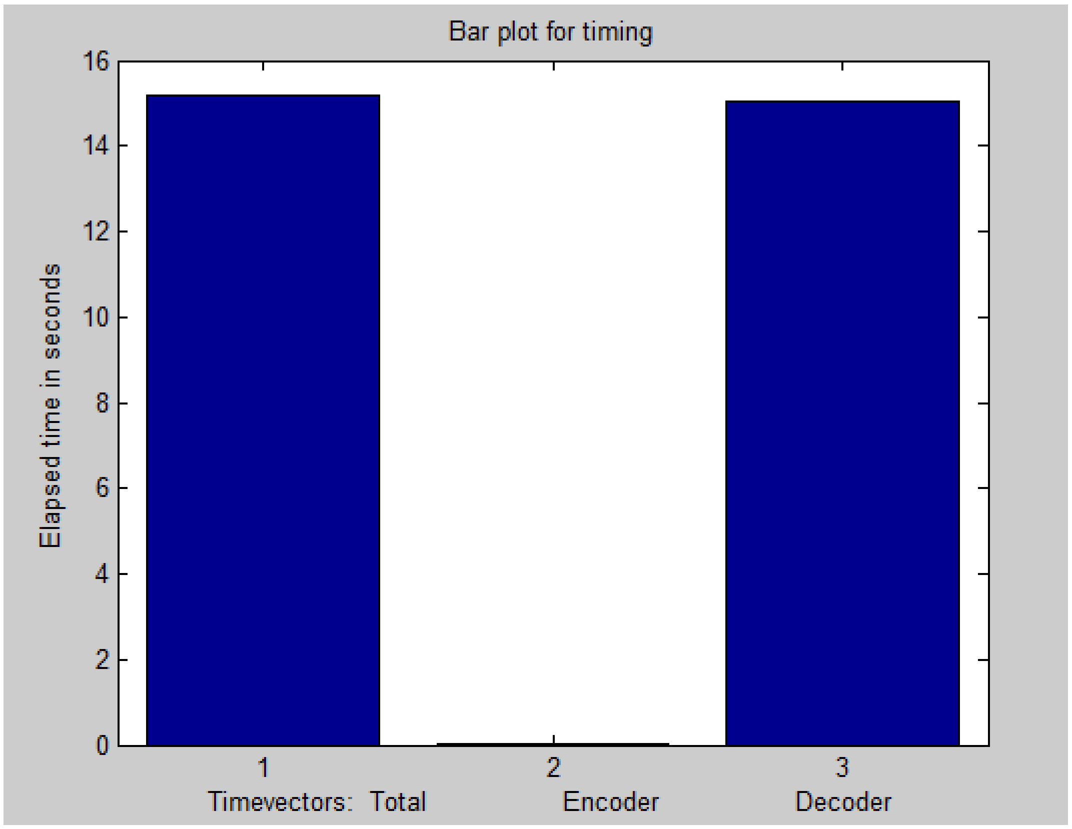


Fig. 6.8 Timing bar plot for Case 4

6.5 Case 5: Use of a Three-Dimensional Table with Three Prime Moduli

Input: The TIMIT sentence was downloaded from the TIMIT database.

Moduli M = [7 31 127]

The three-dimensional residue table was used in place of the two-dimensional table, thus eliminating the search process in the decoder section.

RMS error obtained = $2.766e^{-4}$

Fig. 6.9 shows the input and output figures obtained (see Appendix C for code).

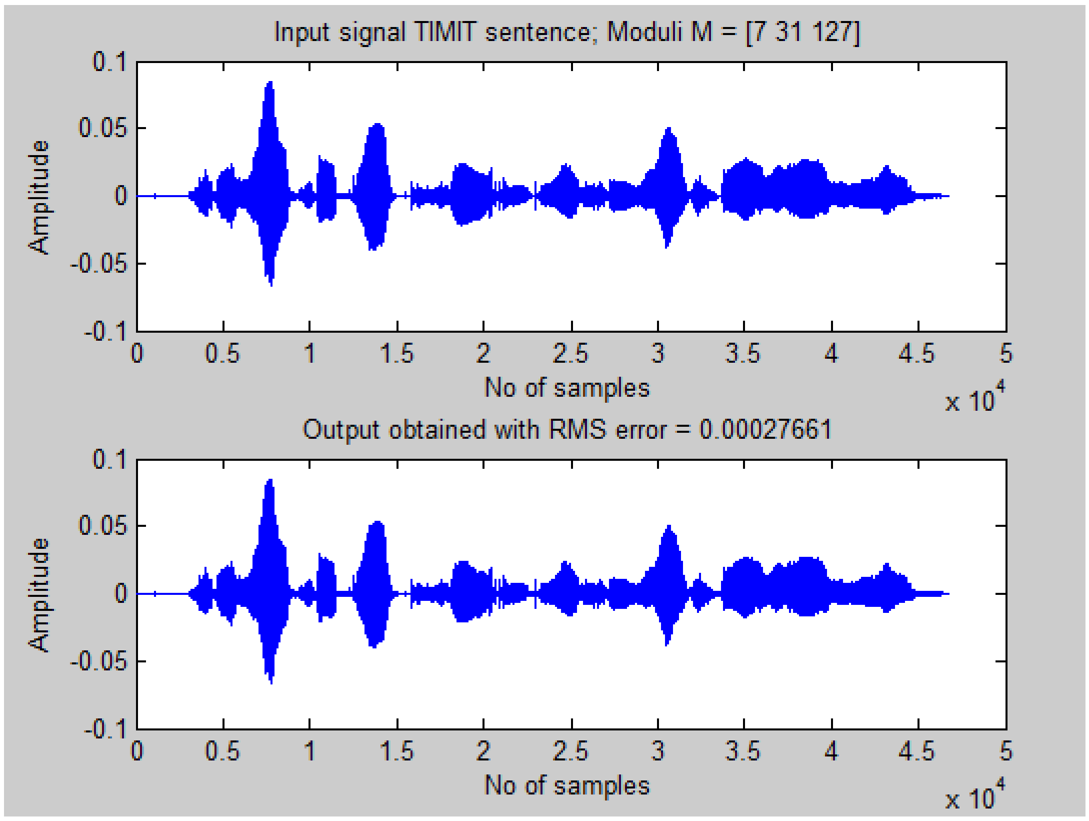


Fig. 6.9 Input and output figures with TIMIT sentence for Case 5

Fig. 6.10 shows a timing plot for Case 5. It can be inferred that the time consumed was reduced to 0.35 second while using the three-dimensional table in place of the two-dimensional table where total time consumed was 15 seconds. However, the speed of the entire process with three-dimensional table came with the additional requirement for memory, which could be really huge in cases of larger moduli.

