FPGA BASED HARDWARE DESIGN OF SIMILARITY SEARCH ALGORITHMS FOR TIME SERIES PROCESSING APPLICATIONS

A

Thesis

Submitted in partial fulfillment of the requirements

for the award of the degree of

Master of Technology

in

VLSI Design and Embedded Systems

Submitted By

Ms. Kriti Suneja

University Roll No: 2K13/VLSI/10

Under the Guidance of

Dr. Malti Bansal

(Assistant Professor)

Department of Electronics and Communication Engineering

Delhi Technological University, Delhi

Department of Electronics & Communication Engineering

Delhi Technological University, Delhi-42

2013-2015

Delhi Technological University

Delhi

CERTIFICATE

This is to certify that the dissertation entitled "**FPGA based Hardware Design of Similarity Search Algorithms for Time Series Processing Applications**" is a bonafide work of **Kriti Suneja** (University Roll No. 2K13/VLSI/10), a student of Delhi Technological University. This project was carried out under my direct supervision and guidance and forms a part of the Master of Technology Course in **Electronics and Communication Engineering** with specialization in "**VLSI Design and Embedded Systems**" at Delhi Technological University, Delhi.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University/ Institute for the award of any Degree or Diploma.

Date:

Dr. Malti Bansal

(Assistant Professor)

Dept. of Electronics and Communication Engg.

Delhi Technological University

Department of Electronics and Communication Engineering

Delhi Technological University

(Formerly **Delhi College of Engineering)**

Bawana Road, Delhi- 110042

CANDIDATE'S DECLARATION

I, **Kriti Suneja**, Roll No. **2k13/VLSI/10**, student of **M.Tech (VLSI Design and Embedded Systems)**, hereby declare that the dissertation entitled " **FPGA based Hardware Design of Similarity Search Algorithms for Time Series Processing Applications**", under the supervision of **Dr. Malti Bansal**, Assistant Professor, Electronics and Communication Engineering Department, Delhi Technological University, in partial fulfillment of the requirements for the award of the degree of Master of Technology in VLSI Design and Embedded Systems, has not been submitted elsewhere for the award of any other degree or diploma.

I hereby solemnly and sincerely affirm that all the particulars stated above by me are true and correct to the best of my knowledge and belief.

Place: **Delhi** **Kriti Suneja**

Date: **2k13/VLSI/10**

ACKNOWLEDGEMENT

I express my sincere gratitude to my project supervisor **Dr. Malti Bansal** for her valuable guidance and support throughout the entire project work. I appreciate her tolerance and thank her for her time spent on advising me and evaluating my work.

I would also like to extend my heartfelt gratitude to **Prof. P. R. Chadha**, Head of the department and all faculty members of Electronics and Communication Engineering Department, Delhi Technological University, for their generous help in various ways for the completion of this thesis.

I would like to thank all my friends and especially my classmates for all the thoughtful and mind stimulating discussions we had, which prompted me to think beyond the obvious. I've enjoyed their companionship so much during my stay at DTU, Delhi.

Finally, to make this acknowledgement complete, a special thank to my parents for their unconditional love and support. The immense sacrifices made by my parents for the education of their children are incomparable. I can never thank them enough for all they have done for me. One extra line of appreciation for my mother for being a great friend, philosopher and guide, this added an essence to my work.

Kriti Suneja

University Roll no: **2K13/VLSI/10**

M.Tech (VLSI and Embedded System Design)

Department of Electronics & Communication Engineering

Delhi Technological University

Delhi – 110042

ABSTRACT

Time series is a huge collection of data indexed sequentially with respect to time. It is being produced at an extremely high rate from almost every domain including stock market, music industry, biomedical industry, etc. Data mining from temporal database requires similarity measures which can distinguish between two or more time series. Many distance functions, such as Dynamic Time Warping, Edit distance on Real Sequences, Move Split Merge, etc., which work efficiently in software for retrieval of similarity in temporal sequences exist.

Since the time series is a massive dataset, features need to be extracted before its analysis.

In this work, synthesis of five similarity measures has been done using the device xc3s400-4-pq208 in Xilinx with Verilog Hardware Description Language (HDL) and a comparison has been made to show the outperformance of one over the other based on the critical parameters of hardware utilization and delay. The purpose behind this project is to make these similarity measures available as portable devices for time series analysis in various domains. Simulations were performed in ModelSim.

To compare the efficacy of these similarity measures in distinguishing the time series, an application of detection of plagiarism in music has been implemented in MATLAB, where all the five algorithms were used to compute distance between plagiarized, unplagiarized, and same pair of songs. Algorithm which could clearly distinguish these three sets of data, as well as performed fairly well in hardware performance, was given the highest score to be used as a separate entity in real time applications.

Also, a comparison was made between the execution time in hardware and software to ensure the speed up of FPGA based implemented algorithms over software. The results showed that while hardware implemented DTW can attain the highest frequency of 18.9 MHz, it is only 9.6 KHz for MATLAB implemented DTW for four element length sequence.

Obtained results suggest that DTW was best for plagiarism detection and LCSS stood second. However, LCSS performed best in hardware utilization and delay. Thus, it is a bestfit algorithm for commercial use.

REFEREED PUBLICATIONS ARISING FROM THIS THESIS

[1] **K. Suneja** and M. Bansal, "Hardware design of similarity measures for time series based on FPGA in Verilog", proc. of 2^{nd} international conference on VLSI, Communication and Networks (VCAN), April 2015.

[2] **K. Suneja** and M. Bansal "Hardware design of Dynamic Time Warping algorithm based on FPGA in Verilog," International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE), vol.4, issue 2, pp. 165-168, February 2015.

[3] **K. Suneja** and M. Bansal "Comparison of Time Series Similarity Measures for Plagiarism Detection in Music" submitted for review.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
CLB	Configurable Logic Block
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DNA	Deoxyribonucleic acid
DTW	Dynamic Time Warping
ECG	Electrocardiograph
EDR	Edit Distance on Real Sequences
EEG	Electroencephalograms
ERP	Edit Distance with Real Penalty
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IOB	Input Output Block
ISE	Integrated Software Environment
LCSS	Longest Common Subsequence
LUT	Look Up Table
MFCC	Mel Frequency Cepstral Coefficient
MIR	Music Information Retrieval
MSM	Move Split Merge
PC	Personal Computer
RAM	Random Access Memory
SRAM	Static Random Access Memory
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

TABLE OF CONTENTS

1 Introduction

1.1 Motivation

25th April, 2015 was a devastating day for Nepal as it was hit by a major Earthquake. Scientists, soon after the quake, analyzed the seismic waves' data and compared it with the past records. They, then concluded that the entire subcontinent of India is moving northward by 1.8 inches/ year. This was possible only after studying the seismic data, which is nothing, but a time series.

Human beings have learnt a lot from birds and animals. Birds also talk to each other, though not in human languages. They produce sounds to allure their mates, to warn others of dangers, etc. Understanding their vocalization requires extraction of acoustic features and doing time series analysis on them which can aid to the human understanding of various species of nature. This requires searching of large temporal database against a query sequence.

The starters at gym do not know how hard to push themselves, especially the heart patients. A heart beat tracker in the form of a small device can keep a track of their heart beats and can help them watch out on their progress. This requires a device which can record and compare the time series data over an interval of time. So, in this project, we have implemented similarity measures for time series on FPGA, so that they can be used as prototypes for their ASIC implementation, which in turn, can be commercialized as portable devices.

Time series data generally emerge in a large variety of applications, such as, scientific domain (Weather activities), medical domain (Electrocardiographs (ECGs), Electroencephalograms (EEGs)), finance domain (stock market data), music industry, etc. Last decade has witnessed an increasing interest in mining time series database. The analysis of time series data generally requires a comparison to be made between two time series. This, in turn, requires similarity/ distance measures which show the extent of similarity between two time series.

For the efficient computation of similarity, the distance measures should be adequate, simple and expressive. Besides, they should be able to incorporate the different lengths, different speeds, anomalies in the time series to be compared. There are few existing similarity measures which are being used for time series analysis.

1.2 Objective and problem statement

- To study the existing similarity measures for time series
- To study the features of music
- To compare the most important similarity measures for their efficacy in detection of plagiarism in music and show that they are feasible, as well as, practical.
- To study the hardware implementation of software algorithms.
- To show that the hardware implementation of similarity measures is feasible and practical through a synthesizable FPGA implementation and comparison of timing and hardware utilization.
- To implement the similarity measure algorithms on FPGA and propose the best fit algorithm for both software and hardware applications.

Software algorithms, if implemented on hardware, can be used in real time applications. Time series is one of the most widely searched database. A time series T is mathematically defined as:

T= $(\{a_1,t_1\}, \{a_2,t_2\},......,\{a_n,t_n\})$, where $a_1,a_2,...,a_n$ are the values (amplitude, position, energy, power, temperature, etc.) of the physical quantities at time samples $t_1,t_2,....,t_n$ respectively. Most of the queries in time series database are based on similarity search. Since the similarity requires only the values of the quantities, irrespective of their sampling rates, time dimension can be removed from the series, however, the sequence has to be kept undisturbed .We, in this project, pursue the study of various similarity search algorithms for time series. A real time application of Music industry, i.e., to detect if the two songs have been copied from one another is implemented in MATLAB to compare the efficacy of five similarity measures in plagiarism detection. The main focus of the project is to confirm if these algorithms can be implemented in hardware, so that they can be used as coprocessors in real time systems, for example, in voice password system.

1.3 Previous Work

In our daily lives, we deal with most of the data in the form of time series. In last few years, many similarity measures have been recommended to measure the distance between time series. The very first distance function which was proposed for similarity search of time series data was Euclidean Distance [1]. However, soon after its introduction, need for a better distance function was felt as it didn't go well with the local time shifting.

As a result, Dynamic Time Warping (DTW), a benchmark algorithm in speech processing, was introduced to handle acceleration/deceleration. DTW is very efficient in software computing. Guo Yuying et al, in 2006, proposed a method of using "DTW with fusing Principle Component Analysis (PCA) for fault diagnosis in complex industrial systems" [2]. Tarar S, in 2010, proposed "the recognition and generation of the commands for desktop items activation using DTW algorithm" [3]. Chalmers N. et al, in 2011, used DTW distance metric for "the assessment of sensorimotor impairment resulting from stroke [4]." Pettjean F. et al, in 2012, used DTW for "Satellite image time series analysis." High resolution images of earth from space are easily available to us because of space missions. But these time series are distorted because of meteorological phenomena. Thus, DTW was used for the comparison of pairs of time series which are irregularly sampled [5]. Xingzhe Xie, in 2014, suggested the use of DTW in "extracting the road between each two directly connected intersections while inferring the road network from Global Position System (GPS) traces" [6].

Sart D. et al investigated the "Graphics Processing Unit (GPU) and FPGA based acceleration of subsequence similarity search under DTW" and acquired the maximum speed up of 4500× times for FPGA and 29× times for GPU over software [7].

James Shueyen Tai, Kin Fun Li and Haytham Elmiligi proposed the hardware design for a DTW processing unit in VHDL [8]. They used Xilinx Virtex 7 xc7vx330t which provides 408,000 programmable slices and 203,000 look-up tables. They showed that their simulated design used only 6,844 (≈ 1%) of slices and 10,287 (≈ 5%) of LUTs. A. Madhavan, in 2014, proposed race logic architecture for hardware acceleration of dynamic programming algorithms, but it was not found feasible for mapping on general purpose FPGA [9].

DTW, though a vital algorithm, does not handle the noise efficiently. Thus, Longest Common Subsequence (LCSS) was introduced to execute time series noisy data [10]. It computes the number of elements, which are alike, between two time series, thus minimizing the effect of noise [11]. In the last decade, a number of applications of LCSS have been proposed. Elhadi M. Et al, in 2009, proposed the use of LCSS in the calculation of similarity between texts on the basis of their syntactical structures to prevent duplicate documents and web pages [12]. Campos R.A.C., in 2012, suggested its use in source code plagiarism detection to provide highly accurate results on the basis of string based

comparison [13]. Jinn-Jian Liaw, in 2013, proposed its use in recognizing the ambulance siren sound in Taiwan using the high and low frequency features for comparison [14].
Paravinmia E. Introduced an improved form of LCSS for searching similar region in DNA sequences with an additional benefit of reducing memory space complexity [15].
Though the elastic measures are efficient enough to incorporate different lengths of the time series, their quadratic computational complexity is a drawback. So, Vladimir Kurbalija et. Al examined the effect of global constraints on DTW and LCSS to compensate for the computational complexity [16].
LCSS can handle noise and local time shifting, but does not give penalty to gaps. So, Edit Distance on Real Sequences (EDR) was popularized [17] which gives penalty to the gaps also. These three distance measures (DTW, LCSS, and EDR) do not meet the requirements of triangle inequality and hence, are not metric. Thus, Edit Distance with Real Penalty (ERP) was proposed by *Chen* and *Ng*, such that it assumes real penalty between two non-gap elements, but a constant value for measuring the distance for gaps. It satisfies metricity property and can support local time shifting [18]. Move Split Merge (MSM) was introduced by *Stefan, Athitsos and Das* (2012) [19]. According to *Stefan* [19], MSM incorporates most of the desirable traits of a similarity measure: it is robust to temporal misalignments, translation invariant and a metric. These are the most important similarity measures which are being used extensively in information retrieval from time series and trajectory data. They are limited to software applications and have yet not been introduced in market as real time processors. Only, DTW's implementation on FPGA has been suggested till now [7,8]. Since, there is enormous number of applications of these similarity measures, we have implemented one of them, i.e. to detect plagiarism in music. It has been an area of interest for many researchers during the past few years. Seifert F., in 2003, proposed a model for the comparison of musical documents based on semantic relationship. Plagiarism detection was a byproduct of this functionality [20]. Soham De et al. presented a method of plagiarism detection in polyphonic music based on DTW by extracting the features using non- negative matrix factorization method [21]. Dittmar C, in 2012, proposed a toolbox for plagiarism detection in music, but under the supervised scenario [22].

1.4 Tools Used

In this thesis, we propose the hardware design of similarity measures using VERILOG Hardware Description Language (HDL).The system is implemented using Xilinx Integrated Software Environment (ISE) which is used to synthesize and process HDL algorithms on FPGA. FPGAs are prototypes for integrated circuit designs [1,2,4]. They are high density semiconductor devices which can be electrically programmed after manufacturing by the end user at any point of the design cycle to realize different logical problems. They are different from Application Specific Integrated Circuits (ASICs) in the sense that they provide high programmability and do not require replacement of hardware in the successive upgradation of designs in contrast to ASICs which are custom built for a particular design. The leading manufacturers of FPGA in market are Xilinx, Altera, Actel, Lattice, Quicklogic etc. Xilinx and Altera are most competitive one. Target device xc3s400-pq208 has the following properties: 3584 Slices, 7168 Look- Up Tables (LUTs), 141 Input/ Output Blocks (IOBs). The application of plagiarism detection in music was implemented in MATLAB.

Table 1.1 Tools used in the project

Tool	Purpose
MATLAB R2009a	For the comparison of five algorithms in their efficacy to detect plagiarism detection in music.
Xilinx ISE	For the synthesis of similarity measures on Spartan xc3s400-pq208 device.
ModelSim	For simulation of similarity measures to ensure their accuracy.

1.5 Organization of Thesis

The rest of the thesis is formulated as follows: In Chapter 2, we describe Dynamic Time Warping (DTW) algorithm, an extensively used similarity measure in speech processing. Then, Longest Common Subsequence (LCSS) has been introduced, which can handle noise as well as local time shifting. It also provides the detailed explanation of Edit Distance on Real sequences, a solution to the shortcomings of LCSS in not giving penalty to gaps. Since the above described algorithms are not metric, the next explanation is of Edit Distance with Real Penalty (ERP), a metric distance measure which can handle noise as well as local time shifting. Finally, the most recently introduced similarity measure Move Split Merge (MSM) has been discussed along with its properties and advantages over other distance measures. Chapter 3 describes the dynamic programming approach, which we have used for the hardware implementation of the similarity measures. In Chapter 4, a comparative analysis of all the five algorithms has been done to show their efficacy in speech processing to detect plagiarism in music. Chapter 5 shows the results of the hardware utilization and delay on implementing the five algorithms. Finally, Chapter 6 concludes the work and presents the directions for future work.

2 Time Series Similarity Measures

2.1 Introduction

Time series is a massive collection of data indexed sequentially with respect to time. It shows the measurements of a quantity taken at equal intervals of time. For example, daily temperature readings, market sales, ECG and EEG data, etc. Temporal database is a huge assembly of time series. One of the most common functions to be performed on time series database is the similarity analysis. If we take the example of ECG, the similarity problem will be defined as “Does the ECG of the patient had similar beat patterns over the past few days?”

Other computations on time series include indexing, subsequence similarity, clustering, etc. They all require the efficient similarity measures to allow for imprecise matches. Examples of the above mentioned problems are given below:

- Indexing problem – “Find all seismic zones whose seismic activity variations are similar to high risk zone.”
- Subsequence similarity problem – “Find all the songs that have the word ‘shining’ in their lyrics.”
- Clustering problem – “Group regions with similar weather patterns.”

The primitive approach to the indexing problem is to select certain features and use them to find the similarity between two time series. For example, while comparing two speech pieces for similarity, features like tonality, mel-frequency cepstral coefficients (mfcc), etc. can be used.

Given two time series A and B, a distance function Dist calculates the distance between the two time series, denoted by Dist(A, B). The distance measures which compare the j^{th} point of one time series to the j^{th} point of another are referred as lock-step measures (e.g., Euclidean distance and the other L_p norms), and distance measures which conduct comparison of one-to-many points (e.g., DTW) and one-to-many/ one-to-none points (e.g., LCSS) as elastic measures.

Following the categorization introduced by Esling P and Agon C [26], the time series distance measures are usually divided into four categories: **shape based, edit based, features based, and structure based.**

2.2 Shape based distance measures

This first category of distances is based on directly comparing the raw values and the shape of the series in different manners.

2.2.1 L_p distances

L_p distances are those that derive from the different L_p norms [27]. These distances are rigid metrics that can only compare series of the same length. However, due to their simplicity, they have been widely used in many tasks related to time series. Given two time series A= {a_0, a_1, ...a_{N-1}} and B={b_0, b_1,..........b_{N-1}}, the different forms of the L_p distances and their formulas are provided in Table 2.1. It must be noted that the Euclidean Distance is a special case of the Minkowski distance, but it is explicitly included because it is a baseline distance measure in the area of time series data mining.

Table 2.1 Lp Distances

Serial No	Distance	p	Distance Formula
1	Manhattan	p = 1	$\sum_{i=0}^{N-1} \mid a_i - b_i \mid$
2	Minkowski	1< p < inf	$\sqrt[p]{\sum_{i=0}^{N-1} (a_i - b_i)^{\frac{1}{p}}}$
3	Euclidean	p = 2	$\sqrt{\sum_{i=0}^{N-1} (a_i - b_i)^2}$
4	Infinite norm	p = inf	$\max_{i = 0,\ldots\ldots,N-1} \mid a_i - b_i \mid$

2.2.2 Dissim Distance

The Dissim distance was introduced by Frentzos, Gratsias, and Theodoridis [43] and is specifically designed for series collected at different sampling rates. This means that each series will be defined in finite set of time instants, but these can be different for each series. The Dissim distance requires a continuous representation of the series and so, the series that are being compared are assumed to be linear between sampling points. Once this is done, the definite integral of the Euclidean distance between them is calculated.

$$\text{Dissim(A,B)} = \sum_{i=0}^{K-1} \int_{t_i}^{t_{i+1}} D_{A,B}\,(t)dt \qquad \text{.........2.1}$$

where, T = { t_0,......,t_{K-1} } is a global time index that fuses the time indexes of both series by taking all the points that appear in both sets. $D_{A,B}(t)$ represents the Euclidean distance between the series at time-stamp t.

2.3 Edit based distances

Edit distance was initially presented to calculate the similarity between two sequences of strings and is based on the idea of counting the minimum number of edit operations (insertion, deletion and substitution) which are necessary to convert one sequence into the other.

The problem of working with real numbers is that it is difficult to find exact matching points in two different sequences and, therefore, the edit distance is not directly applicable. Different adaptations have been proposed in the literature according to recent reviews, for eg. By Wang et al. [28].By using the delete and insert operations, all these distances are able to work with series of different length. LCSS, EDR and ERP come under this category of distance measures.