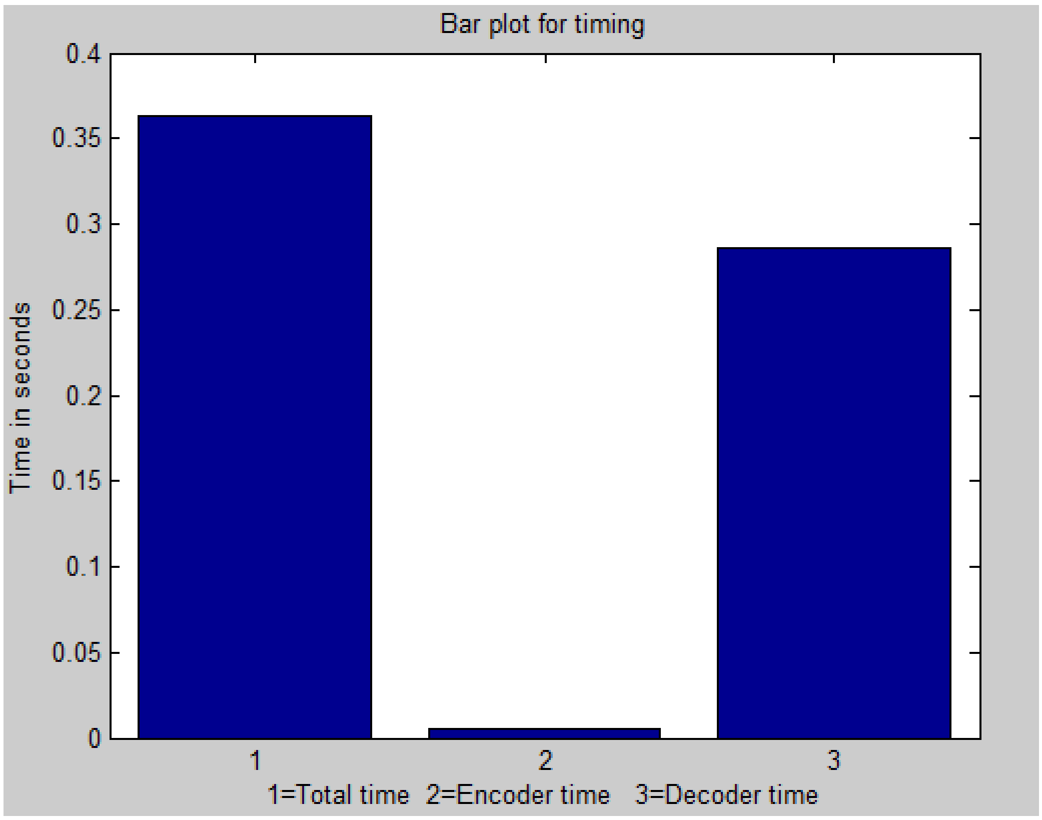


Fig. 6.10 Timing bar plot for Case 5

6.6 Case 6: Use of the Two-Dimensional Table with Two Prime Moduli

Input: The TIMIT sentence was downloaded from the TIMIT database.

Moduli M = [31 127]

A two-dimensional table (table style as in Table 4.1) was used with two moduli with the search process at the decoding end.

RMS error obtained = 2.7495e^{-4}

Fig. 6.11 shows the input and output figures obtained (see Appendix D for code).

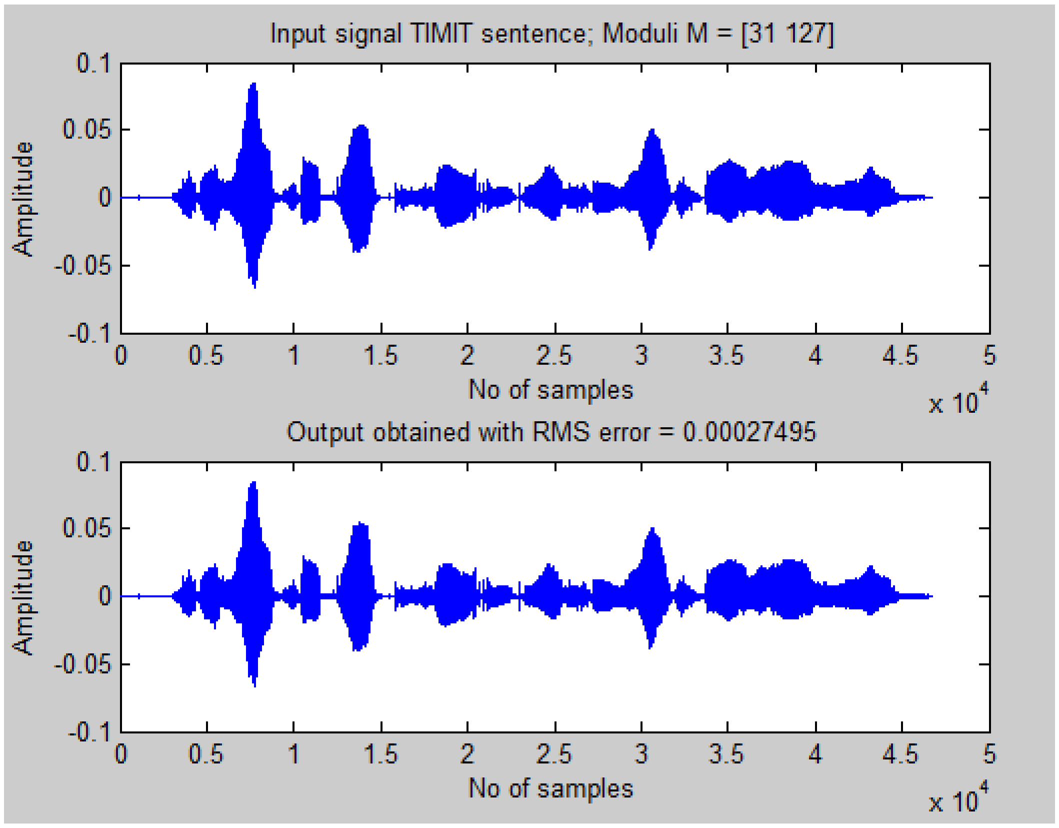


Fig. 6.11 Input and output figures with TIMIT sentence for Case 6

Fig. 6.12 shows timing plot for Case 6. It can be clearly read from the figure that the time consumed came down to 1.3 seconds while using two moduli instead of three and decoding using a two-dimensional table where total time consumed was 15 seconds. Thus, it was inferred that reducing the number of moduli would reduce the computations and hence the time consumed.

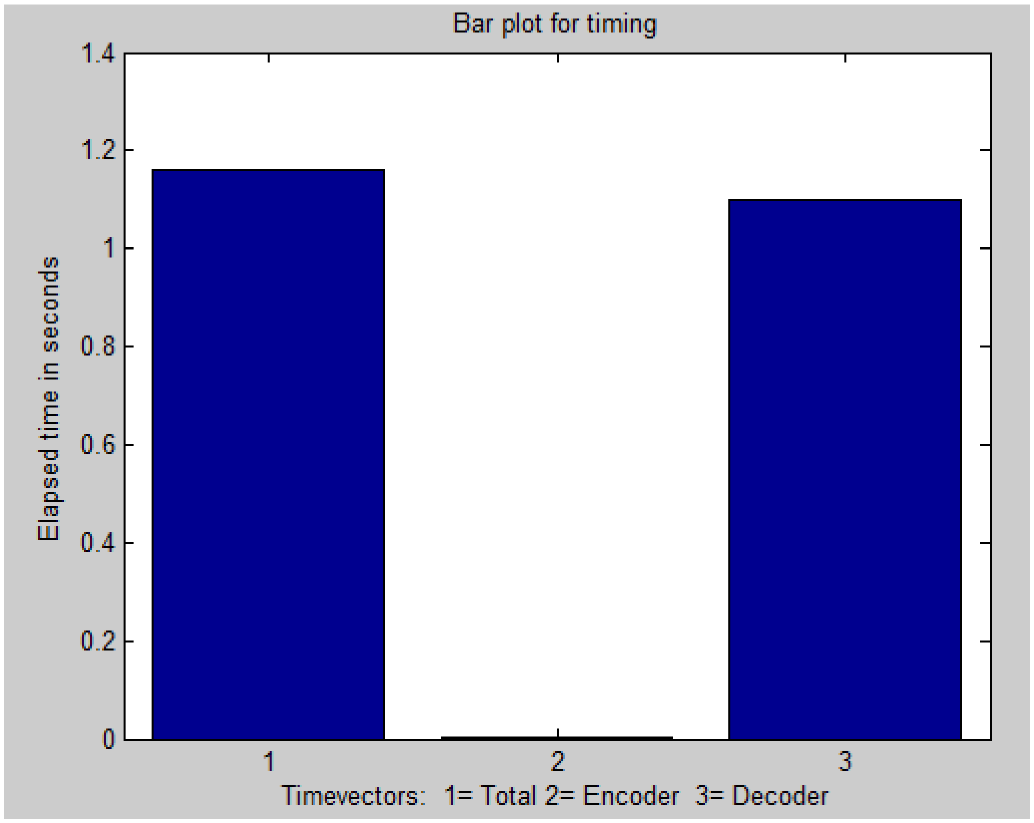


Fig. 6.12 Timing bar plot for Case 6

6.7 Case 7: Use of the Two-Dimensional Canonical Table

Input: The TIMIT sentence was downloaded from the TIMIT database.

Moduli M = [149 269]

A canonical form of the two-dimensional table (Table 4.4 style) was used with two moduli and a new decoding method that addresses the elements directly eliminating the search process.

RMS error obtained = 0

Fig. 6.13 shows the input and output figures obtained (see Appendix E for code).