2.4 Feature based distances

This category of distance measures focuses on extracting a set of features from the time series and calculating the similarity between these features instead of using the raw values of the series.

2.4.1 Distances based on Pearson's correlation

Pearson's correlation between two time series A= {a_0, a_1, ...a_{N-1}} and B={b_0, b_1,..........b_{N-1}} is defined as

$$PC\ (A,B) = \frac{\sum_{i=0}^{N-1}(a_i-\overline{a})(b_i-\overline{b})}{\sqrt{(a_i-\overline{a})^2}\sqrt{(b_i-\overline{b})^2}} \qquad \text{......... 2.2}$$

where $\bar{a}$ and $\bar{b}$ are the mean values of the series.

Based on this value, two distance measures were introduced by Golay, Kollias, Stoll, Meier, Valavanis, and Boesiger [29].

$$d_{PC1}\ (A,B) = \left(\frac{1-PC}{1+PC}\right)^{\beta} \qquad \text{......... 2.3}$$

$$d_{PC2}\ (A,B) = 2(1\text{-}PC) \qquad \text{......... 2.4}$$

where β is a positive parameter defined by the user.

2.4.2 Distance based on the cross- correlation

This distance is presented in [30] and is based on the cross- correlation between two series. The cross- correlation between two series at lag k is calculated as

$$CC_k\ (A,B) = \frac{\sum_{i=0}^{N-1-k}(a_i-\overline{a})(b_{i+k}-\overline{b})}{\sqrt{(a_i-\overline{a})^2}\sqrt{(b_{i+k}-\overline{b})^2}} \qquad \text{......... 2.5}$$

where $\bar{a}$ and $\bar{b}$ are the mean values of the series as in the previous case. Based on this, the distance measure is defined as:

$$d_{CC}(A,B) = \sqrt{\frac{(1-CC_0(A,B))}{\sum_{k=1}^{max}CC_k(A,B)}} \qquad \text{.......... 2.6}$$

2.5 Fourier Coefficients based distance

As its name indicates, the similarity calculation in this case is based on comparing the Discrete Fourier Transform (DFT) coefficients of the series.

Given a numeric series A = $\{a_0, a_1, ...a_{N-1}\}$, its DFT can be easily calculated and contains an array of Fourier Coefficients of the series. The value of each coefficient measures the contribution of its associated frequency to the series and, based on this, the Inverse Fourier Transform provides the means to represent the sequences as a combination of sinusoidal forms.

In the case of real sequences such as time series, the Discrete Fourier Transform is symmetric and therefore it is sufficient to study the first N/2 +1 coefficients. Furthermore, it is commonly considered that, for many time series, most of the information is kept in their first n Fourier Coefficients, where $n < \frac{N}{2} + 1$.

Based on all this information, the distance between two time series A and B with Fourier Coefficients $\{(a_0, b_0),......,(a_{N/2}, b_{N/2})\}$ and $\{(a_0', b_0'),......,(a'_{N/2}, b'_{N/2})\}$ is given by the Euclidean distance between the first n coefficients:

$$F(A,B) = \sqrt{\sum_{i=0}^{n}((a_i - a_i')^2 + (b_i - b_i')^2)} \qquad .... 2.7$$

The most basic distance measure is the Euclidean distance which gives the (dis)similarity between sequences A and B as $L_p(A,B)$, where L_p distance is given by

$$L_p(A,B) = \sqrt[p]{\sum_{i=0}^{N-1}(a_i - b_i)^{\frac{1}{p}}} \qquad ....2.8$$

It is easy to compute, but does not allow for translations on time axis and also for different rate of variations. Also, it does not provide good approximation in feature space as compared to the sequence distance in original space. Goldin and Kanellakis, in 1995, proposed normalization of sequences with respect to mean and variance [31]. But, still it did not allow for acceleration/ deceleration along the time dimension and phase shifts in time.

2.6 Dynamic Time Warping Algorithm

2.6.1 Introduction

When the distance between two sequences has to be found, the most intuitive approach is to use Euclidean Distance. However, in case of temporal sequences, it does not provide promising results because it does the one-to-one sample comparison and thus, very sensitive to translations across time axis. Dynamic Time Warping (DTW) is the benchmark algorithm in speech processing. It can be considered as the generalized form of Euclidean distance as it allows the elastic shifting of time axis i.e. a time series can be compressed or stretched to align with the other series.

DTW is an edit distance based similarity measure. The basic idea behind it is to determine a set of edit operations that can be used to convert one sequence, which can be numerical, alphabetical or temporal sequence, into another. Each operation has an associated edit cost, and the DTW distance between two time series is determined by the lowest cost of the sequence of edit operations that converts one into another. It is very efficient in software computing. However, its applicability for real time processors is limited to literature because of space and time restrictions in its hardware implementation. In order to use dedicated hardware for similarity search using DTW, we need to examine the constraints of the target device and DTW computation.

The edit operations used by DTW to transform one sequence to another are substitution, deletion, and insertion. It, also finds a warping path between the input and the template for the cheapest cost alignment.

2.6.2 Definition of DTW Distance

There are certain desirable properties that a similarity measure should satisfy. One such property is immunity against the misalignments of the sample values. Euclidean distance lacks this property as it does one-to-one comparison. Thus, cannot handle even the single point displacement on the time axis.

Let $A = (a_1,.....,a_m)$ and $B = (b_1,.......,b_n)$ be two time series of length m and n respectively. As shown in Fig 3.1, we use a simple Dynamic Programming (DP) approach to find the DTW distance between A and B. For i^{th} row and j^{th} column, where $1 \leq i \leq m$ and $1 \leq j \leq n$, we define DTW(i,j) as the DTW distance between the first i samples of A and the first j samples of B, i.e., the value of DTW(i, j) depends upon DTW (i, j-1), DTW (i-1,j) and DTW(i-1,j-1) and is calculated recursively. In this way, the DTW distance between A and B is computed to be DTW (m,n).

2.6.3 Definitions of DTW operations

As mentioned in section 2.6.1, DTW uses three edit operations, namely substitution, insertion, and deletion to find the distance between two time series. These operations are explained below with the help of examples:

- Substitution: a template letter or sample has been changed in the input. For eg.

K I T E

C I T E

- Insertion: a spurious letter or sample has been introduced in the input. For eg.

S C H O L

S C H O O L

- Deletion: a template letter or sample is missing from the input. For eg.

P O T A T O E

P O T A T O

The above mentioned examples are small enough to be aligned manually, but for the proper alignment of long sequences, a dynamic algorithm is mandatory which can provide the cost in terms of minimum number of fundamental operations required. An example is shown in Fig 2.1

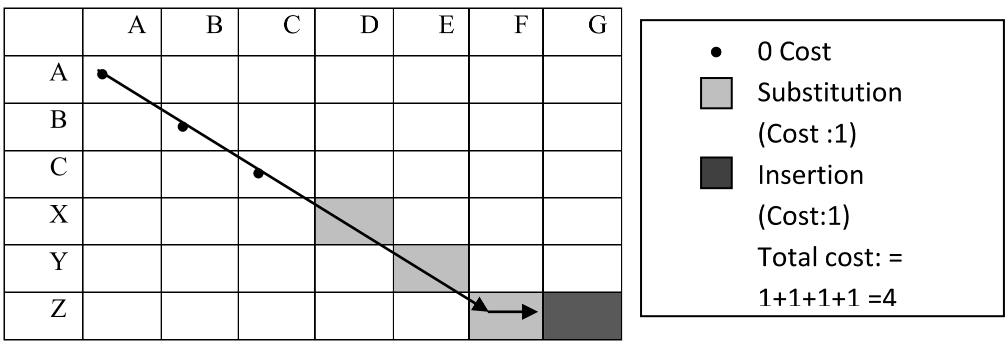

Fig 2.1 Cost of operations

2.7 Longest Common Subsequence

2.7.1 Introduction

Longest Common Subsequence (LCCS) is a similarity measure which is used to find the longest sequence present in both the sequences being compared. Given two sequences (numerical, alphabetical or temporal) A and B of lengths m and n respectively, LCSS (A, B) gives the length of the maximal length subsequence that appears in the same relative order in both A and B.

For example: if A= [a b d f g h] and B = [a f b h], then LCSS (A,B) = 3 and the longest common subsequence is [a f h].

The most common use of LCSS is in studying genetic structures, where DNA sequences are compared for hereditary information [32]. Another use of LCSS is in “diff” utility of Unix program. It is used to compare two different versions of the same file by finding a longest common subsequence of the lines of the two files.

The similarity function is shown in (2.9)

$$\text{LCSS(A,B)}= \begin{cases} 0 \;\; if\; m = 0 \; or \; n = 0 \\ LCSS\big(Rest(A), Rest(B)\big) + 1 \;\; if\; m, n > 0 \; and \; a_i = b_j \\ \max\Big(LCSS\big(Rest(A), B\big), LCSS\big(A, Rest(B)\big)\Big) \;\; if\; a_i \neq b_j \end{cases} \qquad \text{......... 2.9}$$

where max is maximum.

While using LCSS for time series, a matching threshold δ has to be set up because the features of time series can have real values, and not integers. So, if the two elements fall within the threshold limit, they can be assumed similar.

2.7.2 Properties of LCSS

LCSS can take up on noise because it provides the distance between two elements in the form of two values, 0 and 1, thus eliminating the large distance effects caused by noise [11].

Consider the following example:

Let the input sequence A = [1 2 3 4 5] has to be compared with three other temporal sequences:

B = [11 12 7 8 9]

C = [1 2 100 3 4 5]

D = [1 2 100 102 3 4 5]

The third element of C, as well as, the third and the fourth elements of D are noise because their values are abnormally higher than the values in its vicinity. According to the algorithm in 2.9, the legitimate ranking in terms of similarity to A is C/ D, B since except noise, the remaining elements of C and D match fully with the elements of A. The ranking, according to DTW, are B, C, D, because DTW demands each element of the input sequence to have a corresponding element in the template sequence, even for noise.

2.8 Edit Distance on Real Sequences

2.8.1 Introduction

DTW can handle local time shifting but not noise. LCSS sacrifices accuracy in not giving penalty to gaps. It is possible that two query time series have the same LCSS distance to the template sequence, but their gap sizes in between elements differ from the matching subsequence. Thus it will consider the longest word in dictionary "pneumonoultramicroscopicsilicovolcanoconiosis" and the words "mono", "microscopic", "silico", "volcano" same [33]. Thus, Edit Distance on Real Sequences (EDR) was popularized [17] which gives penalty to the gaps also. EDR is robust and accurate approach in computing the dis/similarity between two time series.

2.8.2 Mathematical Formulation of EDR

EDR is an edit distance based similarity measure which uses three operations (Insertion, Deletion, and Substitution) to convert one sequence to other. Given two sequences A and B of lengths m and n, respectively, the EDR between A and B is the number of edit operations (insertion, deletion and substitution) that are required to change A into B.

EDR (A, B) is defined as:

$$EDR\,(A,B) = \begin{cases} m \;\; if \; n = 0 \\ n \;\; if \; m = 0 \\ \min\,(EDR(Rest(A), Rest(B)) + d, EDR(Rest(A), B) + 1, \\ \qquad EDR(A, Rest(B)) + 1) \end{cases} \qquad \text{......... 2.10}$$

where d is 0 if $a_i = b_j$ and 1 otherwise.

The matching threshold δ can be used for time series, such that d is 0 if $|a_i = b_j| \leq \delta$, and 1 otherwise.

2.9 Edit Distance with Real Penalty

2.9.1 Introduction

We have seen that the distance functions for time series are build upon two classes: L_p norms, which cannot handle local time shifting but are metric functions, and those which can work well with local time shifting but lack the property of metricity (DTW, LCSS, EDR).

Edit Distance with Real Penalty (ERP) is a distance function which is a metric and can handle local time shifting. Thus, *Lei Chen* and *Raymond Ng* called it "*Marriage of L_1-norm and the edit distance* [18]."

Edit distance on Real sequences(EDR) work well for strings- sequences of alphabets and symbols, but for time series where the elements are real numbers (varying from 0 to +∞), strict equality is not feasible. Thus a soft limit on equality is required. An alternative is to ease off equality to be within a certain tolerance limit δ. But this δ tolerance makes the edit distance a non- metric. The main reason of DTW not satisfying triangle inequality is that, when a gap has to be added, the previous element is replicated [18].

ERP inculcates the positive points of both DTW and EDR to make it a metric distance measure. It uses one of the L_p norms for non-gap elements, but a constant value is subtracted when a gap is encountered. It differs from EDR in averting δ tolerance and differs from DTW in not imitating the previous element.

2.9.2 Mathematical formulation of ERP

Let A and B be two time series of lengths m and n respectively. The ERP distance between the two is given by:

$$\begin{gathered} \sum_1^n |b_i - g| \; if\ m = 0 \\ \sum_1^m |a_i - g| \;\; if\ n = 0 \\ \min\{ERP\big(Rest(A), Rest(B)\big) + d_{erp}\big(a_i, b_j\big), \\ ERP(Rest(A), B) + d_{erp}(a_i, gap), \\ ERP\big(A, Rest(B)\big) + d_{erp}(b_j, gap)\} \qquad otherwise \end{gathered} \qquad \text{......... 2.11}$$

Where,

$$d_{erp}(a_i, b_j) = \begin{cases} |a_i - b_j| & \text{if } a_i, b_j \text{ are not gaps,} \\ |a_i - g| & \text{if } b_j \text{ is a gap,} \\ |b_j - g| & \text{if } a_i \text{ is a gap.} \end{cases} \qquad \text{........ 2.12}$$

Similar to other edit distances, ERP uses a set of edit operations that can convert any time series to any other time series. The fundamental operations in ERP are Insertion, Deletion, and Substitution. It differs from other edit distances in the cost of operations. The cost of insertion and deletion of a value depends entirely on the absolute magnitude of that value. Thus, ERP does not deal with all the values equally.

2.10 Move Split Merge

2.10.1 Introduction

Move Split Merge (MSM) is the most recent edit distance based similarity measure which uses three operations- *move, split and merge* to convert one time series into other. It was introduced by *Stefan* and *Athitsos* in 2012. Each operation incurs a cost and the cheapest cost of converting one sequence to other is given by MSM distance. It has an important trait of being metric.

The fundamental operations used in this similarity measure are: Move, Split and Merge. A Move operation alters the value of an element in a sequence. A Split operation splits an element into two consecutive elements. The two generated elements have the value of the original element. A Merge operation merges two consecutive elements into a single element if they have the same values. Each operation has a cost associated with it. The Move operation incurs a cost which is equal to the absolute difference between the two elements. The cost incurred by the Split and the Merge operations are equal in value and constant.

MSM was introduced to inculcate all the positive traits of distance measures. It is robust to temporal misalignments, is a metric and treats all values equally in contrast to ERP, where the insertion and deletion costs are determined by the magnitude of the value.

2.10.2 Definition of the MSM Distance

In ERP, the cost of insertion and deletion solely depends on the absolute magnitude of the value being inserted or deleted. But in MSM, the cost does not only depend on that value but also its neighbours i.e. insertion of a 3 between two 3s should cost the same as insertion of a 1 between two 1s. However, this cost should be less than inserting a 3 between two 1s. Thus, here Insert operation is replaced by a Split, which creates a new element followed by a Move, which sets the value of the new element. In the similar fashion, a Delete is replaced by a Move, which makes an element equal to either following or proceeding element, followed by a Merge, which deletes the just moved element.

2.10.3 Mathematical definitions of MSM operations

Let time series A = (a_1,....., a_n) be a real numbered finite sequence. The Move operation and its cost are determined as follows:

$$\text{Move}_{j,p}(A) = (a_1,\ldots\ldots,a_{j-1},\ a_j+k,\ a_{j+1},\ldots,a_n) \qquad \text{.........2.13}$$

$$\text{Cost}\ (\text{Move}_{j,k}) = |k| \qquad \text{.........2.14}$$

Here, a new time series A\` is created by the Move $_{j,p}$(A) operation which is similar to A, except that the j^{th} element is moved from value a_j to value a_j+k. The cost of this movement is given by the absolute value of k.

Example:

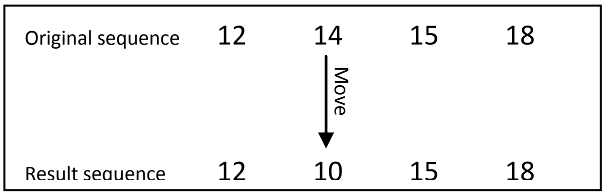

Fig 2.2 Example of Move Operation

The Split operation, and its cost are defined as follows:

$$\text{Split}_j\ (A) = (a_1,\ldots,a_{j-1},\ a_j,\ a_{j+1},\ldots,a_n) \qquad \text{.........2.15}$$

$$\text{Cost}\ (\text{Split}_j) = c \qquad \text{.........2.16}$$

Where, c is a constant.

Here, a new time series A\` is created by Split_j(A) operation, which is similar to A except that the j^{th} element of A is split into two consecutive elements. The cost of this split is a non negative constant c, which is a system parameter.

Example:

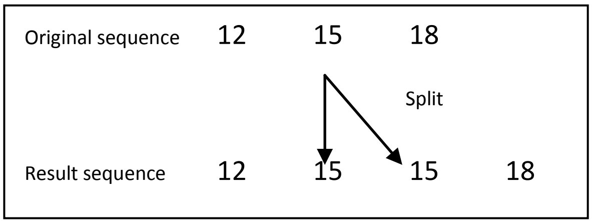

Fig 2.3 Example of Split Operation

The Merge operation, and its cost are defined as follows:

$$\text{Merge}_j(A) = (a_1, \ldots, a_{j-1}, a_{j+1}, \ldots\ldots a_n) \qquad \ldots\ldots\ldots 2.17$$

$$\text{Cost}(\text{Merge}_j) = c \qquad \ldots\ldots\ldots 2.18$$

Where, c is a non negative constant.

It is applicable iff $a_j = a_{j+1}$

Here, operation Merge $_j$(A) creates a new time series A\`, where the elements a_j and a_{j+1} are merged into a single element if they are equal in value.

Example:

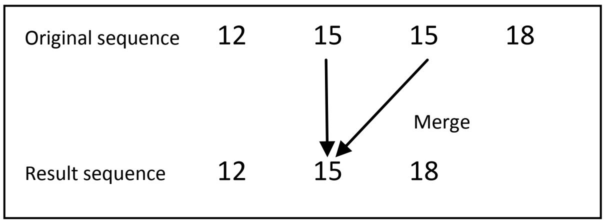

Fig 2.4 Example of Merge Operation

Given two time series A and B, the MSM distance MSM (A, B) is defined as the cost of transformation of A into B.

2.10.4 Properties of MSM

- Symmetry

  Let S be one of the MSM operations and A and B be two time series such that S(A) = B. MSM being symmetric implies $S^{-1}(B) = A$. The inverse of Move $_{j,p}$ is Move $_{j,-p}$, while the Split $_j$ and Merge $_j$ are inverses of each other. S, a sequence of operations, is also reversible. Also, it is intuitive that Cost (S) = Cost (S^{-1}). Therefore, MSM (A,B) = MSM (B,A).

- Triangle Inequality

It states that the sum of the two sides of a triangle is greater than or equal to the third side. Let A, B and C be three time series. Thus,

MSM(A,B) + MSM(B,C) ≥ MSM(A,C) .It means that the cost of transformation of A into C is either less than or equal to the summation of the costs of transformations of A into B and then B into C. Clustering, which is a prevailing operation in data mining, is generally designed for metric spaces. Thus, distance measures for time series should be a metric.

- Invariancy to the choice of origin

Let A = (a_1,....,a_n) be a real numbered time series. Translation of A by a real number t means t is added to each element of the time series, to produce A+t = {a_1+t,......,a_n+t). If MSM distance is invariant to the choice of origin, then for given time series A and B, and a translation t, MSM (A, B) = MSM (A+t, B+t). The MSM distance is said to be invariant to the choice of origin since any transformation S which transforms A to B also transforms A+t to B+t.

3 Dynamic Programming Approach for Hardware Implementation of Similarity Measures

3.1 Introduction

Dynamic programming is an effective method to solve recursive problems as it prevents the overhead of function calls. The approach is to arrange the computation in such a way that whenever the result of a subproblem is required, it has been computed in advance and can simply be found in a table.