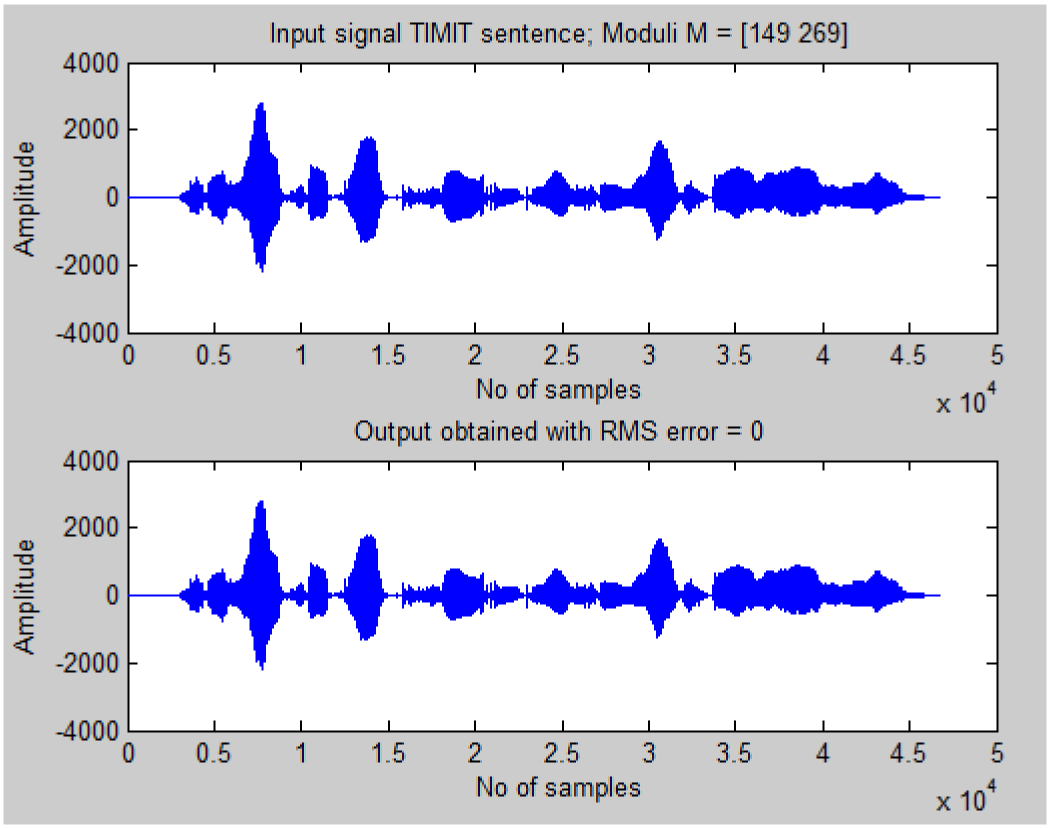


Fig. 6.13 Input and output figures with TIMIT sentence for Case 7

Fig. 6.14 shows a timing plot for Case 7. It can be seen that the time consumed was reduced to 0.082 second when using two moduli canonical table with a new decoding scheme instead of using a two-dimensional table with two moduli using search mechanism where the total time consumed was 1.1 seconds. It should be noticed that the decoder time no longer contributed to 75% of the total execution time.

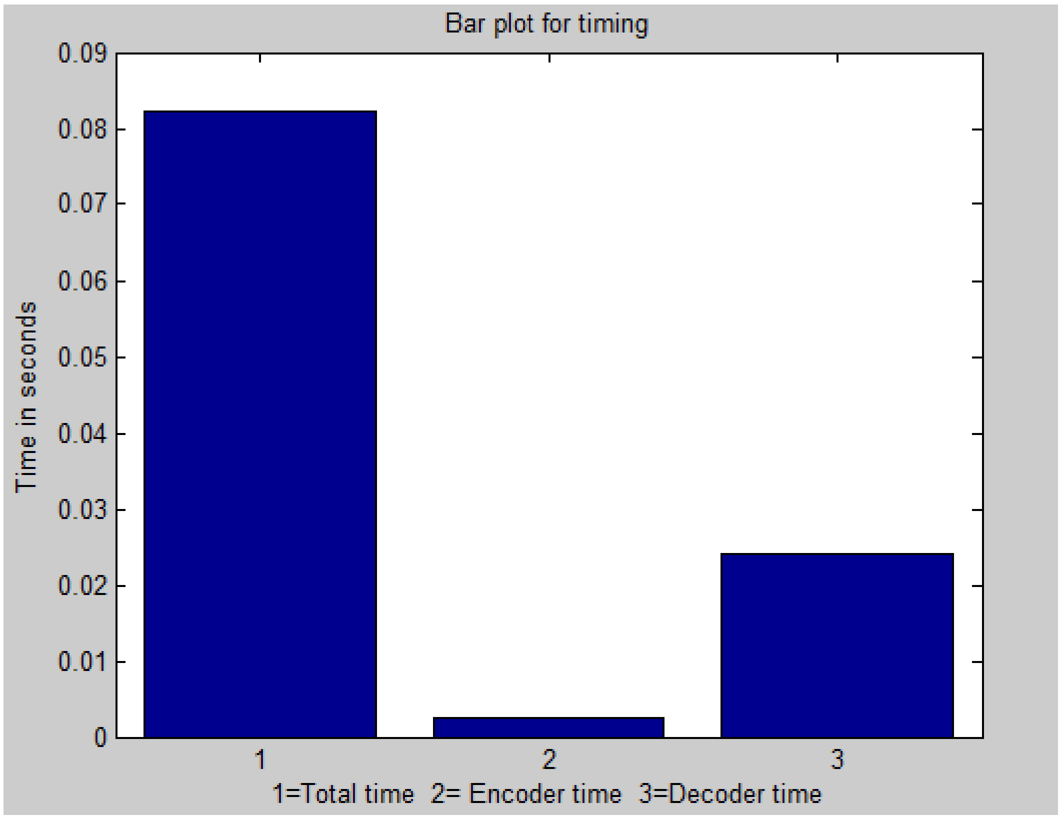


Fig. 6.14 Timing bar plot for Case 7

CHAPTER 7

CONCLUSIONS

As mentioned in Chapter 1, this work presented a novel technique for data compression using an MRS. The basic idea was a simpler and more general approach than solving an under-determined system of equations for data compression. By projecting the signal to be compressed into the MRS field, the system of equations was transformed into a system of congruences. The CRT was first used to solve this system. Other ideas about the solution of the Diophantine equations were analyzed but not reported here [40].

Having programmed in MATLAB, initially the Symbolic Math toolbox was utilized, which was later replaced by the MATLAB script for CRT. Considering the time restrictions the system might face while working in real-time applications, the two-dimensional residue table was introduced. It was later replaced by the three-dimensional table at the expense of more memory.

From the RMS error obtained in various experiments explained in Chapter 6, this method was found to be a lossless compression for all input.

As an extension to the work, replacing three moduli with two arbitrary relatively prime integers and thus reducing the number of computations was considered and successfully tested. This development not only reduced the number of computations but also the time aced and allowed greater choice in the solution on the moduli. To further eliminate the search process, a new decoding method that directly addresses the canonical residue table was used that eliminated the search process used in previous cases. This was considered to be the best method derived from this work due to its 0 RMS error and speed.

The results obtained in this work are summarized in Table 7.1.