Let's say the two time series to be compared are A and B of lengths m and n respectively [35]. The process of dynamic programming is as follows: First, an $m \times n$ two dimensional array M is formulated. Next, each element a_i of A is compared with each element b_j of B for all $1 \le i \le m$ and $1 \le j \le n$. The result of the comparison of a_i and b_j is added to the best cumulative score between $(a_1,.....,a_{i-1})$ and $(b_1,.....,b_{j-1})$ and stored in M at position (i ,j). Once all the mn comparisons have been conducted and the array M is completely filled, the final cost is stored in M(m,n) [35].

In this project, all the five algorithms have been implemented using this approach as shown in the further sections.

3.2 Dynamic Time Warping

3.2.1 Computation of DTW distance

Function DTW(A,B)

Inputs:

Time Series $A = (a_1,.......,a_m)$

Time Series $B = (b_1,.........,b_n)$

Initialization:

$DTW[1,1]: = |a_1 - b_1|^2$

For $i=1.....m$

$DTW[i,1]= d(A[i],B[1])+ DTW[i-1,1]$

For $j= 1......n$

$DTW[1,j] = d(A[1],B[j])+DTW[1,j-1]$

Main Loop:

For $i = 2.......m$

For $j=2....n$

Cost: $= d(A[i],B[j])$

$DTW[i,j]: = Cost+ \min (DTW[i,j-1], DTW[i-1,j], DTW[i-1,j-1])$

Result:

DTW distance is $DTW [m,n]$

Fig 3.1 Mathematical formulation of DTW

where,

d(A[i],B[j]) is the square of Euclidean Distance and DTW[i,j] is the minimum edit distance between i^{th} and j^{th} elements of sequences A and B respectively.

In hardware implementation, the algorithm starts by developing a m×n matrix whose elements are the pair wise distances between A and B. First, the square of Euclidean distance is calculated for the first element of the matrix. Then, first row and first column elements, are calculated which require the previous elements' values and no comparisons (coloured in dark grey). Finally, the rest of the entries in matrix are calculated which require comparison among three previously calculated elements (coloured in light grey). Hence, starting from the origin and computing minimum path cost for every grid point entry, proceeding from top left to bottom right corner provides the minimum edit distance as the last entry. To determine the alignment, a back-pointer is maintained from A to its antecedent, which provides the minimum cost to it. At the end, back tracing gives the best alignment.

Let's take an example to show the procedure:

Let A= [1 2 3 4 5 6 7 8 9 0] and B= [1 1 2 2 3 3 3 4 4 5] be two sequences to be compared. The accumulated distance matrix to find DTW (A, B) is shown in Table 3.1.

Table 3.1 Accumulated distance matrix for DTW

	1	2	3	4	5	6	7	8	9	0
1	0	1	5	14	30	55	91	140	204	205
1	0	1	5	14	30	55	91	140	204	205
2	1	0	1	5	14	30	55	91	140	144
2	2	0	1	5	14	30	55	91	140	144
3	6	1	0	1	5	14	30	55	91	100
3	10	2	0	1	5	14	30	55	91	100
3	14	3	0	1	5	14	30	55	91	100
4	23	7	1	0	1	5	14	30	55	71
4	32	11	2	0	1	5	14	30	55	71
5	48	20	6	1	0	1	5	14	30	55

$(6-2)^2$ + min (14, 30, 14)
= 16 +14
= 30

Minimum edit distance

The distance between the two sequences is DTW (A, B) = 55. In time series analysis, DTW is an algorithm for computation of the extent of similarity between two temporal sequences which may differ in time or speed. If two sequences of similar nature, but running at different speeds are to be compared for similarity, dynamic time warping is an optimum approach. The sequences are "warped" non-linearly in the time dimension to determine a measure of their similarity. This sequence alignment process is generally used in classification problems. Despite the fact that DTW measures a distance-like quantity between two given sequences, it doesn't assure the triangle inequality to hold i.e. if A, B and C are three sequences, then DTW(A,B) + DTW(B,C) may not be greater than or equal to DTW (A,C)

3.3 Longest Common Subsequence

3.3.1 Computation of LCSS distance

Let A = [a_1,...........,a_m] and B = [b_1,..............,b_n] be two sequences to be compared for similarity. Fig 3.2 shows a trivial dynamic programming approach for computing the LCSS distance between A and B. The LCSS distance between the first i elements of A and the first j elements of B is denoted by LCSS [i, j], for each (i, j) such that $1 \leq i \leq m$ and $1 \leq j \leq n$. Consequently, the LCSS distance between A and B is LCSS [m, n]. As the algorithm in Fig 3.2 shows, for $i > 1$ and $j > 1$, LCSS [i, j] can be computed recursively based on LCSS [i, j-1] and LCSS [i-1, j].

Function LCSS (A, B)

Inputs:

Time Series A = (a_1,.......,a_m)

Time Series B = (b_1,.........,b_n)

Initialization:

For i=1.....m

LCSS[i,1]= d(A[i],B[1])+ LCSS[i-1,1]

For j= 1......n

DTW[1,j] = d(A[1],B[j])+LCSS[1,j-1]

Main Loop:

For i = 2.......m

For j=2....n

Cost: = d(A[i],B[j])

LCSS[i,j]: = Cost+ max (LCSS[i,j-1], LCSS[i-1,j])

D(A[i], B[j])= 0 if A[i]≠ B[j]

1 if A[i]=B[j]

Result:

LCSS distance is LCSS [m,n]

Fig 3.2 Algorithmic description of LCSS

A data flow model is used as a solution of our design as shown in Fig 3.3. The key to design the LCSS algorithm in Verilog is to recursively calculate each cell's distance and then add it to the maximum value of its adjacent cells in the cost matrix. Fig 3.3 shows the control flow block diagram in our Verilog solution.

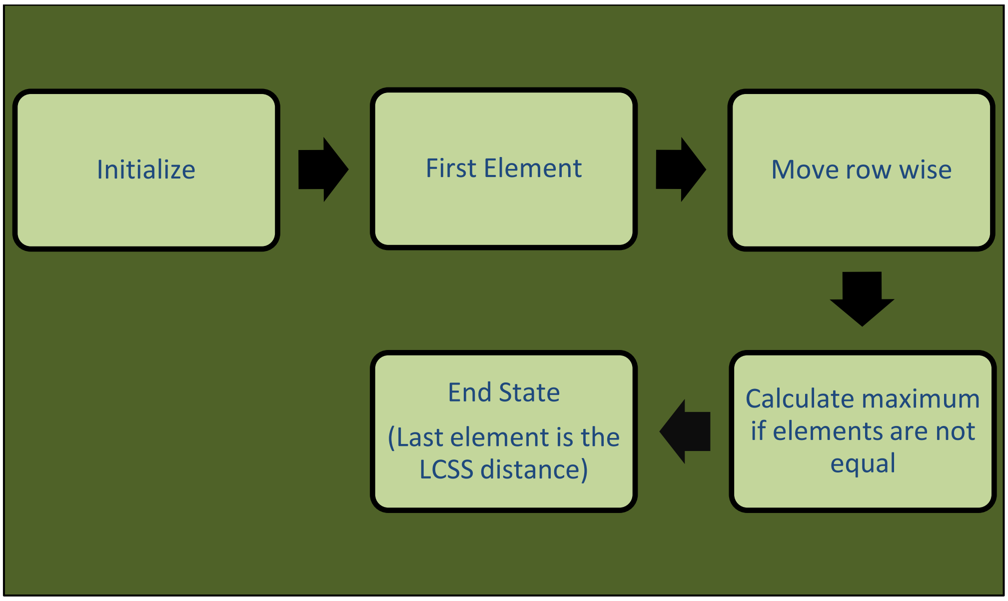

Fig 3.3 Control flow diagram for LCSS

The LCSS distance matrix is separated into four sections for ease of explanation as shown in Table 3.2 Only section 1 (highlighted in black) can be calculated with simple distance calculation from the first elements of two sequences because it is the only element that does not have dependency issue. Sections two and three (both highlighted in deeper grey) are the first row and first column in the matrix. For each element, the distance is calculated and added to the values of its previous elements. A simple design solution could be implemented by the use of case statements. However, Verilog only allows loops with static range, which means only definite number of iterations is allowed. Therefore, additional case statements are required for longer sequences.

Section 4 (highlighted in lighter grey) in the distance matrix is more complicated to compute than sections 2 and 3. As shown in Table 3.2, each element in section 4 requires a process to find the maximum value among the two adjacent elements (if the elements do not match) or to access the diagonally opposite element (if the elements match). Data dependency is tightly coupled to the system clock cycles. Since data assignment to signal requires one clock cycle, it is not possible to compute all elements in one clock cycle without independent element calculations.

Table 3.2 Accumulated distance matrix for LCSS

	1	2	6	3	7	5
6	0	0	1	1	1	1
2	0	1	1	1	1	1
1	1	1	1	1	1	1
2	1	2	2	2	2	2
7	1	2	2	2	3	3
6	1	2	3	3	3	3
5	1	2	3	3	3	4

0 + max (1, 2)
= 2

Length of longest subsequence

The minimum edit distance to transform A into B or B into A is obtained by doing |A| - LCSS (A, B) ‘deletions’ or |B| - LCSS (A, B) ‘insertions’.

3.4 Edit distance on Real Sequence

3.4.1 Computation of EDR distance

Let A = [a_1,...........,a_m] and B = [b_1,...............,b_n] be two real numbered temporal sequences to be compared for similarity. Fig 3.4 describes a simple dynamic programming algorithm for computing the EDR distance between A and B. The EDR distance between first i elements of A and first j elements of B is denoted as EDR [i, j], for each (i,j) such that $1 \leq i \leq m$ and $1 \leq j \leq n$. Consequently, the EDR distance between A and B is simply EDR [m, n]. As the algorithm in Fig 3.4 shows, for i > 1 and j > 1, EDR [i, j] can be computed recursively based on EDR [i, j-1] and EDR [i-1, j].

Function EDR (A,B)

Inputs:

Time Series A = $(a_1,\ldots\ldots.,a_m)$

Time Series B = $(b_1,\ldots\ldots\ldots,b_n)$

Initialization:

EDR[1,1] = d(A[1], B[1])

For i=2.....m

EDR[i,1]= d(A[i],B[1])+ EDR[i-1,1]

For j= 2......n

DTW[1,j] = d(A[1],B[j])+EDR[1,j-1]

Main Loop:

For i = 2.......m

For j=2....n

EDR[i,j]: =min (EDR[i,j-1]+1, EDR[i-1,j]+1, EDR[i-1, j-1]+d(A[i],B[j]))

d(A[i], B[j])= 1 if A[i]≠ B[j]

0 if A[i]=B[j]

Result:

LCSS distance is LCSS [m,n]

Fig 3.4 Algorithmic description of EDR

The EDR distance matrix is separated into four sections for ease of explanation as shown in Table 3.3 Only section 1 (highlighted in black) can be calculated with simple distance calculation from the first elements of two sequences because it is the only element that does not have dependency issue. Sections two and three (both highlighted in deeper grey) are the first row and first column in the matrix. For each element, the distance is calculated and added to the values of its previous elements. A simple design solution could be implemented by the use of case statements. However, Verilog only allows loops with static range, which means only definite number of iterations is allowed. Therefore, additional case statements are required for longer sequences.

Section 4 (highlighted in lighter grey) in the distance matrix is more complicated to compute than sections 2 and 3. As shown in Table 3.3, each element in section 4 requires a process to find the maximum value among the two adjacent elements (if the elements do not match) or to access the diagonally opposite element (if the elements match). Data dependency is tightly coupled to the system clock cycles. Since data assignment to signal requires one clock cycle, it is not possible to compute all elements in one clock cycle without independent element calculations.

Table 3.3 Accumulated distance matrix for EDR

	1	1	1	1	1
2	1	2	3	4	5
2	2	2	3	4	5
2	3	3	3	4	5
2	4	4	4	4	5
1	4	4	4	4	4

Min(3+1,3+1,4+1) = 4

Minimum edit distance

EDR will soon be renowned by being extremely efficient as the time-series similarity measure in software as well as hardware as it minimizes the effects of shifting and distortion in time while detecting similar shapes with different phases.

3.5 Edit distance with Real Penalty

3.5.1 Computation of ERP distance

Function ERP (A,B)

Inputs:

Time Series A = (a_1,.......,a_m)

Time Series B = (b_1,.........,b_n)

Initialization:

ERP[1,1]: = |$a_1 - b_1$|

For i=1.....m

ERP[i,1]= |A[i]-g|+ ERP[i-1,1]

For j= 1......n

DTW[1,j] = |B[j]-g|+ERP1,j-1]

Main Loop:

For i = 2.......m

For j=2....n

ERP[i,j]: = min (ERP[i,j-1]+ |B[j]-g|, ERP[i-1,j] +|A[i]-g|, ERP[i-1,j-1]+|A[i]-B[j]|)

Result:

ERP distance is ERP [m,n]

Fig 3.5 Algorithmic description of ERP

In hardware implementation, the algorithm starts by developing a m×n matrix whose elements are the pair wise distances between A and B. First, the L_1 norm is calculated for the first element of the matrix. Then, first row and first column elements, are calculated which require the previous elements' values (coloured in dark grey). Finally, the rest of the entries in matrix are calculated which require comparison among three previously calculated elements (coloured in light grey). Hence, starting from the origin and computing minimum path cost for every grid point entry, proceeding from top left to bottom right corner provides the minimum edit distance as the last entry.

Table 3.4 Accumulated distance matrix for ERP

	1	2	3	3	4	6
1	0	2	5	8	12	18
1	1	1	4	7	11	17
1	2	2	3	6	10	16
2	4	2	3	4	8	14
2	6	4	3	4	6	12
2	8	6	5	4	6	10

3.6 Move Split Merge

3.6.1 Computation of MSM distance

Function MSM_dist (A,B)

Inputs:

Time Series $A = (a_1,......,a_m)$

Time Series $B = (b_1,......,b_n)$

Initialization:

$Cost(1,1)=|a_1 - b_1|$.

For $i=2,...,m$:

$Cost(i,1) = Cost(i-1\ ,1)+C(a_i,a_{i-1},b_1)$

For $j=2,....,n$:

$Cost(1,j) = Cost(1,j-1)+C(b_j,a_1,b_{j-1})$

Main Loop:

For $i= 2,........,m$:

For $j=2,...,n$:

$Cost(i,j) = min\{Cost(i-1,j-1)+|a_i - b_j|,\ Cost(i-1,j)+C(a_i,a_{i-1},b_j),\ Cost(i,j-1)+C(b_j,a_i,b_{j-1})\}$

Output : *The MSM distance MSM(A,B)is Cost(m,n).*

Fig 3.6 Algorithm to find MSM distance

where,

$$C\ (a_i, a_{i-1}, b_j) = \begin{cases} c \ \ if\ a_{i-1} \le \ a_i \le b_j or \\ a_{i-1} \ge a_i \ge b_j\ , \\ c + \min\{abs(a_i - a_{i-1}), abs(a_i - b_j)\} \end{cases} \quad \text{.........3.1}$$

where *abs* is the absolute value and c is a constant value.

Let the two time series be A = {1 2 3 4 5} and B = {0 1 1 2 3}.

The methodology used is to arrange the computation in the form of a table, so that whenever the solution of a subproblem is required, it is already available to us as shown below:

Table 3.5 Accumulated distance matrix for MSM

	1	2	3	4	5
0	1	4	7	10	13
1	3	2	4	6	9
1	5	5	4	5	6
2	8	8	7	6	6
3	11	11	10	9	8

The first cell, colored in blackish grey, is the easiest value to compute as it is the absolute difference of the two values i.e. |1 – 0|.

The first row and the first column, coloured in dark grey, are recursively computed with the help of the previous results in the row and the column respectively.

The middle section, coloured in light grey, is the most complicated part of the computation as it demands the comparison between three previously computed values. The last element of the table is the required distance.

4 Software Based Evaluation of Similarity Measures for Plagiarism Detection in Music

4.1 Introduction

Music is present everywhere around us. It is present in car rides, hotels, homes, television shows, movies, etc. With a huge demand of songs for bands, movies, etc., writers and singers are pressurised to produce new songs, but face the challenge of ensuring that they are not copying an already existing song in any way. With the growing music industry, cases of plagiarism have become a critical concern for musicians. An enormous number of musical tracks are released every year. So, there must be a reliable and easier way to search through the huge database of songs that match the query song. If a real time processing tool for this purpose were exist, writers and singers could easily ensure that their songs are not already there in the market before releasing them to the public. Copyright infringement is an offence that, for the purpose of this project, refers to the writer or singer of a song reproducing some aspect of a prior copyrighted song, intentionally or unintentionally. To prevent the violation of copyright, an automated approach to plagiarism detection is essential [33]. DTW has been used a number of times to find the similarity. But, before using it, features need to be extracted which can effectively characterize a song.

The most important features of a song are tempo, rhythm, pitch, and melody. Tempo is the speed with which the notes are played. Rhythm is the time distance between each note in a melody. Each note has a different frequency, called pitch and melody is a succession of notes, varying in pitch, taking a recognizable and organized shape. These features are single valued, thus give an average approximation and do not require edit distance measures to find the distance. However, there are multi-valued features also, which can show the dynamic variations of the song. One of them is Mel Frequency Cepstral Coefficient (MFCC), which has been discussed in section 4.3.

4.2 Our contribution

As a part of this project, we have implemented the five distance measures (DTW, LCSS, EDR, ERP, and MSM) in MATALB and did the comparative analysis of using them to distinguish among three sets of songs (MATLAB code is given in Appendix A):

a. A pair of plagiarized songs,
b. Same song with different lengths, and
c. A random pair of songs

Intuitively, if a, b and c represent the distances of these sets respectively, then $c > a > b$, because the same song with different lengths should be detected as the same song. A plagiarized song may have some features close to the original one, but will not be exactly similar, while a random song pair will have highly distinguished features. We took 50 pairs of each set and did the computation on them (Plagiarized song pairs are given in Appendix C). We have used MFCC feature for the comparison.

4.3 Mel Frequency Cepstral Coefficient (MFCC)

Features are those components of audio signal which identify the linguistic contents and eliminate all the other worthless data like background noise, emotions etc.

Mel Frequency Cepstral Coefficients (MFCCs) are a feature widely used in Music Information Retrieval (MIR) systems. Introduced by Davis and Mermelstein in 1980 [39], MFCCs have become state- of-art. Computation of MFCCs is a five step process as shown below:

The first step in finding MFCCs of an audio file is to break it in frames to analyze the dynamic evolution of the feature [36]. A frame is nothing but the position of the window that moves sequentially along the temporal signal (as shown in Fig 4.1). The default length of the window is 50msec with half overlapping [36].

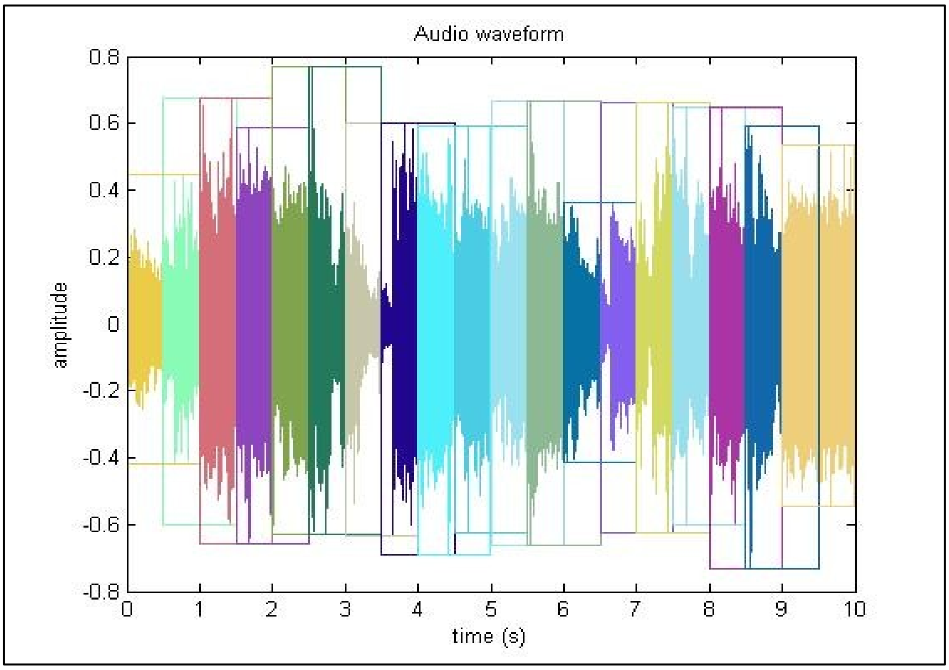

Fig 4.1 Decomposition of audio waveform in frames

The next step is to calculate the power spectrum of each frame [38]. It gives the power content of each frequency in a given frame.

The next step is to apply the ‘mel’ filterbank and sum the energy in each frequency region. The first filter, in melfilterbank, is very narrow and gives an indication of how much energy exists near 0 Hertz. The filters get wider as the frequency increases [38] (as shown in Fig 4.2).

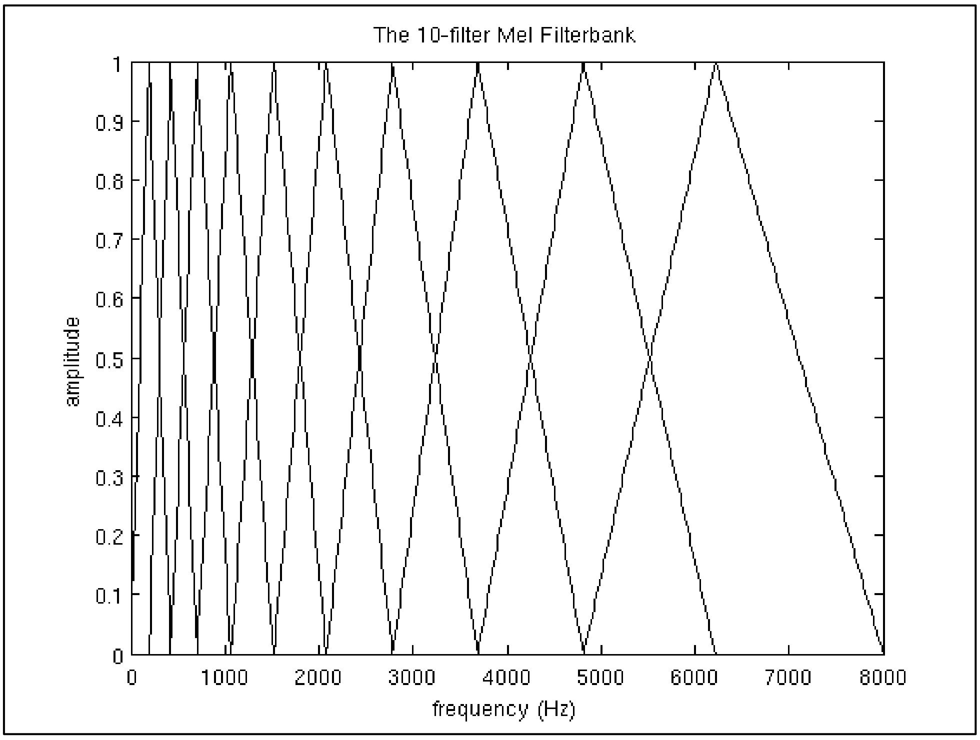

Fig 4.2 ‘mel’ filterbank [38]

The next step is to take the logarithm of the filterbank energies. The last step is to compute Discrete Cosine Transform (DCT) of the log filterbank energies. A DCT is a fourier related transform similar to the Discrete Fourier Transform (DFT), but it differs from DFT in the sense that it uses real numbers [37]. It has a property of energy packing or compaction. Also, it decorrelates the overlapping filterbank energies. First 13 coefficients are retained because the increasing number of coefficients represent faster change in the estimated energies and thus have less information for classifying audio signals [37].

4.4 Comparison of the five algorithms

The five algorithms were run on the 150 pairs of songs (50 pairs per set) successively, and the average distance was calculated as shown in Table 4.1.

From the table, it is clear that DTW works exceptionally well in distinguishing the three sets of data as it gives a very less value of edit distance for Case2 (0.0564) while a significant value for random pair (4.5391). If we look at Fig 4.3, it is clear that DTW distances for same song with different length pair (red plot) is much lesser than the plagiarized song pairs (blue plot), which in turn, even lesser than random song pairs (green plot).

Table 4.1 Average distance measures of five algorithms for plagiarism detection

Similarity Measure	Case 1: Pair of plagiarized songs	Case 2: Same song with different lengths pair	Case 3: A random pair of songs
DTW	2.3355	0.0564	4.5391
MSM	0.6856	0.0854	1.0718
ERP	0.6856	0.0854	1.0718
EDR	0.8000	0.1400	0.9800
LCSS	0.6723	0.9815	0.6369

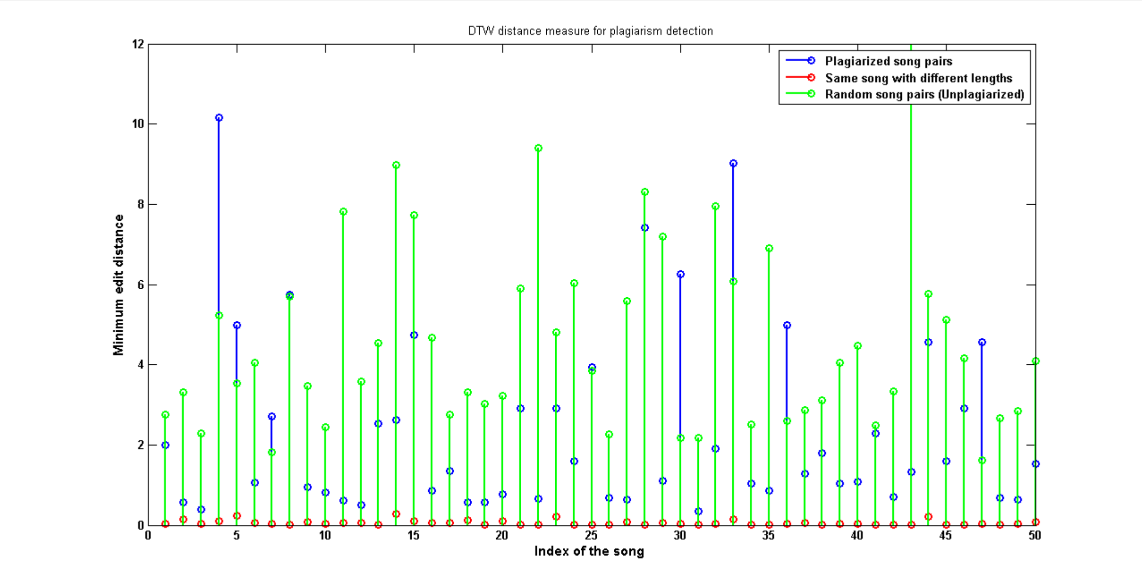

Fig 4.3 DTW distance for plagiarism detection

MSM is the second most effective measure in this case. It also differentiates among the three sets (0.0854 (Case 2) < 0.6856 (Case 1) < 1.0718 (Case 3)). However, the difference is lesser than that of DTW, so it may give negative results in some cases as shown in Fig 4.4. Here, though the distance for 'same song with different length pair' is considerably lower than the other two cases, but plagiarized pairs (blue plot) have, in some pairs (1, 4, 5, 7, 9, 10, 30, 36, 37, 40, 44, 47 as shown in Fig 4.4), more distance than the random pairs (green plot), which is not desirable.

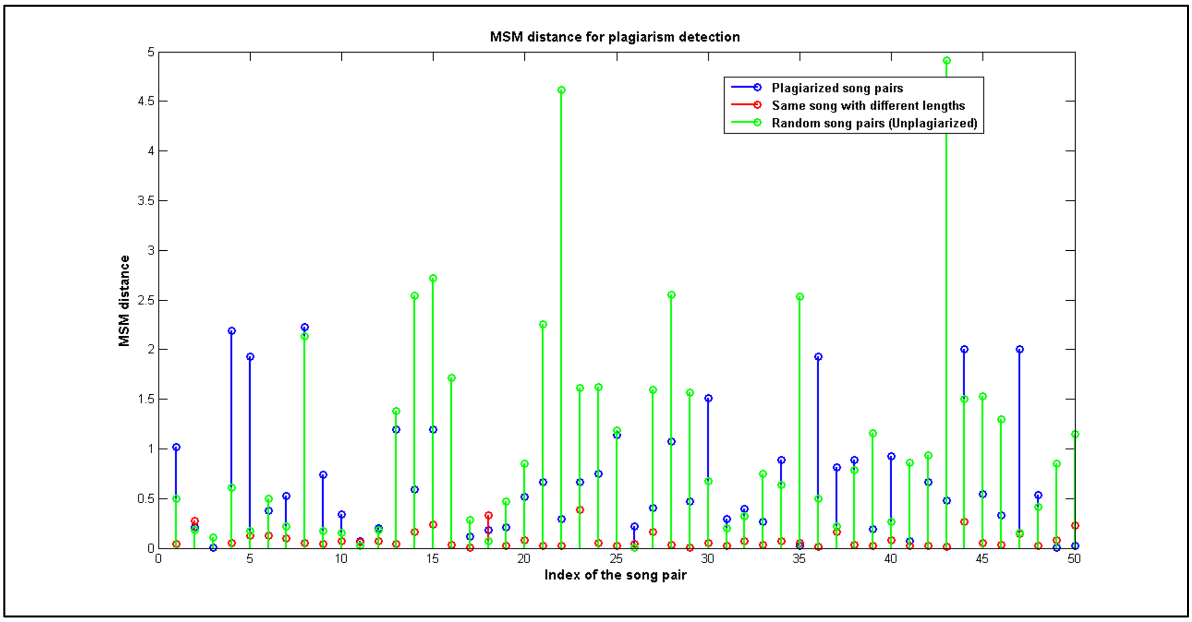

Fig 4.4 MSM distance for plagiarism detection

ERP gives the same results as MSM (as shown in Fig 4.5), because the cost value used in MSM makes it equivalent to ERP.

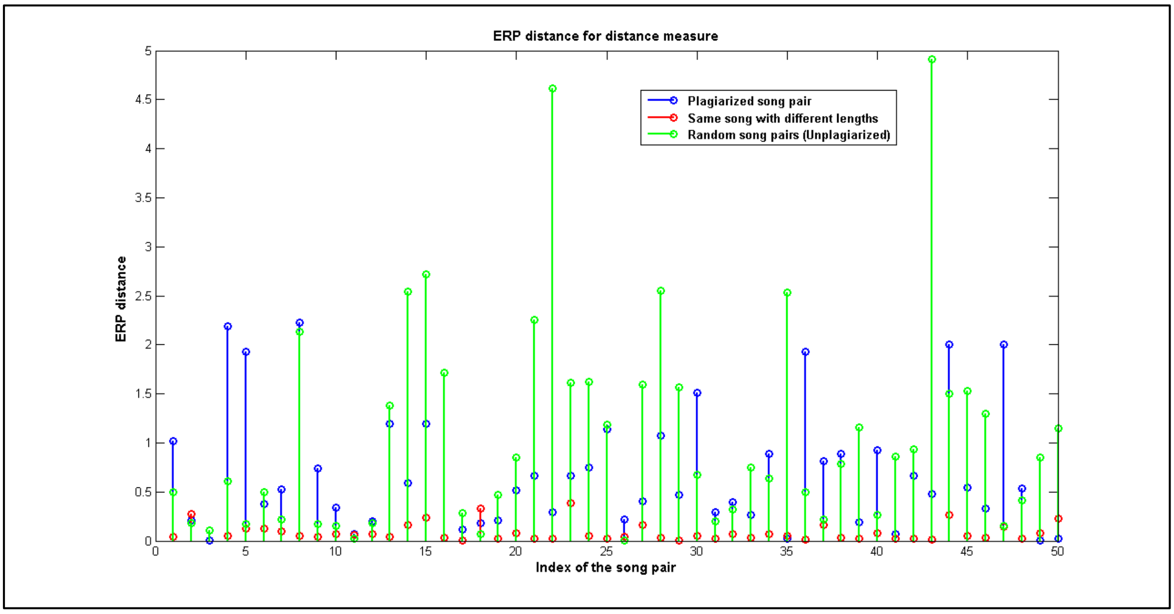

Fig 4.5 ERP distance for plagiarism detection

In EDR and LCSS, we have used a matching threshold of δ = 0.2 because the MFCC coefficients are real numbers and not a sequence of alphabets, which can be compared directly.

As the results show in Fig 4.6, LCSS gives the length of the longest common sequence. Thus, the order of values should be opposite to the previous three cases, i.e., $LCSS_{CASE\ 3} < LCSS_{CASE\ 1} < LCSS_{CASE\ 2}$. The average values in Table 4.1 gives the desired results (0.6369 < 0.6723 < 0.9815).

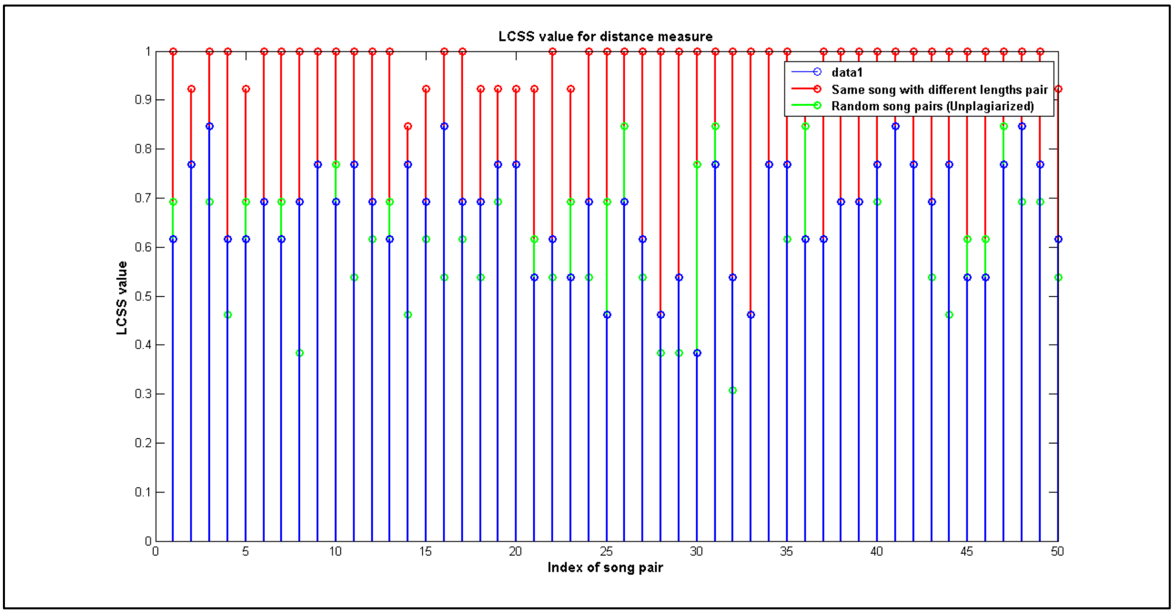

Fig 4.6 LCSS distance measure for plagiarism detection

EDR does not give the promising results, as the distance values for the three cases are quite close to each other as shown in Table 4.1 and Fig 4.7. Thus, it will not be able to clearly differentiate among the plagiarized, unplagiarized, and same song pairs.

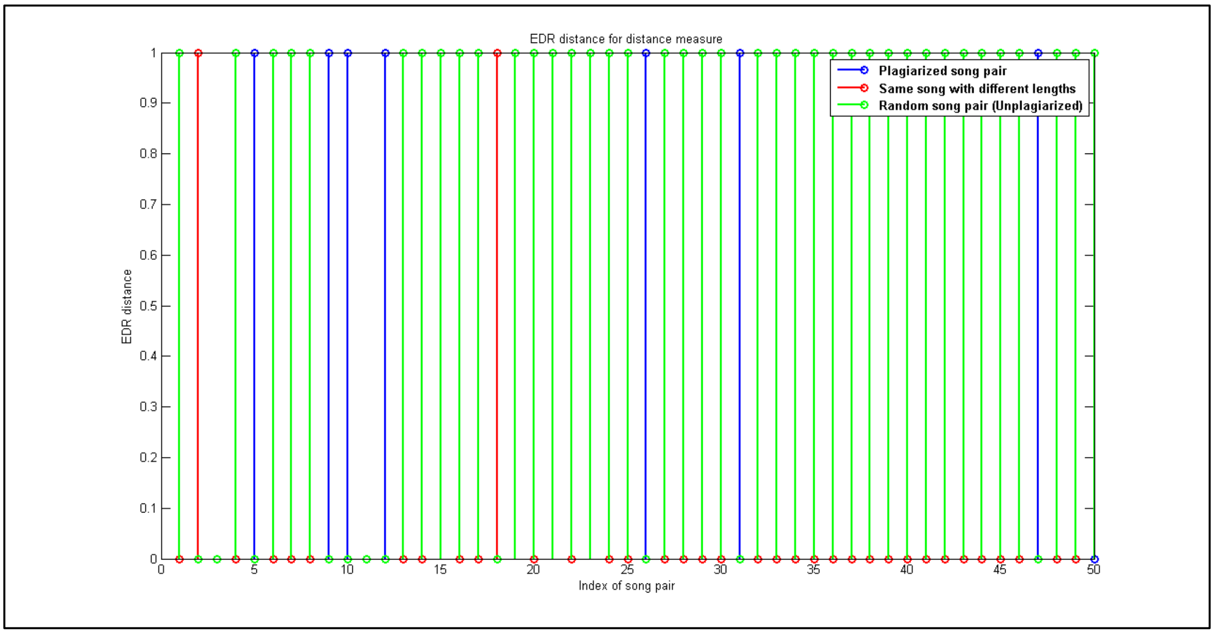

Fig 4.7 EDR distance measure for plagiarism detection

5 Hardware based Design of Similarity Search Algorithms

5.1 Experimental setup

In this chapter, experimental results of the work done have been presented. The five algorithms were first implemented in MATLAB . Then, they were designed for hardware implementation using Verilog hardware description language in Xilinx (Verilog codes are given in Appendix B). The device used was xc3s400-pq208. The algorithms were then simulated using ModelSim to confirm their accurate functioning.

5.2 Matlab and ModelSim Simulations

In this section, the simulation results of the five algorithms (DTW, LCSS, EDR, ERP and MSM) have been presented to confirm their proper functioning.

5.2.1 DTW simulations

Let the two sequences to be compared are A = [1 2 3 4 7 8] and B = [1 1 2 2 3 3 3 4].
For ModelSim simulations, they were stored in array1 and array2 as the binary numbers. The accumulated DTW distance matrix was stored row wise in the form of a linear array 'test', so that the last element of the array (test[47]) is DTW (6,8) = 011001, a binary number (as shown in Fig 5.1).
The MATLAB simulations show the same results as shown in Fig 5.2.