Table 7.1: Summary of results

Case considered with input: a TIMIT sentence of 2.92-second duration	RMS error	Total execution time in seconds
Case 3: Using CRT script	$2.766e^{-4}$	225
Case 4: Using 2D table with search	$2.766e^{-4}$	15
Case 5: Using 3D table without search	$2.766e^{-4}$	0.37
Case 6: Using 2D table with a search for 2 moduli	$2.7495e^{-4}$	1.19
Case 7: Using canonical 2D table without a search for 2 moduli	0	0.082

The next level of achievement for this work would be in image and video compression. Having a video frame with a three-dimensional r ed-green-blue (RGB) data matrix, this work could be challenging in terms of managing the time for overall computation. It should be observed that in an audio-visual implementation, we would need a playback of the audio segment that corresponded to 30-frame/second video. The time of our playback is 0.026 second for the TIMIT sentence whose duration is 2.92 seconds. Therefore, the audio process should be developed more than adequately for an audio-visual application.

APPENDIX A

CASE 3 PROGRAM

```
%----------------------------------------------------
%-- MATLAB program for Case 3
%----------------------------------------------------

function Docv2

tstart = tic;

%     Input : timi_in.wav; Fs : 16k Hz
%reading .wav file which is TIMIT sentence "She had your dark suit in
%greasy wash water all year."
[y,fs]= wavread('timi_in');


%calculating peak to peak of input signal to lift the signal to make it
%positive
p2p = max(y)-min(y);
dat = round((y+p2p).*1000);

% Parameters for simulation
M = [127 31 7];

% Encoding
tenc_start = tic;
[A,ty_enc] = mr_enc(dat, M,tenc_start);

% Decoding
tdec_start = tic;
[dA, ty_Dec,ty_crt] = mr_decv1(dat, A, M,p2p,tdec_start);

dA = dA(:);
dA = (dA./1000)-p2p;

%--Error calculation
err = dA(2:end) - y(2:end);
err_rms = sqrt((sum(err.^2))/length(err));
RMS = sqrt((sum(dA.^2))/length(dA));
fprintf('\nRMS error between input and output signals is %f', err_rms);
fprintf('\n');

fprintf('\nRMS of obtained signal is %f', RMS);
fprintf('\n');

%----------plotting input and output signals
figure
subplot(2,1,1)
plot(y)
% title('Input sound wave')
title('Input sound wave; Moduli M = [7 31 127]');
xlabel('No of samples')
ylabel('Amplitude')
subplot(2,1,2)
plot(dA)
% title('Output sound wave obtained')
xlabel('No of samples')
ylabel('Amplitude')
title(['Output obtained with RMS error = ',num2str(err_rms)])
```

```
%writing the obtained signal to .wav file
% wavwrite(dA ,fs ,'timi_run1');

ty_total = toc(tstart);
timevectors = [ty_total ty_enc ty_Dec ty_crt]
figure
bar(timevectors)
title('Bar plot for timing')
xlabel('timevectors = [1 = Total; 2 = Encoder; 3 = Decoder; 4 = CRT]')
ylabel('Elapsed time for each function')



%Encoding function
function [A,ty_enc] = mr_enc(dat, M,tenc_start)

%----------------------------------------------------------------------%%
% Encoding for multi-residue system
%
% Input:
%       M: [p1 p2 p3] moduli with size of dM
%       dat: data for simulation, size of dLen
% Output:
%       A: size of dLen x M, residue matrix
%----------------------------------------------------------------------%%

dat = dat(:);

dM = length(M);

for id = 1 : dM-1
    A(:, id) = mod(dat, M(id));
end
ty_enc = toc(tenc_start);

%Decoding function
function [dA, ty_Dec,ty_crt] = mr_decv1(dat, A, M,~,tdec_start)

%----------------------------------------------------------------------%%
% Decoding for multi-residue system
%
% Input:
%       M: [p1 p2 p3] modulis with size of dM
%       dat: original data
%       A: size of dLen x dM, dLen is the length of data
% Output:
%       dA: size of dLen, the decoded for A
%----------------------------------------------------------------------%%

dM = length(M);
[dLen, col] = size(A);
% A = double(A);
dA(1) = dat(1);
% dA = double(dA);
% For loop for all the input data
tcrt_start = tic;
```

```
for id = 2 : dLen
    outTmp = [];
    for im = 0 : M(dM)-1
        res = [A(id, 1:dM-1) im];
        [outTmp(im+1),ty_crt] = crt(res, M,tcrt_start);
    end

    [~, Idx] = min(abs(outTmp - dA(id-1)));
    dA(id) = outTmp(Idx);
end

% dA = int16(dA);
ty_Dec = toc(tdec_start);

%Chinese remainder theorem
function [x,ty_crt] = crt(a,m,tcrt_start)
% This function solves the Chinese Remainder Theorem problem:
%   x= a(1) mod m(1)
%   x= a(2) mod m(2)
%   ...
%   x= a(r) mod m(r)
% The values for a and m should be a vector of the same dimension

if any(size(a) ~= size(m)),
   error('The vectors a and m should be the same size');
end;

r=length(a);

M=prod(m);  % calculate the total modulus

x=0;
for j=1:r
   temp = invmodn(M/m(j),m(j));
   x=x+ a(j)*(M/m(j))*temp;
   x=mod(x,M);
end;
ty_crt = toc(tcrt_start);

function y = invmodn( b,n)
% This function calculates the inverse of an element b mod n
% It uses the extended Euclidean algorithm

n0=n;
b0=b;
t0=0;
t=1;

q=floor(n0/b0);
r=n0-q*b0;
while r>0,
   temp=t0-q*t;
   if (temp >=0),
      temp=mod(temp,n);
   end;
   if (temp < 0),
      temp= n - ( mod(-temp,n));
   end;
   t0=t;
   t=temp;
   n0=b0;
   b0=r;
   q=floor(n0/b0);
   r=n0-q*b0;
end;

if b0 ~=1,
   y=[];
   disp('No inverse');
else
   y=mod(t,n);
end;
```