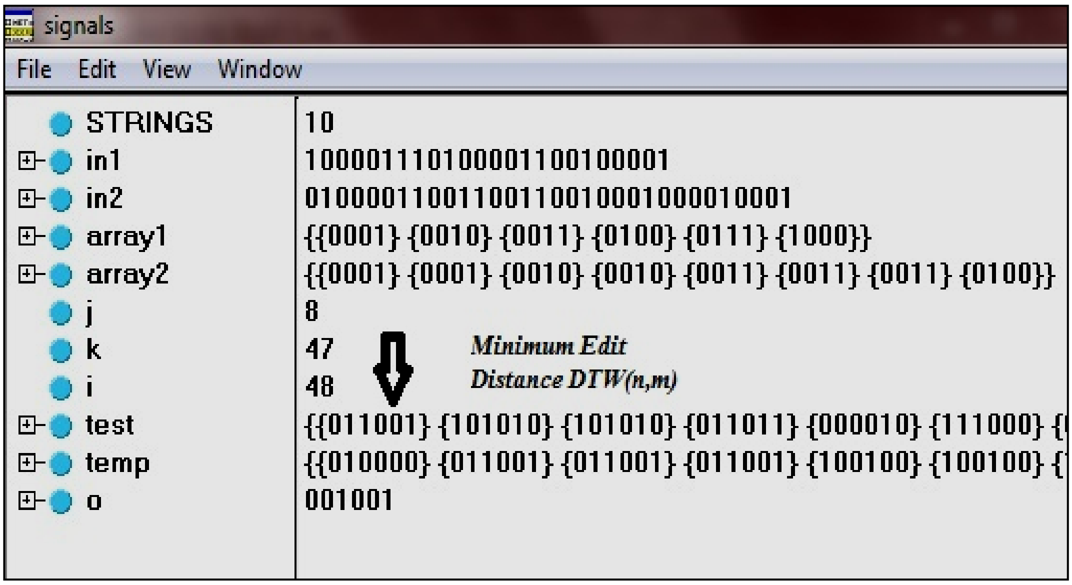

Fig 5.1 ModelSim simulations of DTW

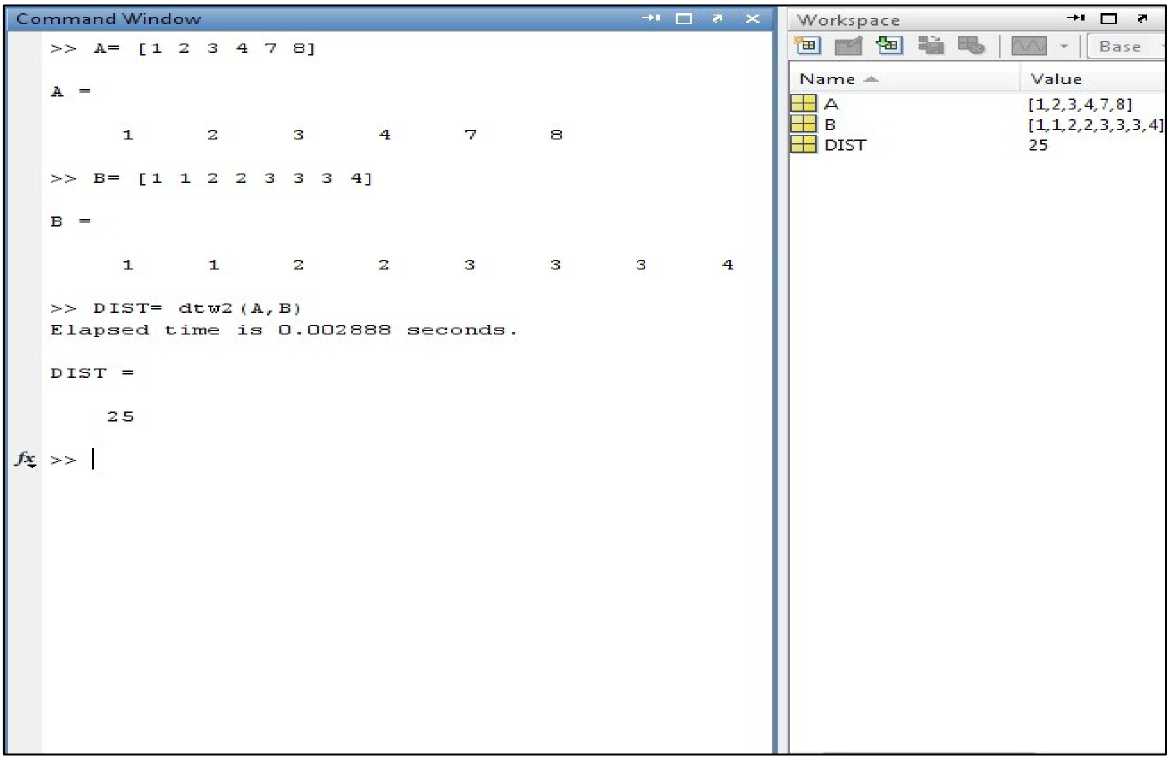

Fig 5.2 MATLAB simulations of DTW

5.2.2 LCSS simulations

Let the two sequences to be compared are A = [1 2 6 3 7 5] and B = [6 2 1 2 7 6 5]. For ModelSim simulations, the two sequences were stored as binary numbers in array1 and array2 as shown in Fig 5.3. The accumulated distance matrix is stored as a linear array 'test' whose last element is the last element of the matrix, thus the length of the longest common subsequence, i.e. test [41] = LCSS(6, 7) = 100, a binary number. MATLAB simulations show the same result (as shown in Fig 5.4).

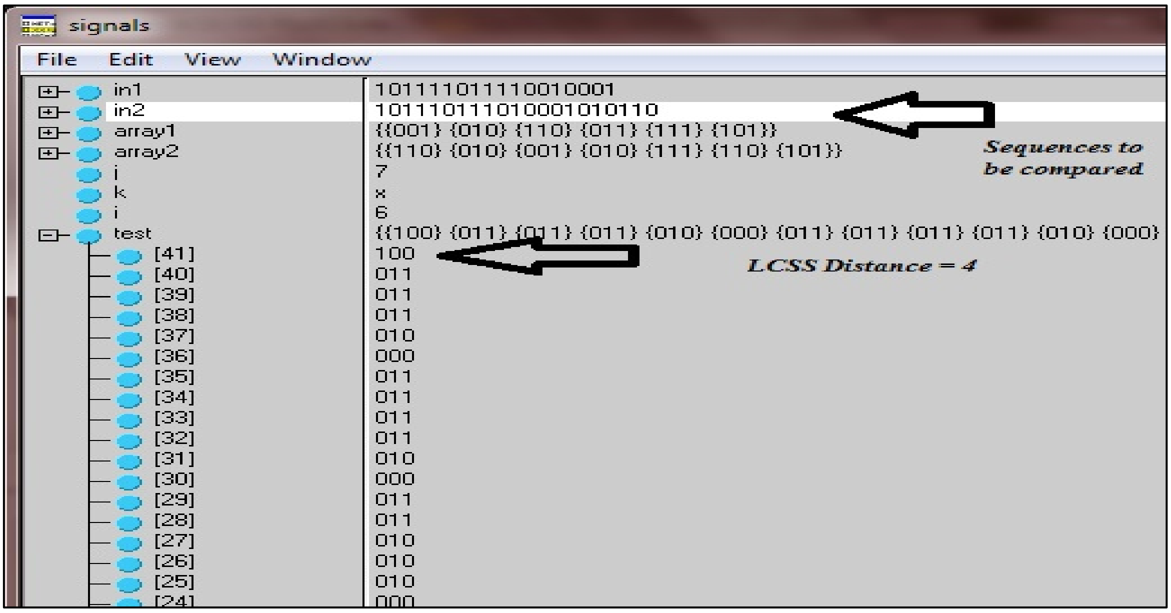

Fig 5.3 ModelSim simulations of LCSS

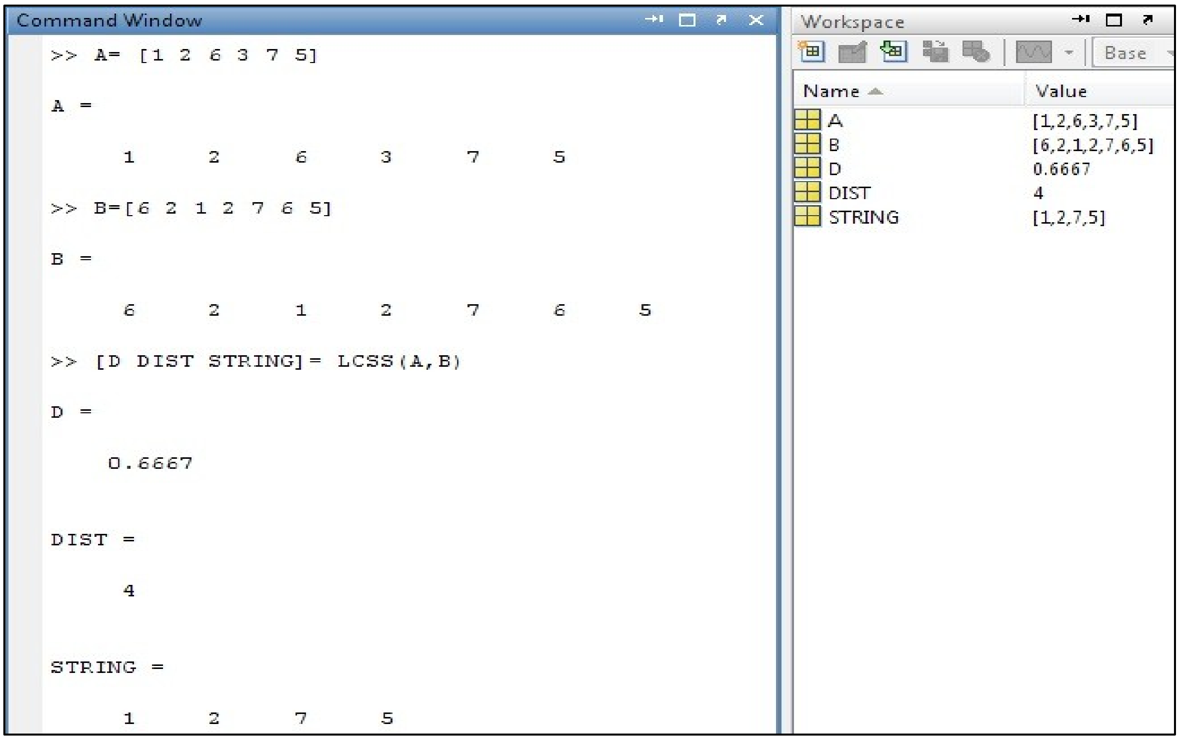

Fig 5.4 MATLAB simulations of LCSS

5.2.3 EDR simulations

Let the two sequences to be compared be A = [1 1 1 1] and B = [2 2 2 2].

They are stored in array1 and array2 as binary numbers and the elements of the accumulated distance matrix are stored in a linear array ‘test’. The minimum edit distance is given by the last element of test i.e. test [15] = 100, a binary number (as shown in Fig 5.5). MATLAB simulations sow te same result as shown in Fig 5.6.

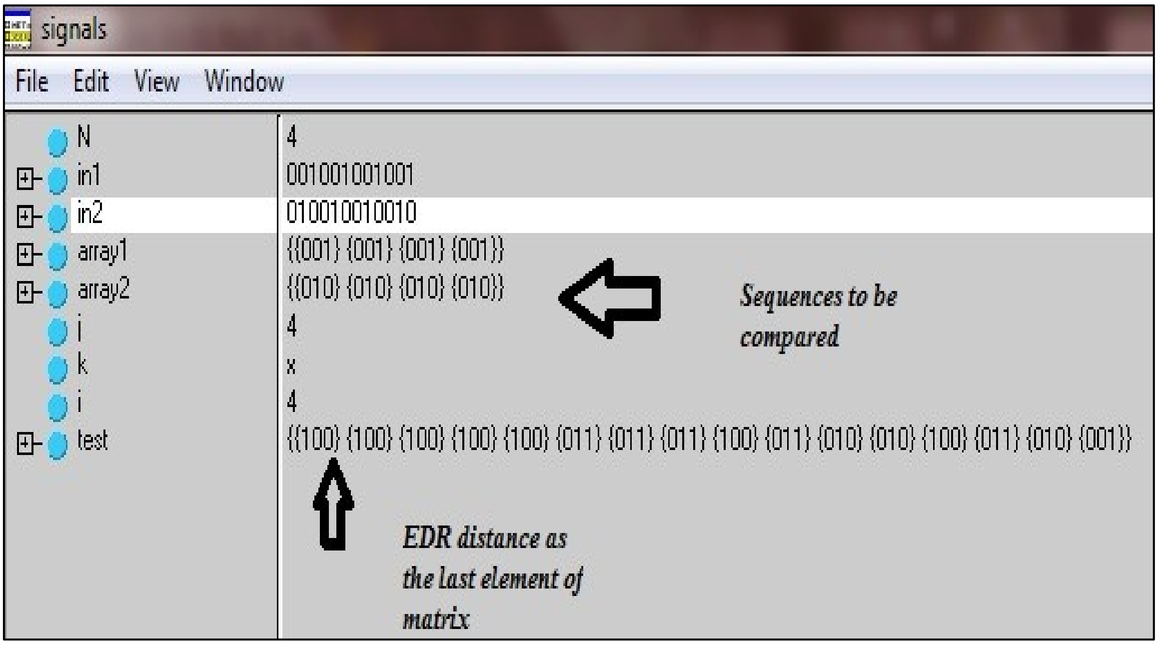

Fig 5.5 ModelSim simulations of EDR

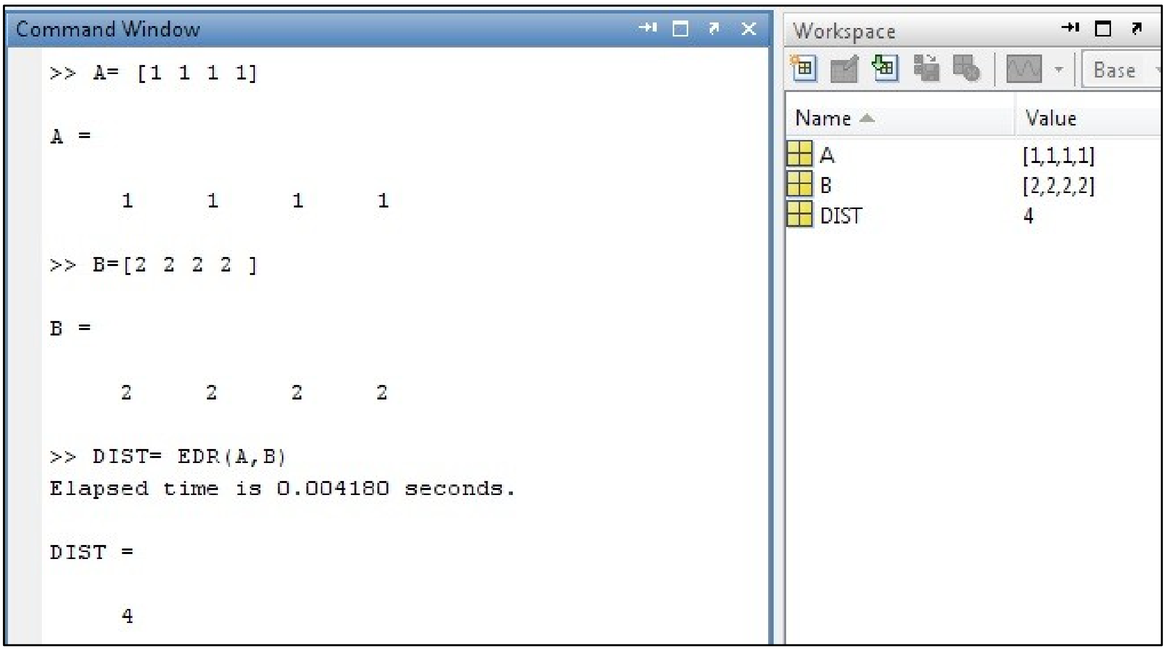

Fig 5.6 MATLAB simulations of EDR

5.2.4 ERP simulations

Let the sequences to be compared are A = [1 2 3 3 4 6] and B = [1 1 1 2 2 2]. For ModelSim simulations, they are stored in array1 and array2 in the form of binary numbers (as shown in Fig. 5.7). The minimum edit distance is stored in the last element of the linear array ‘test’, i.e. ERP (A,B) = test [35] = 1010, a binary number.

MATLAB simulations show the same results (as shown in Fig. 5.8).

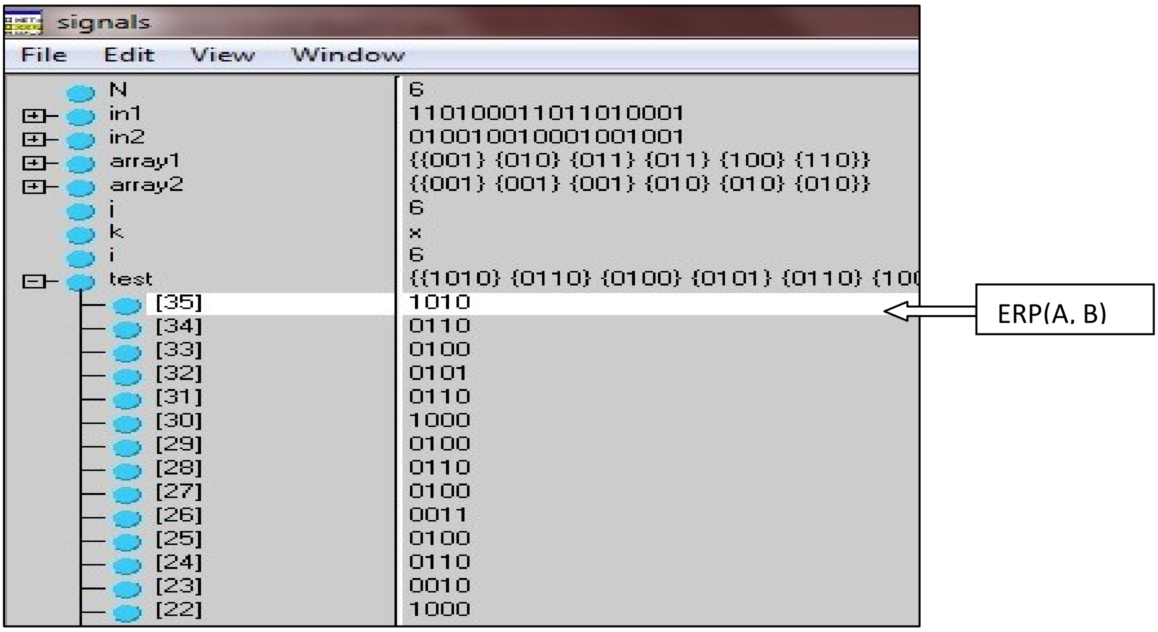

Fig 5.7 ModelSim simulations of ERP

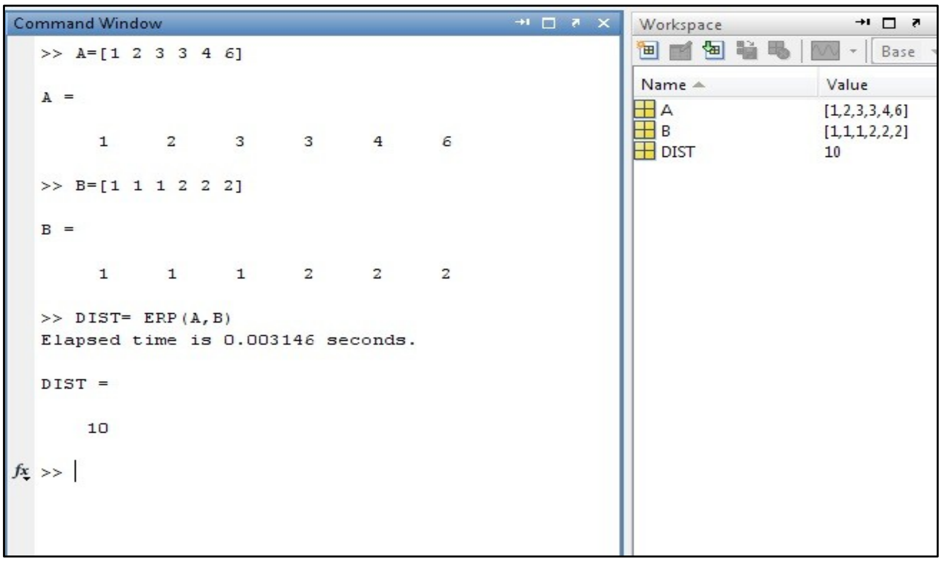

Fig 5.8 MATLAB simulations of ERP

5.2.5 MSM simulations

Let the sequences to be compared are A = [1 2 3 4 5] and B = [0 1 1 2 3]. For ModelSim simulations, they are stored in array1 and array2 in the form of binary numbers (as shown in Fig. 5.9). The minimum edit distance is stored in the last element of the linear array 'test', i.e. MSM (A,B) = test [24] = 1000, a binary number.

MATLAB simulations show the same results (as shown in Fig. 5.10).