APPENDIX B

CASE 4 PROGRAM

```
%----------------------------------------------------
%-- MATLAB program for Case 4
%----------------------------------------------------

clear all
clc

% -------------Step 1: Defining moduli for compression--------------------
M = [7 31 127];

%------------Step 2: Generating 2D residue table for given moduli------------
--
k=M(1)*M(2);
t=k*M(3);
twoD(1,1)=1;
for i=2:M(3)
    twoD(i,1)=twoD(i-1,1)+k;
end
col=1;
for i=2:k
twoD(1,col+1)=mod((twoD(M(3),col)+1),t);
for j=2:M(3)
    twoD(j,i)=mod((twoD(j-1,i)+k), t);
end
col=col+1;
end

%Creating order matrix for decoding
mod_M1 = mod(1:M(1),M(1));
mod_M2 = mod(1:M(2),M(2));
for l = 1:M(1):M(2)*M(1)
    order(1,l:l+(M(1)-1)) = vertcat(mod_M1);
end
for l = 1:M(2):M(2)*M(1)
    order(2,l:l+(M(2)-1)) = vertcat(mod_M2);
end
order = order';

% Defining time vector to calculate time taken by process
tstart = tic;

% Step 3: Reading input: TIMIT sentence
y = wavread('timi_in.wav');

% -----------------Step 4: Augmenting the signal to the limit of product of
moduli-------------%
p2p = max(y)-min(y);
dat = round((y+p2p).*1000);

% --------------------Step 5: multi- residue coding of augmented signal------
-------------%
dat = dat(:);
dM = length(M);

%Initialising time vector for encoding section
tenc_start = tic;
for id = 1 : dM-1
```

```
    A(:, id) = mod(dat, M(id));
end
ty_enc = toc(tenc_start);

%----------- Step 6: Decoding the signal using 2D table and order matrix-----
---------%
dA = dat(1);
dLen = length(dat);
tdec_start = tic;
for id =2:dLen
    for i=1:length(order)
        l1 = order(i,:);
        l2 = A(id,:);
        if l1 == l2
            col_IDX = i;
        end
    end
    %Searching the nearest neighbour to previous sample
    [~,row_IDX] = min(abs(dA(id-1)-twoD(:,col_IDX)));
    dA(id) = twoD(row_IDX,col_IDX);
end
ty_Dec = toc(tdec_start);

% ------Step 7: Converting (augmented) decoded signal to limits of original
signal----%
dA = dA';
dA = (dA ./ (1000))- p2p;
ty_total = toc(tstart);

%-----------------Error calculation------------------%
e = dA(2:end)-y(2:end);
err_rms = sqrt((sum(e.^2))/length(e));
fprintf('\nrms is %f', err_rms);
fprintf('\n');

%-----------------------Fig.ures-----------------------%
%plotting input and output signals
subplot(2,1,1)
plot(y)
title('Input signal TIMIT sentence; Moduli M = [7 31 127]');
xlabel('No of samples')
ylabel('Amplitude')
subplot(2,1,2)
plot(dA)
xlabel('No of samples')
ylabel('Amplitude')
title(['Output obtained with RMS error = ',num2str(err_rms)])


timevectors = [ty_total ty_enc ty_Dec]
figure
bar(timevectors)
title('Bar plot for timing')
xlabel('Timevectors:  Total                 Encoder                 Decoder')
ylabel('Elapsed time in seconds')
%-----------MATLAB script for Case 4 in Chapter 6 ends--------------
```

APPENDIX C

CASE 5 PROGRAM

```
%----------------------------------------------------
%-- MATLAB program for Case 5
%----------------------------------------------------

clear all;
close all;
clc;

% Moduli for compression
M = [7 31 127];


% Step 1: Generate the 2-D table
for im1 = 1 : M(1)
    for im2 = 1 : M(2)
        mlist((im1-1)*M(2)+im2, :) = [im1-1 im2-1];
    end
end


% Using CRT for the 2D table
for ic = 1 : M(1)*M(2)
    for ir = 1 : M(3)
        ms2D(ir, ic) = crt([mlist(ic,:) ir-1], [M(1) M(2) M(3)]);
    end
end

% Step 2: Construct the 3-D table called ms3D
for ic = 1 : M(1)*M(2)
    for ir = 1 : M(3)
        % Locate the previous samples
        Pre_Sample = ms2D(ir, ic);

        for iz = 1 : M(1)*M(2)
            % Find M(3) possible values for the current sample
            Curr_Sample_Candi = ms2D(:, iz);

            % Find the distance between current and previous
            dist = abs(Pre_Sample - Curr_Sample_Candi);
            [dummy idx] = min(dist);

            ms3D(ir, ic, iz) = Curr_Sample_Candi(idx);
        end
    end
end
save Table3D ms3D;
load Table3D;

tstart = tic;

%reading .wav file which is TIMIT sentence "She had your dark suit in
%greasy wash water all year."
y = wavread('timi_in.wav');

%Calculating peak to peak of input signal to lift the signal to make it
%positive
p2p = max(y)-min(y);
```

```
dat = round((y+p2p).*1000);


% Encoding for multi-residue system%%
dat = dat(:);
dM = length(M);
tenc = tic;
for id = 1 : dM
    A(:, id) = mod(dat, M(id));
end
tencd= toc(tenc);

%Decoding for multi-residue system
tdec = tic;
yout(1) = dat(1);

prev_m1 = mod(yout(1), M(1));
prev_m2 = mod(yout(1), M(2));
prev_m1m2 = prev_m1*M(2) + prev_m2;
prev_m3 = mod(yout(1), M(3));
% decoding using direct decoding matrix
for id = 2 : length(A)

    curr_m1m2 = A(id, 1)*M(2) + A(id, 2);

    [prev_m1m2+1, prev_m3+1, curr_m1m2+1];
    % 3D table look up
    yout(id) = ms3D(prev_m3+1, prev_m1m2+1, curr_m1m2+1);

    % Update the memory
    prev_m1m2 = curr_m1m2;
    prev_m3   = mod(yout(id), M(3));

    if yout(id) ~= dat(id)
        fprintf('here');
    end

end
tdecd = toc(tdec);

% Converting yout (augmented) decoded signal its original magnitude
yout = (yout./1000)-p2p;
yout = yout(:);

tend =toc(tstart);

%Error calculation
e = yout(2:end)-y(2:end);
err_rms = sqrt((sum(e.^2))/length(e));
fprintf('\nrms is %f', err_rms);
fprintf('\n');

%-----------------------Figures-----------------------%
%plotting input and output signals
subplot(2,1,1)
plot(y)
title('Input signal TIMIT sentence; Moduli M = [7 31 127]');
```

```
xlabel('No of samples')
ylabel('Amplitude')
subplot(2,1,2)
plot(yout)
xlabel('No of samples')
ylabel('Amplitude')
title(['Output obtained with RMS error = ',num2str(err_rms)])

t = [tend tencd tdecd];
figure;
bar(t)
title('Bar plot for timing')
ylabel('Time in seconds')
xlabel('1=Total time  2=Encoder time   3=Decoder time')
```