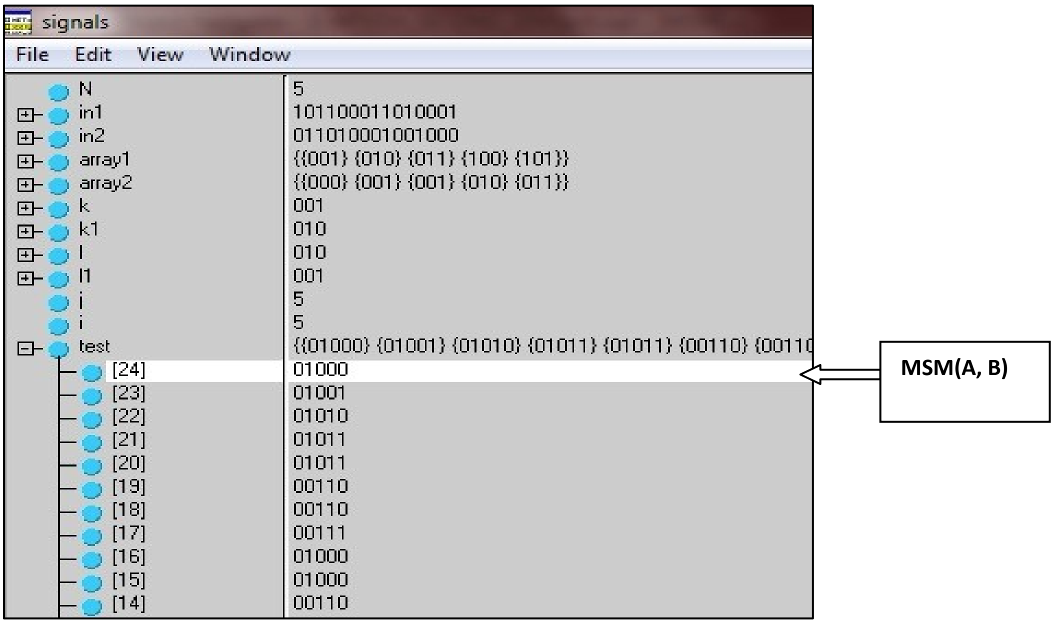

Fig 5.9 ModelSim simulations of MSM

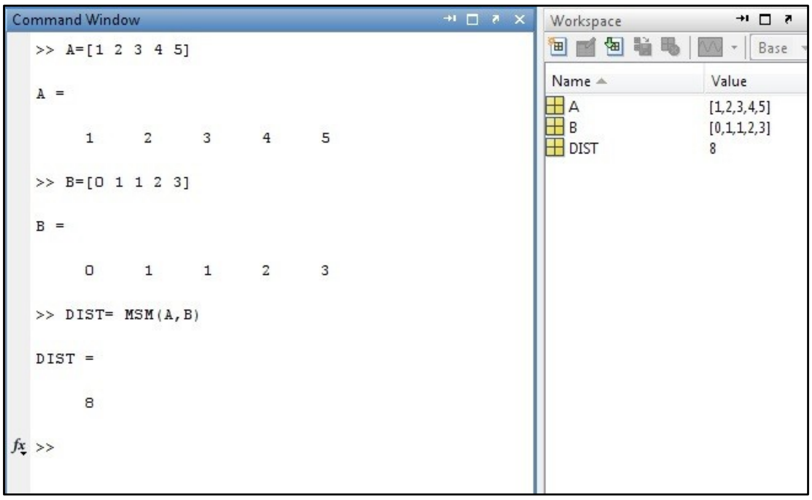

Fig 5.10 MATLAB simulations of MSM

5.3 Synthesis results in Xilinx

The system is implemented using Xilinx Integrated Software Environment (ISE) which is used to synthesize and process HDL algorithms on FPGA. Target device Spartan3 with package pq208 has the following properties: 3584 Slices, 7168 Look- Up Tables (LUTs), 141 Input/ Output Blocks (IOBs). The input is taken as an array of binary numbers. The design of an FPGA configuration requires a hardware description language

(VHDL/Verilog). In this work, we have used verilog for the programming of the processing units.

5.3.1 Hardware utilization analysis of implemented Similarity Measures

In the synthesis of DTW with input sequences of length 8, the hardware utilization in terms of slices, LUTs and IOBs is 32.7%, 46.7% and 49.6% respectively of the available resources. However, number of slices and LUTs increase exponentially with the increase in input size as shown in Fig 5.11 and Fig 5.12 respectively; and IOBs increase linearly as shown in Fig 5.13. LCSS has 8.06%, 7.07% and 34.7% utilization of the resources for the same input sequence, which is considerably lower than DTW (Exact values of the usage are given in Appendix D). EDR also shows satisfactory results with 14.0%, 12.6% and 34.7% resource utilization. However, ERP and MSM do not give promising results as they require complex computations, thus extra hardware resources.

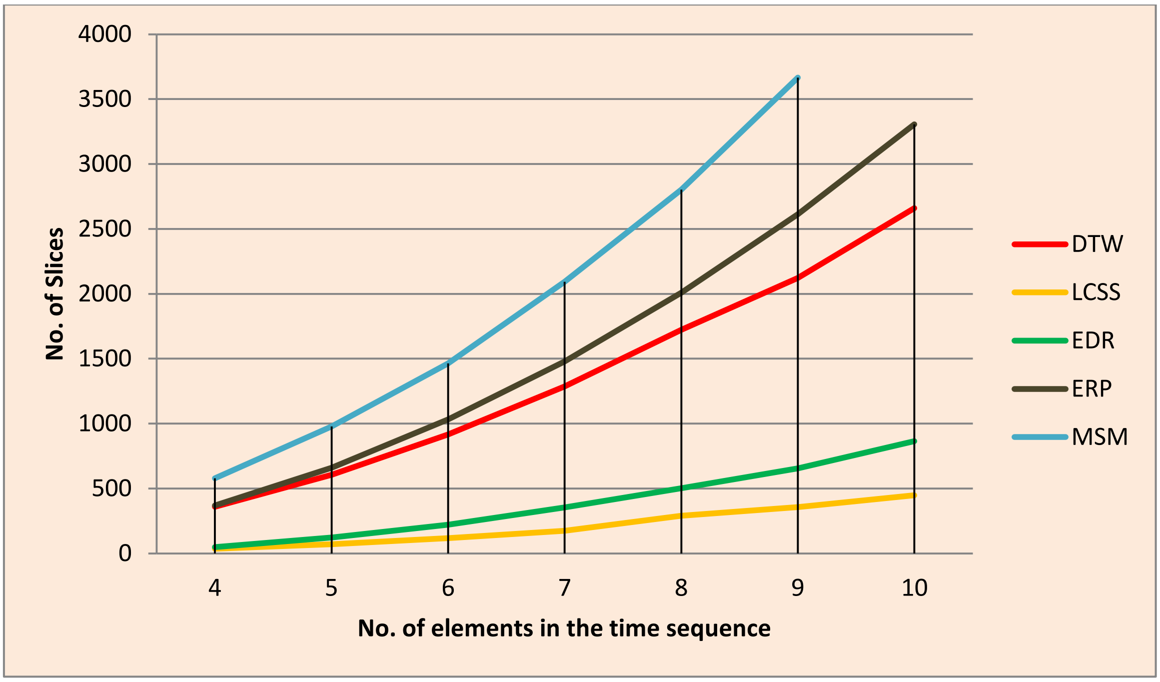

Fig 5.11 Slices utilization versus no. of elements in sequence

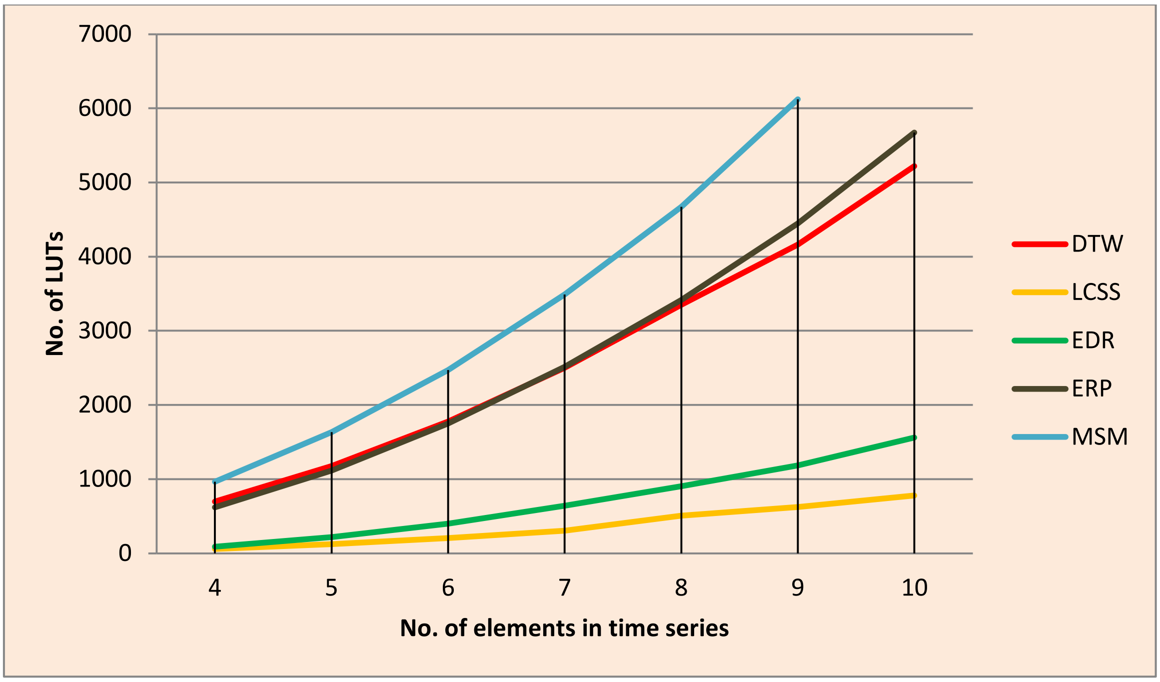

Fig 5.12 LUTs utilization versus no. of elements in sequence

In all the five algorithms, the pattern of change is same for slices and LUTs, i.e. exponential. However, EDR and LCSS perform exceptionally well in hardware utilization. MSM does not give satisfactory results as it requires extensive computation.

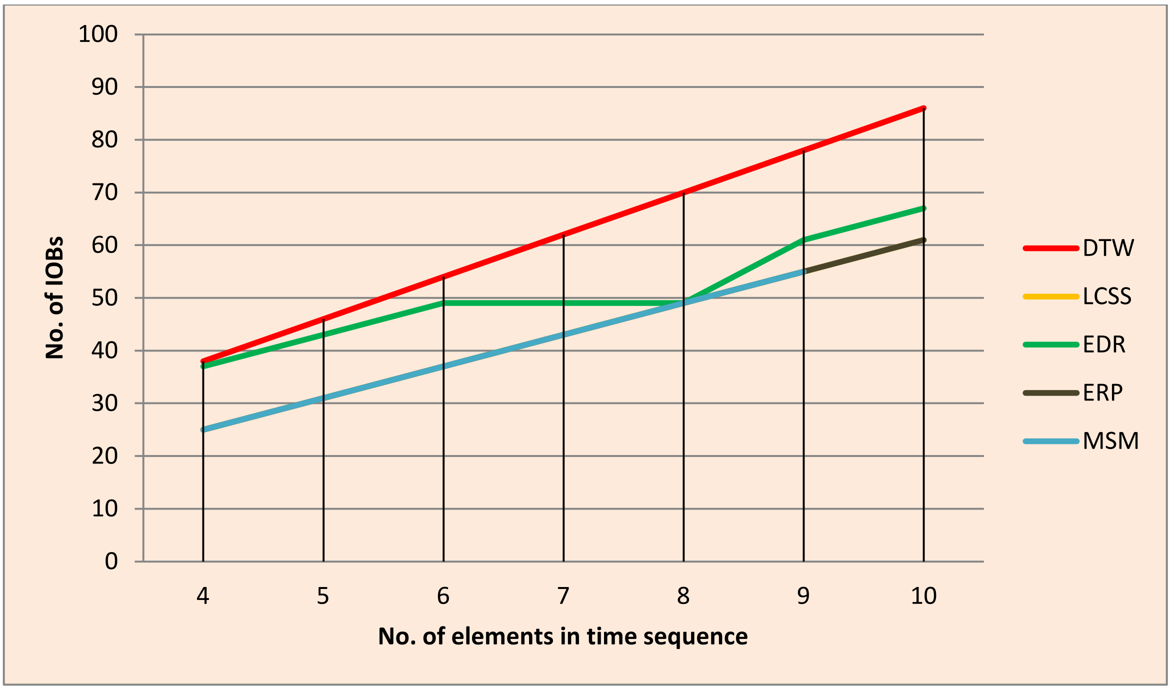

Fig 5.13 IOBs utilization versus no. of elements in sequence

Again all the five algorithms' IOBs utilization varies linearly with the number of elements. LCSS, ERP, and MSM have exactly similar IOBs utilization, EDR has a slight variation from linearity for n= 6 to 9, while DTW show perfect linear characteristics.

5.3.2 Timing analysis of implemented similarity measures

Delay is another critical parameter in real time systems. From the delay point of view, again LCSS outperforms other four algorithms as shown in Fig 5.14. The best point is they all are showing linear variation of delay with the number of elements in sequence. Thus, the hardware can be designed for relatively large sequences.

Fig 5.15 shows the variation of execution time of these similarity measures with the number of elements in sequences in MATLAB. Here, LCSS, as well as, DTW perform satisfactorily. However, the hardware design shows a considerably high speed up over software implementation. Also, the execution time variation in MATLAB is not linear, but slightly exponential, which is not desirable for long length sequences.

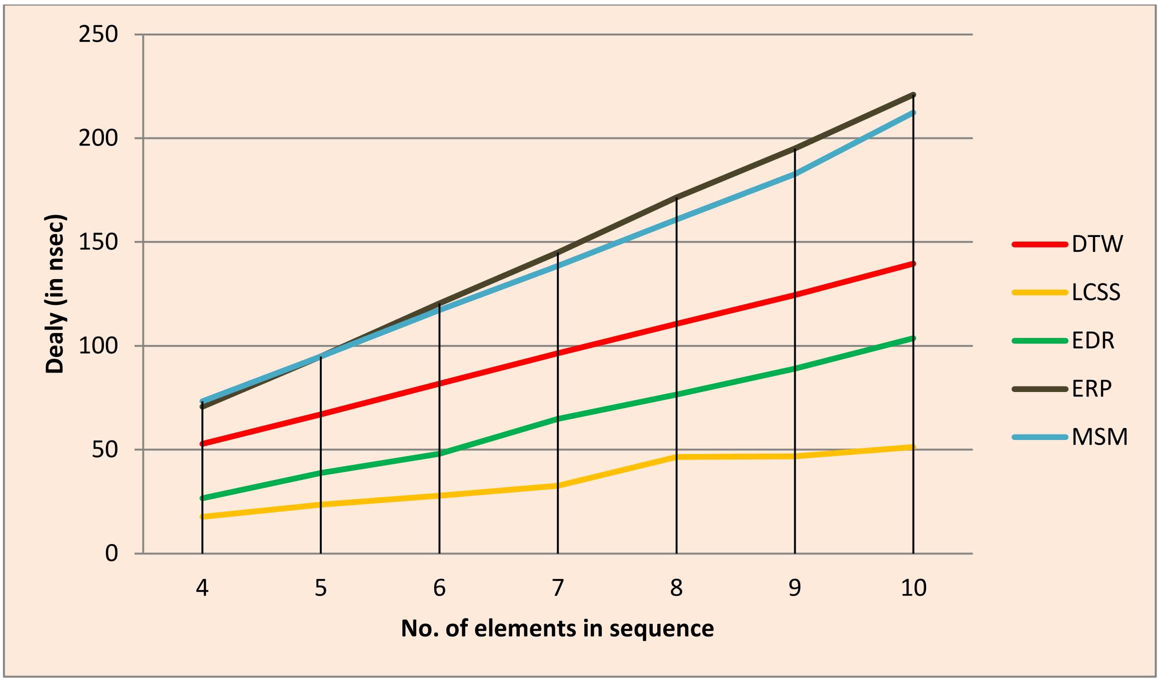

Fig 5.14 Delay versus no. of elements in sequence

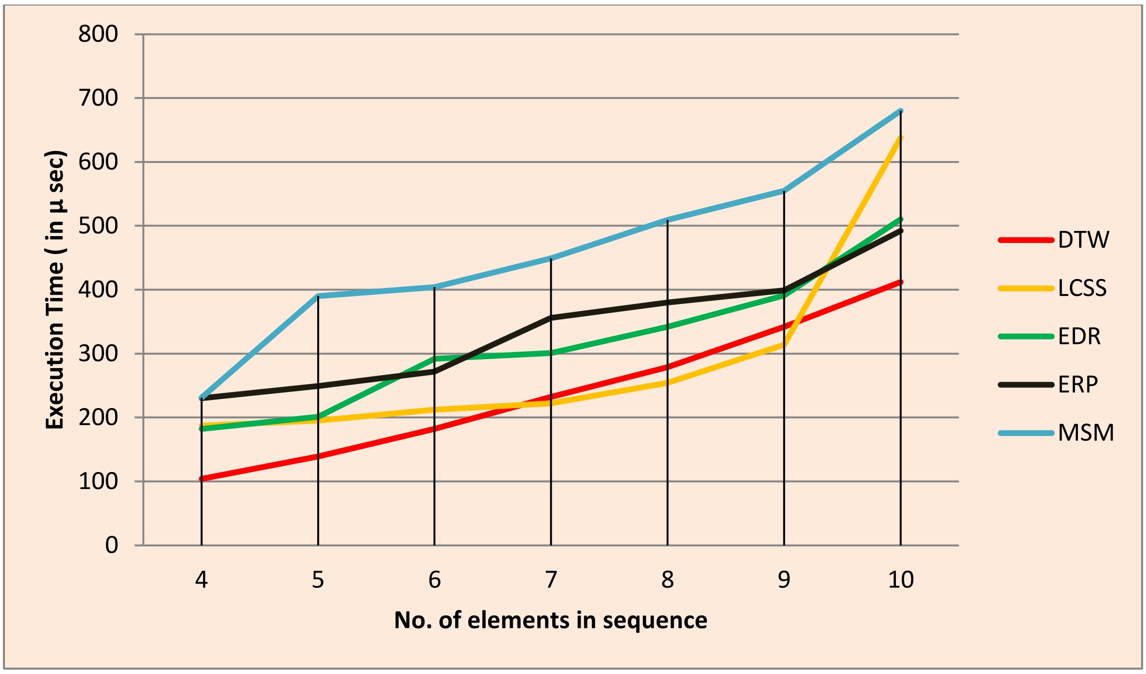

Fig 5.15 Software execution time versus no. of elements

6 Conclusions and Future Directions

6.1 Conclusion

Time series are woven into the fabric of everyday life. There are a number of software algorithms available to process time series for information retrieval. Hardware design of software algorithms is a subject of critical research as it provides an appealing choice of using them in real time applications.

In this project, we have chosen five critical similarity measures (DTW, LCSS, EDR, ERP, and MSM). DTW between two time series does not require the two series to be of same length, and it allows the local time shifting by repetition of elements. ERP uses a constant value for computing the distance for gaps and L_1 norm for non gap elements. LCSS gives the match reward of 1 if the elements are same and no reward if they fail to match. The EDR technique uses gap and mismatch penalties. A comparison of these five algorithms on the basis of certain properties has been shown in Table 6.1. We proposed their hardware design based on FPGA with a comparative analysis on the basis of hardware utilization and delay.

We show that despite being a non metric, LCSS provides the best solution for real time processors in the similarity retrieval of time series as seen in Fig 6.1. In terms of delay (logic + route), LCSS is again better than any other similarity measure (Fig.6.2).

Table 6.1 Comparison of the properties of similarity measures

Distance Function	Handling local time shifting	Different lengths	Noise	Requirement of matching threshold	Metri -city	Matching alphabets as well as symbols
DTW	YES	YES	NO	NO	NO	NO
LCSS	YES	YES	YES	YES	NO	YES
EDR	YES	YES	YES	YES	NO	YES
ERP	YES	YES	NO	NO	YES	NO
MSM	YES	YES	NO	NO	YES	NO

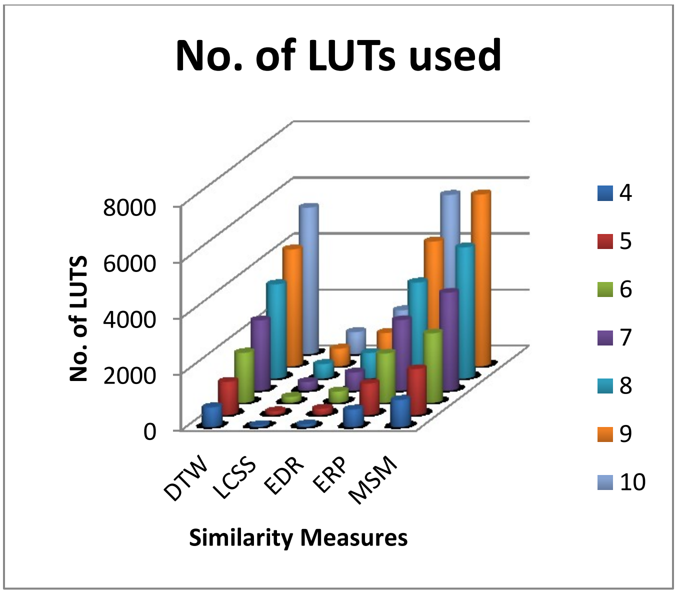

Fig 6.1 LUTs utilization summary for implemented similarity measures

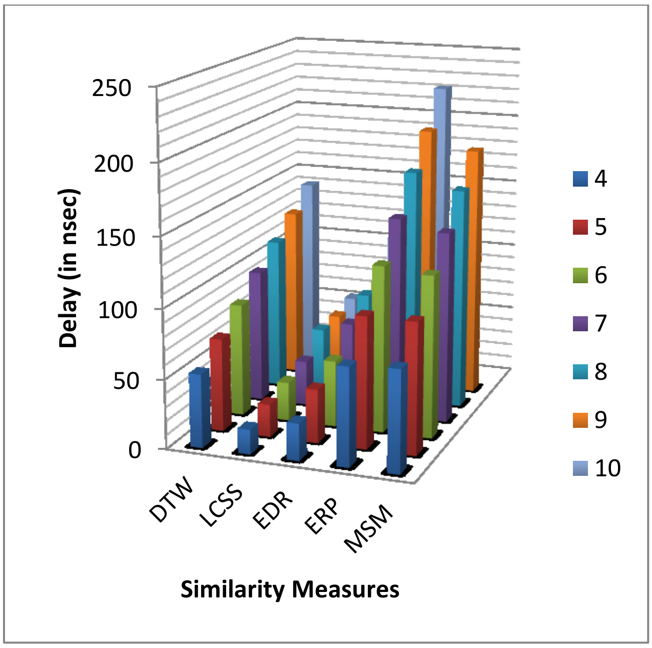

Fig 6.2 Delay variation summary for implemented similarity measures

For a device to be used in real time applications, it should be time efficient. Since we are suggesting the hardware implementation of the software algorithms, a comparison of time between the hardware and software execution is required. Thus, all the five similarity measures were implemented in MATLAB R2009a with Intel Core2Duo processor on PC and the results clearly show the speed up of hardware units over the software. For instance, when the two sequences have 4 elements each, on FPGA implementation of DTW to compute the distance between them, the minimum delay is 52.8 nsec which corresponds to the maximum frequency of 18.9 MHz, while the minimum execution time in MATLAB is 104 µsec which corresponds to the maximum frequency of 9.6 kHz.

The unification of the efficiency of these algorithms with the amenity of FPGA provides an excellent solution for leading edge speech processing applications.

Although we did not produce running hardware, the hardware design was taken to a point where both the hardware and timing requirements for the implementation can be accurately predicted. It can be concluded that this designing process has a vast scope in the further development of these distance measures as an integral part of real time processors.

DTW has been used a number of times in different applications related to time series, especially speech processing. To ensure the efficacy of other four algorithms, we have implemented an application in MATLAB. We chose three sets of songs, each containing 100 songs (50 pairs). Set 1 had plagiarized songs, set 2 had the same song with different lengths and set 3 had pairs of non plagiarized songs. We applied all the five distance measures to distinguish among the three sets. The results clearly showed that DTW outperforms among the five algorithms (DTW, LCSS, EDR, ERP, and MSM) as the average distance for set1, set2 and set3 were 2.3355, 0.0564 and 4.5391 which are considerably far apart to ensure the detection of plagiarism.

6.2 Future directions

There are a number of issues, in this area, which require further research. One disadvantage of the introduced processing units is they cannot handle very large dimensions because of the limitation of hardware resources. The hardware performance of LCSS, EDR, DTW, ERP and MSM (in decreasing order) is a significant indicator of their remunerative nature of being feasible for complex hardware designs based on FPGAs. Future work includes the hybridization of two or more algorithms discussed above to inculcate all the properties of being efficient. Another problem with the above mentioned technique to implement these similarity measures is its quadratic computational complexity. So, with the increase in the length of the data sequences, they become more and more computationally intensive. The future work includes the segmentation of long sequences into smaller ones and then applying these measures with cheap approximations. DTW is a benchmark algorithm in speech processing and has a potential to be used as an independent unit in embedded systems for music retrieval, plagiarism detection etc. So, there is a need to improve upon its hardware utilization which can be done by adapting a fast and efficient multiplication algorithm. The future work also includes integration of actual FPGA board with performance measurement and finding hardware design solution for handling large stream-in data sets. The above mentioned similarity measure algorithms are software algorithms, where DTW performs better than LCSS and EDR. But, in hardware, LCSS has a better performance. Thus, to bring the software and hardware implementations on the same track and to develop a new hybrid algorithm having the best features of all the above five algorithms would be the part of our future investigation.

FPGAs require reducing the cardinality/ precision of the data. In modern FPGAs, the floating point arithmetic does not scale well with larger applications due to the additional complexity for handling the mantissa and exponent separately. Since, time series has real numbered values, there is a requirement for optimization techniques before their implementation on FPGA, the most important area of future research. Besides, to study and explore the influence of global constraints on EDR, ERP and MSM to reduce computational time and complexity is a part of future work.

REFERENCES

[1] C. Faloutsos, M. Ranganathan, and Y.Manolopoulos "Fast subsequence matcing in time-series databases", SIGMOID '94 proceedings of the ACM SIGMOID international conference on management of data, pp. 419-429, 1994.

[2] G. Yuying, J. Bin, and Z. Zhengwei "A fault diagnosis method based on DTW", IEEE Control Conference CCC 2006, pp. 1281-1284, Harbin, Aug. 2006.

[3] Tarar, S. "*Speech analysis:* Desktop items activation using dynamic time warping algorithm", 3rd IEEE international conference on Computer Science and Information Technology (ICCSIT)", pp. 657-659, vol. 6, July 2010.

[4] Chalmers, N., Glasgow,J., and Scott,S., "Dynamic Time Warping as a spatial assessment of sensorimotor impairment resulting from stroke", IEEE annual conference on Engineering in Medicine and Biology Society, EMBC, pp. 8235-8238, Sept. 2011.

[5] Petitijean, F., Inglada, J., and Gancarski, P. "Satellite image time series analysis under time warping" IEEE trans. on Geoscience and Remote Sensing, vol. 50, no.8, pp. 3081-3095, July 2012.

[6] Xingzhe, X., Philips, W., Veelaert, P, and Aghajan, H. "Road network inference from GPS traces using DTW algorithm" IEEE 17th international conference on Intelligent Transportation Systems (ITSC), pp. 906-911, Oct. 2014.

[7] D. Sart, A. Mueen, W. Najjar, E. Keogh and V. Niennattrakul, "Accelerating dynamic time warping subsequence search with GPUs and FPGAs," IEEE 10th International Conference on Data Mining, pp. 1001-1006, 2010.

[8] J.S. Tai, K.F. Li and H. Elmiligi, "Dynamic time warping algorithm: A hardware realization in VHDL," IEEE International Conference on IT Convergence and Security(ICITCS), pp. 1-4, December 2013.

[9] Madhavan. A, Sherwood.T, and Strukov.D " Race logic: A hardware acceleration for dynamic programming algorithms", 41st international symposium on Computer Architecture (ISCA), pp. 517-528, June 2014.

[10] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories" , 18th international conference on Data Engineering, pp. 673-684, 2002.

[11] L. Chen, " Similarity search over time series and trajectory data" a thesis presented for Doctor of Philosophy to the University of Waterloo, Canada, 2005.

[12] M. Elhadi and A. Al-Tobi, "Duplicate detection in documents and web pages using improved Longest Common Subsequence and documents syntactical structures", IEEE 4th international conference on Computer Sciences and Convergence Information Technology, ICCIT, pp. 679-684, Nov. 2009.

[13] R.A.C. Campos and F.J.Z. Martinez, " Batch source-code plagiarism detection using an algorithm for the bounded longest common subsequence problem" IEEE 9th international conference on Electrical Engineering, Computing Science and Automatic Control (CCE), pp. 1-4, Sept. 2012.

[14] J.Liaw, W. Wang, H. Chu, M. Huang and C. Lu, "Recognition of the ambulance siren sound in Taiwan by the Longest Common Subsequence" IEEE international conference on Systems, Man, and Cybernetics (SMC), pp. 3825-3828, Oct. 2013.

[15] E. Parvinnia, M. Taheri and K. Ziarati, " An improved Longest Common Subsequence algorithm for reducing memory complexity in global alignment of DNA sequence" IEEE international conference on BioMedical Engineering and Informatics (BMEI), vol. 1, pp. 57-61, May 2008.

[16] V. Kurbalija, M. Radovanovic, Z. Geler and M. Ivanovic " The influence of global constraints on DTW and LCS similarity measures for time-series databases" 3rd international conference on Software, Services and Semantic Technologies S3T, Advances in Intelligent and Soft Computing, vol. 101, pp. 67-74, 2011.

[17] L. Chen, M.T. Ozsu, and V. Oria., "Robust and efficient similarity search for moving object trajectories", In CS Tech. Report, CS -2003-30, School of Computer Science, University of Waterloo.

[18] L. Chen and R. Ng, "On the marriage of Lp-norms and edit distance," Proc. 30th international conference on Very Large Data Bases (VLDB), vol. 30, pp.792-803, 2004.

[19] A. Stefan, V. Athitsos, and G. Das, "The Move-Split-Merge metric for time series," IEEE transactions on Knowledge and Data Engineering, vol.25, no.6, pp. 1425-1438, June 2013.

[20] F.Seifert and W.Benn, "Semantic relationship and identification of music", IEEE 3rd international conference on Web Delivering of Music, pp. 85-92, Sept. 2003.

[21] S. De, I. Roy, T. Prabhakar, K. Suneja, S. Chaudhuri, R. Singh and B. Raj, "Plagiarism detection in polyphonic music using monaural signal separation" ISCA's 13th annual conference on INTERSPEECH, pp. 1744-1747, Sept. 2012.

[22] C.Dittmar, K.F. Hildebrand, D. Gaertner, M. Winges , F. Muller and P. Aichroth, " Audio forensics meets Music Information Retrieval- A toolbox for inspection of music plagiarism" IEEE proc. Of 20th European Signal Processing Conference (EUSIPCO), pp. 1249-1253, Aug. 2012.

[23] Xilinx datasheet of Spartan-3 FPGA Family, DS099, June 2013 www.**xilinx**.com/support/documentation/**data_sheets**/ds099.pdf

[24] R.J. Francis, "A tutorial on logic synthesis for LookUp –Table based FPGAs", IEEE/ ACM international conference on Computer Aided Design (ICCAD), pp. 40-47, 1992.

[25] P.P.Chu, "FPGA prototyping by VHDL examples" Xilinx Spartan-3 Specific Memory, Chapter-11, pp.243-256, John Wiley, 2008.

[26] Esling P, Agon C, “Time- Series Data Mining ,” ACM Computing Surveys, 45(1), 12:1 – 12:34, 2012.

[27] Yi B.K, Faloutsos C, “ Fast Time Sequence Indexing for Arbitrary Lp Norms,” In Proceedings of the 26th International Conference on Very Large Data Bases, VLDB, pp. 385-394, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN 1-55860715-3.

[28] Wang X, Mueen A, Ding H, Trajcevski G, Scheuermann P, Keogh E, “Experimental comparison of representation methods and distance measures for time series data,” Data Mining and Knowledge Discovery, 26(2),pp. 275-309, 2012.

[29] Golay X, Kollias S, Stoll G, Meier D, Valavanis A, Boesiger P, “ A new correlation based fuzzy logic clustering algorithm for FMRI,” Magnetic Resonance in Medicine, 40(2), pp. 249-260 , 1998.

[30] Liao TW, “Clustering of Time Series Data, a Survey,” Pattern Recognition, 38(11),2005, pp.1857-1874.

[31] Dina Q. Goldin, Paris C. Kanellakis, “On similarity queries for time-series data: Constraint specification and implementation”, 1st international conference on Constraint Programming (CP), Principles and Practice of Constraint Programming book-CP’95, pp. 137-153, Springer Berlin Heidelberg, Sept. 1995.

[32] T.F. Smith and M.S Waterman, “Identification of common molecular subsequences”, Journal of Molecular Biology, vol. 147(1), pp. 195-197, March 1981.

[33] K. Suneja and M. Bansal, “Hardware design of similarity measures for time series based on FPGA in Verilog”, proc. of 2nd international conference on VLSI, Communication and Networks (VCAN), April 2015.

[34] K. Suneja and M. Bansal “Hardware design of Dynamic Time Warping algorithm based on FPGA in Verilog,” International Journal of Advanced Research in

Electronics and Communication Engineering (IJARECE), vol.4, issue 2, pp. 165-168, February 2015.

[35] M.Morse and J.M. Patel, "An efficient and accurate method for evaluating time series similarity" SIGMOD, Beijing, China, June 2007.

[36] O. Lartillot MIR toolbox 1.3.2 User's manual, University of Jyvaskyia, Finland, January 2011.

[37] S. Gupta, J. Jaafar, W.F. Ahmad and A. Bansal " Feature extraction using mfcc" international journal on Signal and Image Processing, vol.4, no.4, Aug. 2013.

[38] Mel Frequency Cepstral Coefficient (MFCC) tutorial file:///E:/MTECH_FOUTH_SEM/thesisis/References/18_chapter_8_mfcc.htm

[39] S.B. Davis and P.Mermelstein "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences" IEEE trans. on Acoustics, Speech, and Signal Processing, no.4, Aug 1980.

[40] Dongyu Z., W. Zuo, David Z., Hongzht Z. and N. Li "Classification of pulse waveforms using edit distance with real penalty", EURASIP Journal on Advances in Signal Processing, Hindawi publishing corporation, Aug 2010.

[41] F. Afonso and F. Barbosa "Trajectory data similarity with metric data structures" conference proc. Of Geographical Information Systems Research UK (GISRUK), 2011.

[42] "Longest common subsequence" web source, wikipedia http://www.algorithmist.com/index.php/Longest_Common_Subsequence

[43] Frentzos E, Gratsias K, Theodoridis Y, " Index- based most similar trajectory search," IEEE 23rd International Conference on Data Engineering(ICDE), pp. 816-825, 2007.

[44] C.D. Meliza, S.C. Keen, and D.R. Rubenstein, "Pitch- and spectral-based dynamic time warping methods for comparing field recordings of harmonic avian vocalizations" Journal of the Acoustical Society of America, 134(2), pp. 1407-1415, Aug 2013.

BOOKS:

[1] R. Woods, J. McAllister, G. Lightbody and Y. Yi, " FPGA- based implementation of signal processing systems" John Wiley and Sons, Ltd, publication, 2008.

[2] S. Palnitkar, "*Verilog HDL* – A guide to digital design and synthesis", Sun Soft Press publication, 1996.

Appendix A

MATLAB code for the detection of plagiarism in music

```
clear all
close all
for i=1:50

    a = strcat(int2str(i),'1');
    a1 = miraudio(a);
    b1 = miraudio(a,'extract',-3,+3,'middle');
    f1 = mirframe(a1,'length',2) ;


%Feature02 mfcc
mfcc1=mirgetdata(mirmfcc(a1))
mfcc2=mirgetdata(mirmfcc(b1))
l1=strcat(a,'-mfcc.mat');
l2=strcat(a,'-newmfcc.mat');
save(l1,'mfcc1');
save(l2,'mfcc2');


   end
    for i=1:50


    t7=importdata(strcat(int2str(i),'1','-mfcc.mat'))


    r7=importdata(strcat(int2str(i),'1','-newmfcc.mat'))
      d0(2,i)=dtw2(t7,r7);

      d1(2,i)=MSM(t7,r7);

      d2(2,i)=ERP(t7,r7);

      d3(2,i)=LCSS(t7,r7);

      d4(2,i)=EDR(t7,r7);

    save('distance0.mat','d0');
    save('distance1.mat','d1');
    save('distance2.mat','d2');
    save('distance3.mat','d3');
    save('distance4.mat','d4');
    close all;
end
```

Appendix B

Verilog codes for the hardware implementation of five similarity measures.

Dtw

```
module dtw(in1,in2);

                    //output reg out;
                    input[23:0]in1;
                    input[31:0]in2;
                    reg[3:0]array1[0:5];
                    reg[3:0]array2[0:7];
                    parameter STRINGS =10;
                    integer j,k;   //No. of elements in a string
                    integer i;  //No. of strings
                    //reg[1:0]state;
                    reg[5:0]test[47:0];
                    reg[5:0]temp[47:0];

                    reg[5:0]o;

always@(in1 or in2)begin
                    {array1[5],array1[4],array1[3],array1[2],array1[1],array1[0]}=in1;
{array2[7],array2[6],array2[5],array2[4],array2[3],array2[2],array2[1],array2[0]}=in2;


for (j=0;j<8;j=j+1)begin

                    for(i=0;i<6;i=i+1)      begin
                    k=(8*i)+j;
temp[k]=(array1[i]>array2[j])?((array1[i]-array2[j])*(array1[i]-array2[j])):((array2[j]-array1[i])*(array2[j]-
array1[i]));


    end
 end
                                   for(i=0;i<48;i=i+1)begin
                                   if(i==0)begin
                                   test[i]=temp[i]  ;
                                   end
                                          else if((i>0) && (i<8))begin
                                                 test[i]=temp[i]+test[i-1];
                                          end
                    else if((i>0) && (i%8==0)) begin
                    test[i]=temp[i]+test[i-8];
                    end
                                               else if(i>0) begin
 o=(test[i-1]<test[i-8])?((test[i-1]<test[i-9])?test[i-1]:test[i-9]):((test[i-8]<test[i-9])?test[i-8]:test[i-9]);

                                          test[i]=temp[i]+o;
                                          end
                                   end
                                          out=test[47];
 end
                    endmodule
```

LCSS

```
module lcss(in1, in2);
                //input clk;
                input[17:0]in1;
                input[23:0]in2;
                reg[2:0]array1[0:5];
                reg[2:0]array2[0:7];
                //output reg out;
                //parameter N =4;
                integer j,k;    //No. of elements in a string
                integer i;  //No. of strings
                                reg[2:0]test[47:0];
                //reg[5:0]temp[41:0];
                                //reg[2:0]o;

always@(in1 or in2)begin
                {array1[5],array1[4],array1[3],array1[2],array1[1],array1[0]}=in1;
{array2[7],array2[6],array2[5],array2[4],array2[3],array2[2],array2[1],array2[0]}=in2;



for (j=0;j<8;j=j+1)begin

                for(i=0;i<6;i=i+1)      begin

                    //if (array1[i] == array2[j])
                    //begin
                    if((i==0) || (j==0))begin
                                                if (array1[i] == array2[j])
                    begin
                        test[6*j+i]=3'b001;
                                                    end
                                                    else begin
                                                    test[6*j+i]=3'b000;
                                                    end
                                                    end
                        else begin
                        if(array1[i] == array2[j])begin
                        test[6*j+i]= 3'b001+test[6*(j-1)+(i-1)];
                                end

                                else begin
                                if(i>0 && j>0)begin
                   test[6*j+i]=(test[6*(j-1)+i]>test[6*j+(i-1)]? test[6*(j-1)+i]:test[6*j+(i-1)]) ;
                                                    end

                   else if (i>0 && j==0)

                        begin

                            test[6*j+i]=test[6*j+(i-1)];

                        end

                                            else if  (i==0 && j>0)begin

                                            test[6*j+i]=test[6*(j-1)+i];
                                            end
end
```

```
end
                    end

                                        end

                    //out=test[47];
                                            end
endmodule
```