APPENDIX D

CASE 6 PROGRAM

```
%----------------------------------------------------
%-- MATLAB program for Case 6
%----------------------------------------------------

clear all
close all
clc

%--Contrusting two dimensional table for given 2 moduli matrix M
M = [31 127];

M1=M(1)*M(2);
twoD(1,1)=0;
for i=2:M(1)
twoD(i,1)=mod(twoD(i-1,1)+M(2),M1);
end
col=1;
for i=2:M(2)
twoD(1,col+1)=mod((twoD(M(1),col)+1),M1);
for j=2:M(1)
twoD(j,i)=mod((twoD(j-1,i)+M(2)), M1);
end
col=col+1;
end
twoD = int16(twoD);

tstart = tic;

%---Reading input: TIMIT sentence
y = wavread('timi_in.wav');
p2p = abs(min(y));
dat = round((y + p2p).*1000 );

%---Encoding--------
tenc_start = tic;
A = mod(dat, M(2));
ty_enc = toc(tenc_start);

%----Decoding-------
tdec_start = tic;
dLen = length(A);
y_out(1) = dat(1);

for id =2:dLen
    col_IDX(id) = A(id)+1;
   [~,row_IDX(id)] = min(abs(y_out(id-1)-twoD(:,col_IDX(id))));
    y_out(id) = twoD(row_IDX(id),col_IDX(id));
end
ty_Dec = toc(tdec_start);

% Converting (augmented) decoded signal to limits of original signal
y_out = (y_out./1000) - p2p;
ty_total = toc(tstart);

%Error calculation
y_out = y_out(:);
e = y_out(2:end)-y(2:end);
```

```
err_rms = sqrt((sum(e.^2))/length(e));
fprintf('\nrms is %f', err_rms);
fprintf('\n');

%------------------------Figures------------------------%
%---Plotting input and output signals
subplot(2,1,1)
plot(y)
title('Input signal TIMIT sentence; Moduli M = [31 127]');
xlabel('No of samples')
ylabel('Amplitude')
subplot(2,1,2)
plot(y_out)
xlabel('No of samples')
ylabel('Amplitude')
title(['Output obtained with RMS error = ',num2str(err_rms)])


timevectors = [ty_total ty_enc ty_Dec]
figure
bar(timevectors)
title('Bar plot for timing')
xlabel('Timevectors:  1= Total 2= Encoder  3= Decoder')
ylabel('Elapsed time in seconds')
```

APPENDIX E

CASE 7 PROGRAM

```
%----------------------------------------------------
%-- MATLAB program for Case 7
%----------------------------------------------------

clear all
close all
clc
%----Creating 2D table---------
M = [236 269];

Table=zeros(M(1),M(2));
start=0;
for i=1:M(1)
    for j=1:M(2)
        Table(i,j)=start;
        start=start+1;
    end
end

tstart = tic;
%-------Reading sound file----------
f1=wavread('timi_in.wav','native');
[m,n]=size(f1);

%Lifting the signal and making it positive
p2p = abs(min(f1));
f2 = (f1 + p2p);


Isend = int16(zeros(m,n));
a = int16(zeros(m,n));
Isave = int16(zeros(m,n));

%---Encoding-----
tenc = tic;
a = mod(f2(:,:,1),M(2))+1;
Isave = a;

Isend=((f2(:,:,1)-a)./M(2))+1;
tencd= toc(tenc);

%---Decoding--------
tdec = tic;

    for j=1:n
        for i=1:m
            r(i,j)=Table(Isend(i,j),Isave(i,j));
        end
    end

tdecd = toc(tdec);

r=int16(r);
r1 = r- p2p;
```

```
tend =toc(tstart);
%-------------Error calculation---------
e = f1 - r1;
err_rms = sqrt((sum(e.^2))/length(e));
fprintf('\nrms is %f', err_rms);
fprintf('\n');

%------------Plotting--------------
figure
subplot(2,1,1)
plot(f1)
title('Input signal TIMIT sentence; Moduli M = [236 269]');
xlabel('No of samples')
ylabel('Amplitude')
subplot(2,1,2)
plot(r1)
xlabel('No of samples')
ylabel('Amplitude')
title(['Output obtained with RMS error = ',num2str(err_rms)])


t = [tend tencd tdecd];
figure;
bar(t)
title('Bar plot for timing')
ylabel('Time in seconds')
xlabel('1=Total time  2= Encoder time  3=Decoder time')
```

BIBLIOGRAPHY

[1] Oscar N Garcia, "Error Codes for Arithmetic and Logical Operations," College Park, MD, Doctoral Dissertation 1969.

[2] Harvey L Garner, "Error Codes for Arithmetic Operations," *IEEE Transactions on Electronic computers*, vol. EC-15, no. 5, pp. 763 - 770, 1966.

[3] F.J Taylor, "Residue Arithmetic: A Tutorial with Examples," *Computer*, vol. 17, no. 5, pp. 50-62, May 1984.

[4] C.F Gauss, *Disquisitiones arithmeticae*, 1st ed.: Yale University Press, 1966.

[5] H S Shapiro, "Some Remarks on Modular Arithmetic and Parallel Computation," *Mathematics of Computation* , vol. 16, pp. 218-222, 1962.