EDR

```
module EDR(in1, in2);
input[23:0]in1;
input[23:0]in2;
reg[2:0]array1[0:7];
reg[2:0]array2[0:7];
 // output reg out;
  //parameter N =4;
                    integer j,k;    //No. of elements in a string
                    integer i;  //No. of strings
                                    reg[3:0]test[63:0];
                    //reg[5:0]temp[41:0];
                                    //reg[2:0]o;

always@(in1 or in2)begin
{array1[7],array1[6],array1[5],array1[4],array1[3],array1[2],array1[1],array1[0]}=in1;
{array2[7],array2[6],array2[5],array2[4],array2[3],array2[2],array2[1],array2[0]}=in2;


                    for (j=0;j<8;j=j+1)begin

                    for(i=0;i<8;i=i+1)        begin

                         //if (array1[i] == array2[j])
                         //begin
                         if((i==0) && (j==0))begin
                         if (array1[i] == array2[j]) begin
                                test[(8)*j+i]=3'b000;
                                                            end
                                                            else begin
                                                            test[(8)*j+i]=3'b001;
                                                            end
                                                            end

                                            else begin
                                            if (i>0 && j>0)begin

                                            if(array1[i] == array2[j])begin
                                            test[(8)*j+i]=(test[(8)*(j-1)+i]<test[8*j+(i-1)])?
(((test[8*(j-1)+i]+3'b001)<test[8*(j-1)+(i-1)])?(test[8*(j-1)+i]+3'b001):test[8*(j-1)+(i-1)]):(((test[8*j+(i-
1)]+3'b001)<test[8*(j-1)+(i-1)])?(test[8*j+(i-1)]+3'b001):test[8*(j-1)+(i-1)]) ;
                                                              end
                                                                  else begin
```

```
test[(8)*j+i]=(test[(8)*(j-1)+i]<test[(8)*j+(i-1)])?            ((test[(8)*(j-1)+i]<test[(8)*(j-1)+(i-1)])?test[(8)*(j-1)+i]+3'b001:test[(8)*(j-1)+(i-1)]+3'b001):((test[8*j+(i-1)]<test[(8)*(j-1)+(i-1)])?test[8*j+(i-1)]+3'b001:test[(8)*(j-1)+(i-1)]+3'b001) ;
                                                                    end
                                                                    end

                                else if (i>0 && j==0)

                                     begin

                                          test[(8)*j+i]=3'b001+test[(8)*j+(i-1)];

                                     end


                                                    else if (i==0 && j>0) begin

                                                    test[(8)*j+i]=3'b001+test[(8)*(j-1)+i];


                                end
                             end
                            end
                                end


//out=test[47];
                                end


endmodule
```

ERP

```
module ERP(in1,in2);

input[17:0]in1;
input[17:0]in2;
reg[2:0]array1[0:5];
reg[2:0]array2[0:5];
  //output reg out;
  parameter N =6;
                            integer j,k;    //No. of elements in a string
                            integer i;  //No. of strings
                                          reg[3:0]test[(N*N)-1:0];
                            //reg[5:0]temp[41:0];
                                          reg[2:0]o;
always@(in1 or in2)begin
                            {array1[5],array1[4],array1[3],array1[2],array1[1],array1[0]}=in1;
                            {array2[5],array2[4],array2[3],array2[2],array2[1],array2[0]}=in2;
```

```
for (j=0;j<N;j=j+1)begin

                                 for(i=0;i<N;i=i+1)       begin

                                      //if (array1[i] == array2[j])
                                      //begin
                                      if((i==0) && (j==0))begin
                                      if (array1[i] == array2[j])  begin
                                                 test[(N)*j+i]=3'b000;
                                                                                         end
                                                                                         else begin

                                 test[(N)*j+i]=((array1[i]>array2[j])?(array1[i]-array2[j]):(array2[j]-array1[i]));
                                                                                         end
                                                                                         end

                                                                         else begin
                                                                         if (i>0 && j>0)begin

                                                                         if(array1[i] == array2[j])begin

  test[(N)*j+i]=(test[N*(j-1)+i]+array2[j]<test[N*j+(i-1)]+array1[i])?((test[N*(j-1)+i]+array2[j]<test[N*(j-1)+(i-1)])?test[N*(j-1)+i]+array2[j]:test[N*(j-1)+(i-1)]):((test[N*j+(i-1)]+array1[i]<test[N*(j-1)+(i-1)])?test[N*j+(i-1)]+array1[i]:test[N*(j-1)+(i-1)]);

                                                                                                      end

                                                                                                      else begin

o=((array1[i]>array2[j])?(array1[i]-array2[j]):(array2[j]-array1[i]));


test[(N)*j+i]=(test[N*(j-1)+i]+array2[j]<test[N*j+(i-1)]+array1[i])?((test[N*(j-1)+i]+array2[j]<test[N*(j-1)+(i-1)]+0)?test[N*(j-1)+i]+array2[j]:test[N*(j-1)+(i-1)]+o):((test[N*j+(i-1)]+array1[i]<test[N*(j-1)+(i-1)])?test[N*j+(i-1)]+array1[i]:test[N*(j-1)+(i-1)]+o);

                                                                                                       end
                                                                                                       end


                                     else if (i>0 && j==0)begin

                                                       test[(N)*j+i]=array1[i]+test[(N)*j+(i-1)];

                                              end

                                              else if (i==0 && j>0) begin

                                                                test[(N)*j+i]=array2[j]+test[(N)*(j-1)+i];

                                                       end

                                                                 end
end
                                                                  end
                                     //out=test[N*N-1];
end
 endmodule
```

MSM

```
module MSMS(in1,in2);
input[14:0]in1;
input[14:0]in2;
reg[2:0]array1[0:4];
reg[2:0]array2[0:4];
reg[2:0]k,k1;
reg[2:0]l,l1;
  //output reg out;
  parameter N =5;
                              integer j;   //No. of elements in a string
                              integer i;  //No. of strings
                                                   reg[4:0]test[N*N-1:0];
                              //reg[5:0]temp[41:0];
                                                   reg[2:0]o1,o2,o3,o4,o5;

always@(in1 or in2)begin
                              {array1[4],array1[3],array1[2],array1[1],array1[0]}=in1;
 {array2[4],array2[3],array2[2],array2[1],array2[0]}=in2;
                              test[0]=array1[0]>array2[0]?array1[0]-array2[0]:array2[0]-array1[0];
                              for(i=1;i<N;i=i+1) begin
                              k=(array1[i]>array1[i-1])?array1[i]-array1[i-1]:array1[i-1]-array1[i];
l=(array1[i]>array2[0])?array1[i]-array2[0]:array2[0]-array1[i];
if((array1[i-1]<=array1[i] && array1[i]<=array2[0])||(array1[i-1]>=array1[i] && array1[i]>=array2[0]))begin
o1=3'b010;
end
else begin
o1=3'b010 + (k>l?l:k);
end


                              test[i]=test[i-1]+o1 ;
                              end
                              for(j=1;j<N;j=j+1)begin
                              k=(array2[j]>array1[0])?array2[j]-array1[0]:array1[0]-array2[j];
l=(array2[j]>array2[j-1])?array2[j]-array2[j-1]:array2[j-1]-array2[j];
if((array1[0]<=array2[j] && array2[j]<=array2[j-1])||(array1[0]>=array2[j] && array2[j]>=array2[j-1]))begin
o2=3'b010;
end
else begin
o2=3'b010 + (k>l?l:k);
end

                              test[N*j]=test[N*(j-1)]+o2;
                              end
                              for(i=1;i<N;i=i+1)       begin
                              for(j=1;j<N;j=j+1) begin
                              o3=array1[i]>array2[j]?array1[i]-array2[j]:array2[j]-array1[i];
                              k=(array1[i]>array1[i-1])?array1[i]-array1[i-1]:array1[i-1]-array1[i];
l=(array1[i]>array2[j])?array1[i]-array2[j]:array2[j]-array1[i];
if((array1[i-1]<=array1[i] && array1[i]<=array2[j])||(array1[i-1]>=array1[i] && array1[i]>=array2[j]))begin
o4=3'b010;
end
else begin
o4=3'b010 + (k>l?l:k);
end
```

```
k1=(array2[j]>array1[i])?array2[j]-array1[i]:array1[i]-array2[j];
l1=(array2[j]>array2[j-1])?array2[j]-array2[j-1]:array2[j-1]-array2[j];
if((array1[i]<=array2[j] && array2[j]<=array2[j-1])||(array1[i]>=array2[j] && array2[j]>=array2[j-1]))begin
o5=3'b010;
end
else begin
o5=3'b010 + (k1>l1?l1:k1);
end


test[N*i+j]=(test[N*(i-1)+(j-1)]+o3<test[N*(i-1)+j]+o4)?(test[N*(i-1)+(j-1)]+o3<test[N*i+(j-1)]+o5?test[N*(i-1)+(j-1)]+o3:test[N*i+(j-1)]+o5):(test[N*(i-1)+j]+o4<test[N*i+(j-1)]+o5?test[N*(i-1)+j]+o4:test[N*i+(j-1)]+o5);

                    end
                    end
                    //out=test[N*N-1];
                    end
endmodule
```

Appendix C

Dataset of pair of songs for confirming the efficacy of similarity measures in plagiarism detection.

Source: http://www.quora.com/What-are-the-worst-cases-of-plagiarism-in-music

Index of song	Song	You tube link
11	Jay Z & Beyoncé "Drunk in Love"	https://youtu.be/p1JPKLa-Ofc
12	Mitsou "Bajba, Bajba Pelem	https://youtu.be/A59LegFfYRw
21	Eminem "Rap God"	https://youtu.be/XbGs_qK2PQA
22	Hot Stylz "Lookin' Boy"	https://youtu.be/Mu4iYTrJbwk
31	Jay Z & Kanye West, ft. Frank Ocean "Made in America"	https://youtu.be/zSDzPByjEuM
32	Joel McDonald "Made in America"	https://youtu.be/LiUHYrN4NDI
41	The Beach Boys "Surfin' U.S.A."	https://youtu.be/sNypbmPPDco
42	Chuck Berry "Sweet Little Sixteen"	https://youtu.be/ZLV4NGpoy_E
51	Skillet "Monster"	https://youtu.be/1mjlM_RnsVE
52	Three Days Grace "Animal I Have Become"	https://youtu.be/xqds0B_meys
61	Nirvana "Come As You Are"	https://youtu.be/vabnZ9-ex7o
62	Killing Joke "Eighties"	https://youtu.be/x1U1Ue_5kq8
71	Katy Perry "Dark Horse"	https://youtu.be/0KSOMA3QBU0
72	Flame ft. Lecrae "Joyful Noise"	https://youtu.be/QCcW-guAs_s
81	Will.i.am "Let's Go"	https://youtu.be/pMpFN4k5piU

82	Arty & Mat Zo "Rebound"	https://youtu.be/Kmp3MguaTSA
91	Led Zeppelin "Whole Lotta Love"	https://youtu.be/Q0utAHY3xo4
92	Muddy Waters "You Need Love"	https://youtu.be/pM8_HuQ0b34
101	Coldplay "Viva La Vida"	https://youtu.be/dvgZkm1xWPE
102	Joe Satriani's "If I Could Fly"	https://youtu.be/nrEXnizgt9c
111	Avril Lavigne "Girlfriend"	https://youtu.be/Bg59q4puhmg
112	The Rubinoos "I Wanna Be Your Boyfriend"	https://youtu.be/j3t66Nrqteo
121	One Direction "Live While We're Young"	https://youtu.be/AbPED9bisSc
122	The Clash "Should I Stay or Should I Go"	https://youtu.be/xMaE6toi4mk
131	Justin Bieber "Baby"	https://youtu.be/kffacxfA7G4
132	Perla "Tremendo Vacilão"	https://youtu.be/2IMi3GwD9LI
141	The Doors "Hello, I Love You"	https://youtu.be/hzM71scYw0M
142	The Kinks "All Day and All of the Night"	https://youtu.be/mMWNwHof0kc
151	James Blunt "Heart to Heart"	https://youtu.be/CsFb661EXsI
152	Five for Fighting "100 Years"	https://youtu.be/tR-qQcNT_fY
161	Bruno Mars "Treasure"	https://youtu.be/nPvuNsRccVw
162	Breakbot "Baby I'm Yours"	https://youtu.be/6okxuiiHx2w
171	Oasis "Whatever"	https://youtu.be/EHfx9LXzxpw
172	Neil Innes "How Sweet to Be an Idiot"	https://youtu.be/nZ9EWcaS7II

181	Simple Plan "Your Love is a Lie"	https://youtu.be/XAbcgmwq3EU
182	Green Day "Boulevard of Broken Dreams"	https://youtu.be/Soa3gO7tL-c
191	Katy Perry "Roar"	https://youtu.be/CevxZvSJLk8
192	Sara Bareilles "Brave"	https://youtu.be/QUQsqBqxoR4
201	Led Zeppelin "Stairway to Heaven"	https://youtu.be/w9TGj2jrJk8
202	Taurus "Spirit"	https://youtu.be/xd8AVbwB_6E
211	Meghan Trainor "All About That Bass"	https://youtu.be/7PCkvCPvDXk
212	Koyote's "Happy Mode"	https://youtu.be/Bg8dlyO7T3Y
221	The Beatles "Come Together"	https://youtu.be/axb2sHpGwHQ
222	Chuck Berry "You Can't Catch Me"	https://youtu.be/wfD4Eo7cA0Y
231	Jennifer Lopez "On the Floor"	https://youtu.be/t4H_Zoh7G5A
232	Kaoma "Lambada"	https://youtu.be/WJTwgMTI704
241	Rod Stewart "Do Ya Think I'm Sexy?"	https://youtu.be/Hphwfq1wLJs
242	Jorge Ben "Taj Mahal"	https://youtu.be/ILZjZ85mASk
251	Radiohead "Creep"	https://youtu.be/XFkzRNyygfk
252	Albert Hammond "The Air That I Breathe"	https://youtu.be/9HglphdXqMg
261	Robin Thicke "Blurred Lines"	https://youtu.be/yyDUC1LUXSU
262	Marvin Gaye "Got to Give It Up"	https://youtu.be/fp7Q1OAzITM
271	Ray Parker Jr. "Ghostbusters Theme"	https://youtu.be/Fe93CLbHjxQ
272	Huey Lewis "I Want a New Drug"	https://youtu.be/N6uEMOeDZsA

281	Jet "Are You Gonna Be My Girl"	https://youtu.be/tuK6n2Lkza0
282	Iggy Pop "Lust for Life"	https://youtu.be/jQvUBf5l7Vw
291	David Guetta "Play Hard"	https://youtu.be/5dbEhBKGOtY
292	Alice Deejay "Better Off Alone"	https://youtu.be/hneLe48CpEs
301	Jay Z 'Big Pimpin'"	https://youtu.be/2ceEnFpU2m4
302	Baligh Hamdi "Khosara Khosara"	https://youtu.be/bKcVDJGOvNU
311	“Kaho Na Kaho" – Murder	https://youtu.be/5G4bqxClDqk
312	"Tamaly Maak" - Amr Diab	https://youtu.be/EgmXTmj62ic
321	"Neend Churayee Meri" - Ishq	https://youtu.be/Y2Ne_C6dfOg
322	"Sending All My Love" - Linear	https://youtu.be/WE-5KBuDG4k
331	"Raja Ko Rani Se" - Akele Hum Akele Tum	https://youtu.be/t5JSLRB9ZAU
332	"The Love Theme" - Godfather - Nino Rota	https://youtu.be/zMGE8pks9UE
341	"Chakle Chakle" - Deewane Huye Paagal	https://youtu.be/r9KrHvr-SrE
342	"Turn Me On" - Kevin Lyttle	https://youtu.be/Vgy8vOzl-po
351	"Dil Mera Churaya Kyon" - Akele Hum Akele Tum	https://youtu.be/P3n6I91yhcs
352	"Last Christmas" - Wham!	https://youtu.be/E8gmARGvPlI
361	"Aisa Milan Kal Ho Na Ho" - Hameshaa	https://youtu.be/xIfYQaYsj6Y
362	"The Phantom of the Opera" - Andrew Lloyd Webber	https://youtu.be/Ej1zMxbhOO0
371	"Dil Le Le Lena" - Auzaar	https://youtu.be/0cMfviJD24I
372	"Macarena" - Los Del Rio	https://youtu.be/XiBYM6g8Tck

381	"Aisa Zakhm Diya Hai" - Akele Hum Akele Tum	https://youtu.be/P2bhDV8ZIpI
382	"Child In Time" - Deep Purple	https://youtu.be/PfAWReBmxEs
391	"Jaane Jaana" - Murder	https://youtu.be/EfBUj1JMmzI
392	"Firiye Dao" - Miles	https://youtu.be/79mm04ArWU0
401	"Tu Waaqif Nahin" - Khiladiyon Ka Khiladi	https://youtu.be/5KpH2Dck6CE
402	"Fernando" - ABBA	https://youtu.be/dQsjAbZDx-4
411	"Yeh Kaali Kaali Ankhen" - Baazigar	https://youtu.be/jyeR5tjZfiw
412	"The Man who plays the Mandolino" - Dean Martin	https://youtu.be/fSQRTo82--8
421	"Is Tarah Aashiqui Ka" - Imtihaan	https://youtu.be/XTTi3ivv8wg
422	"Autumn Leaves" - Nat King Cole	https://youtu.be/684eg6S8dCw
431	"Tera Gussa" - Kareeb	https://youtu.be/mXs32S_DREc
432	"The Happy Birthday Song"	https://youtu.be/qCJSNMqub8g
441	"Jaane Kaise" - Raqeeb	https://youtu.be/M5VgVam1GFk
442	"Allem Alby" - Amr Diab	https://youtu.be/u4bCK-M5TL0
451	"Kya Mujhe Pyaar Hai" - Woh Lamhe	https://youtu.be/nbkbk32UEh4
452	"Tak Bisakah" - Peterpan	https://youtu.be/n_rwZ3ETimI
461	"Zara Zara Touch Me" - Race	https://youtu.be/Sv_kEdNwYtQ
462	"Zhu Lin Shen Chu" - Leehom Wang	https://youtu.be/8vzTAOttZec
471	"Pehli Nazar Mein" - Race	https://youtu.be/BadBAMnPX0I

472	"Sarang Hae Yo" - Kim Hyung Sup	https://youtu.be/4JaxfCUxofY
481	"Badtameez Dil" - Yeh Jawaani Hai Deewani	https://youtu.be/F7k_U1ZXybo
482	"Ranjana Ami Aar Asbo Na" - Anjan Dutt	https://youtu.be/R3nyGm397k8
491	"Hare Krishna Hare Ram" - Bhool Bhulaiya	https://youtu.be/2Tb6Q1PH-_0
492	"My Lecon" - JTL	https://youtu.be/Mecp01YMrA
501	"Yeh Ishq Hai" - Jab We Met	https://youtu.be/p9dsPjx17Yw
502	"Etra Una Femme" - Anggun	https://youtu.be/T4poevqspsI

Appendix D

Numerical values for the hardware utilization and delay obtained after synthesis.

I Slice utilization versus number of elements in sequence

N	DTW	LCSS	EDR	ERP	MSM
4	360	35	49	368	579
5	606	71	121	660	977
6	915	117	220	1033	1463
7	1285	174	354	1479	2093
8	1722	289	502	2009	2802
9	2122	356	656	2612	3666
10	2660	447	864	3305	NA

II LUTs utilization versus number of elements in sequence

N	DTW	LCSS	EDR	ERP	MSM
4	699	60	88	621	963
5	1177	124	218	1116	1633
6	1778	204	398	1752	2471
7	2501	304	641	2513	3490
8	3353	507	906	3418	4670
9	4162	623	1185	4446	6122
10	5219	780	1560	5672	NA

III IOBs utilization versus number of elements in sequence

N	DTW	LCSS	EDR	ERP	MSM
4	38	25	37	25	25
5	46	31	43	31	31
6	54	37	49	37	37
7	62	43	49	43	43
8	70	49	49	49	49
9	78	55	61	55	55
10	86	61	67	61	NA

IV Delay (in nsec) variation with no. of elements in sequence

N	DTW	LCSS	EDR	ERP	MSM
4	52.8	17.7	26.6	70.6	73.227
5	67	23.5	38.8	94.9	94.82
6	81.7	27.9	48	120.5	117.286
7	96.4	32.6	64.8	145	138.42
8	110.6	46.4	76.6	171.4	160.785
9	124.4	46.8	89	195.1	182.798
10	139.5	51.2	103.7	221	212.345

V MATLAB execution time (in μ sec) variation with no. of elements in sequence

N	DTW	LCSS	EDR	ERP	MSM
4	104	187	182	230	231
5	139	195	201	249	390
6	182	212	292	272	404
7	232	222	301	356	449
8	279	254	342	380	509
9	342	314	391	399	555
10	412	638	510	492	680