[6] ROHAN Academic Computing. [Online]. http://www-rohan.sdsu.edu/doc/matlab/toolbox/comm/galois.html

[7] Wikipedia: The Free Encyclopedia. [Online]. http://en.wikipedia.org/wiki/Trigonometry_in_Galois_fields

[8] Harvey L Garner, "The residue number system," *IRE Transactions on Electronic Computers*, vol. EC -8, no. 2, pp. 140 - 147, 1959.

[9] Yuke Wang, "New Chinese Remainder theorems," in *Conference on Signals, Systems & computers Conference Record of the Thirty- Second Asilomar*, vol. 1, 1998, pp. 165-171.

[10] Shenghui Su, "To Solve the High Degree Congruence," in *Proceedings of the 2007 International Conference on Computational Intelligence and Security*, Washington, DC, 2007, pp. 672-676.

[11] Subhash Kak, "Computational Aspects of Aryabhatta Algorithm," *Indian Journal of History of science*, vol. 21, no. 1, pp. 62-71, 1986.

[12] Sreeram Vuppula, "The Aryabhatta Algorithm Using Least Absolute Remainders," *CoRR*, vol. 0604012, 2006.

[13] C.E Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379-423 and 623-656, July and October 1948.

[14] imatest- Digital Image Quality. [Online]. http://www.imatest.com/docs/shannon/

[15] D. Gerlic, "The Inverse of a Block-Circulant Matrix," *IEEE Transactions on Antenna and Propagation*, vol. 31, no. 5, pp. 808 - 810, September 1983.

[16] Arash Amini, "Deterministic Construction of Binary, Bipolar and Ternary Compressed Sensing matrices," *IEEE Transactions on Information Theory*, vol. 57, no. 4, pp. 2360 - 2370, 2011.

[17] Rick Chartnard, "Exact Reconstruction of Sparse Signals via Nonconvex Minimization," *IEEE Signal Processing Letters*, vol. 14, no. 10, pp. 707 - 710, October 2007.

[18] Richard G. Baraniuk, "Compressive Sensing," *IEEE Signal Processing Magazine*, vol. 24, no. 4, pp. 118-124, July 2007.

[19] John Edwards, "Focus on Compressive sensing," *IEEE Signal Processing Magazine*, vol. 28, no. 2, pp. 11-13, March 2011.

[20] Vivek Goyal, "The Optimum Bayesian: Replica method of Compressed Sensing," in *Illinois/Missouri Applied Harmonic Analysis Seminar*, 2010.

[21] Emmanuel J. Candes, "Decoding by Linear Programming," *IEEE Transactions on Information Theory*, vol. 51, no. 12, pp. 4203 - 4215, December 2005.

[22] W.Edwin Clark, "On Arithmetic Weight for a General Radix Representation of Integers," *IEEE Transactions on Information Theory*, vol. 19, no. 6, pp. 823 - 826, November 1973.

[23] Marco F. Duarte, "Sparse Signal Detection From Incoherenct Projections," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)- III*, pp. 305--308, 2006.

[24] Juan M. Restrepo. (2001, July) Eigen Values and the Canonical Forms of Matrices. [Online]. http://www.physics.arizona.edu/~restrepo/475A/Notes/sourcea/node56.html

[25] Juan M. Restrepo. (2001, July) [Online]. http://www.physics.arizona.edu/~restrepo/475A/Notes/sourcea/sourcea.html

[26] Justin Romberg, "Compressed Sensing: A Tutorial," in *IEEE Statistical Signal Processing Workshop*, 2007.

[27] Albert Cohen, "Compressed Sensing and Best k-term Approximation," *Journal of the American Mathematical Society*, vol. 22, pp. 211-231, 2009.

[28] Cleve Moler, "Cleve's Corner- "Magic" Reconstruction: Compressed Sensing," *MathWorks*

News & Notes, 2010.

[29] JH Jordan, "Complete Residue Systems in the Gaussian Integers," *Math Magazine*, vol. 38, pp. 1-12, 1965.

[30] Oscar N Garcia and T R N Rao, "Cyclic and Multiresidue Codes for Arithmetic Operations," *IEEE Transactions on Information Theory*, vol. 17, no. 1, pp. 85-91, January 1971.

[31] Oscar N Garcia and J L Massey, "Error Correcting Codes for Computer Arithmetic," *Advances in Information Systems Science*, vol. IV, pp. 273-326, 1972.

[32] Riccardo Masiero, "Data Acquisition Through Joint Compressive Sensing and Principal Component Analysis," in *GLOBECOM 2009, IEEE Global Telecommunications Conference*, Honolulu, HI, 2009, pp. 1 - 6.

[33] Ronald A DeVore, "Deterministic Constructions of Compressed Sensing Matrices," *Science Direct- Journal of Complexity*, vol. 23, no. 4-6, pp. 918 – 925, August 2007.

[34] Edinburgh Compressed Sensing Group. [Online]. http://ecos.maths.ed.ac.uk/ric_bounds.shtml

[35] Olga V Holtz. (2009, January) An Introduction to Compressive Sensing. [Online]. http://www.cs.berkeley.edu/~oholtz/Talks/CS.pdf

[36] Wei Dai, "Weighted Superimposed Codes and Constrained Integer Compressed Sensing," *IEEE Transactions on Information Theory*, vol. 55, no. 5, pp. 2215 - 2229, May 2009.

[37] Wikipedia: The Free Encyclopedia. [Online]. http://en.wikipedia.org/wiki/Compressed_sensing

[38] W.Edwin Clark, "On Modular Weight and Cyclic Nonadjacent Forms for Arithmetic Codes," *IEEE Transactions on Information Theory*, vol. 20, no. 6, pp. 767 - 770, November 1974.

[39] W.Edwin Clark, "Equidistant Binary Arithmetic Codes," *IEEE Transactions on Information Theory*, vol. 32, no. 1, pp. 106 - 108, January 1986.

[40] Keith Conrad. (2009, Fall) Gaussian integers. [Online]. http://www.math.uconn.edu/~kconrad/blurbs/ugradnumthy/Zinotes.pdf