Reconfigurable Design For Pattern Recognition Using Field Programmable Gate Arrays

A thesis presented to

the faculty of

Fritz J. and Dolores H. Russ
College of Engineering and Technology

Ohio University

In Partial Fulfillment

of the Requirement for the Degree

Master of Science

By

Aman Sareen

August, 1999

This Thesis Entitled

Reconfigurable Design For Pattern Recognition Using Field Programmable Gate Arrays

by Aman Sareen

has been approved

for the school of Electrical Engineering and Computer Science and the Russ College of Engineering and Technology

---

Janusz Starzyk, Professor
School of Electrical Engineering and Computer Science

Warren K. Wray, Dean
Fritz J. and Dolores H. Russ
College of Engineering and Technology

Acknowledgment

I am highly thankful to my thesis advisor, Dr. Janusz Starzyk, for his sincere advice and persistent support. Dr. Starzyk recommended many useful references and attributed his insights in many theoretical issues. He changed my life as a normal student into one as a hard worker, researcher and knowledge-seeker. I am especially grateful for the time he dedicated by driving from Dayton to Athens each week to help me.

I thank also Dr. Curtis and Dr. Giesey for their precious time, valuable help and support with their guidance and ideas.

Special thanks go to my parents, and my brother, for their support and encouragement that gave me spiritual strength and endurance in pursuing my ambition.

I am also very thankful to Abdulqadir Al-aqeeli with whom I had many brainstorming sessions that helped to decide the architecture on the FPGA. Finally, thanks to all my friends who assisted me upon request and were very helpful in their suggestions.

Table of Contents

Page

List of Tables

List of Figures

Chapter 1

Introduction

Pattern recognition techniques are often an important component of intelligent systems and are used for both data pre-processing and decision making. Broadly speaking, pattern recognition is the science that concerns the description or classification (recognition) of measurements. The following enumerates a few of the areas where pattern recognition finds its application [2]:

- Image preprocessing, segmentation, and analysis
- Computer vision
- Artificial intelligence
- Seismic analysis
- Radar signal classification/analysis
- Speech recognition/understanding
- Fingerprint identification
- Character (letter or number) recognition
- Handwriting analysis
- Electro-cardiographic signal analysis/understanding
- Medical diagnosis
- Socioeconomic
- Archaeology

- Data mining/reduction

There are three inter-related pattern recognition approaches:

- Statistical Pattern Recognition
- Syntactic Pattern Recognition
- Neural Pattern Recognition

1.1 Statistical pattern recognition approach

Statistical pattern recognition attempts to classify patterns based on a set of extracted features and an underling statistical model for the generation of these patterns. It assumes a statistical basis for classification of algorithms. A set of features is extracted from the input data and is used to assign each feature vector to one of the classes. We concentrate on developing decision or classification strategies, which form classifiers. The classifier design attempts to integrate all available information and decision rules are formulated. In the use of statistical pattern recognition approach, the structure of the pattern is insignificant, so if the structural information is unavailable or irrelevant, it makes sense to use statistical pattern recognition approach. Some existing statistical approaches are discussed in Chapter 2 of this thesis [3].

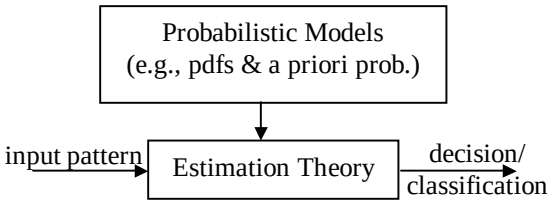


Fig. 1.1: Statistical PR approach.

1.2 Syntactic pattern recognition approach

The principle behind this approach is that many times the interrelationships or interconnections of features yield important structural information, which facilitates structural description or classification. Therefore, in these approaches we must be able to quantify and extract structural information and to assess structural similarity of patterns. In general, these approaches formulate hierarchical description of complex patterns built up from simpler sub-patterns. For e.g. musical patterns. There are 6 octaves and each octave is subdivided into distinct tones, giving a total of about 72 distinct tones, which are common to all musical classes. The premise of syntactic pattern recognition is that the structure of an entity is paramount, and that it may be used for classification and description. It is used for both classification and description. Classification may be based on measures of pattern structural similarity. When explicit structural information about the patterns is available, it makes sense to use syntactical pattern recognition approach [3].

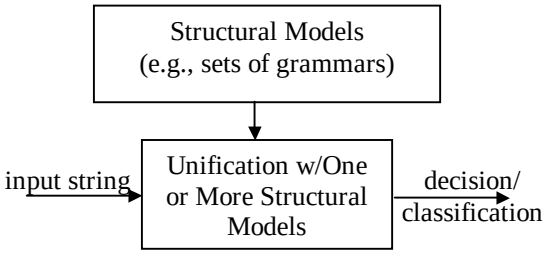


Fig. 1.2: Syntactic PR approach.

1.3 Neural pattern recognition approach

This approach makes use of the knowledge of how biological neural systems store and manipulate information. They are particularly well suited for pattern association applications. Artificial neural networks (ANNs) provide an emerging paradigm for pattern recognition implementation that involves large interconnected networks of relatively simple and typically nonlinear units so called neural-nets. Basically, three entities characterize an ANN [3]:

1. The network topology, or interconnection of neural units,
2. The characteristics of individual units or artificial neurons, and
3. The strategy for pattern learning or training.

The myriad of potential neural network applications for pattern recognition includes:

- Feature extraction from complex data sets (e.g., images and speech);
- Character recognition and image processing applications, and
- Direct and parallel implementation of matching and search algorithms.

The problems are characterized by following qualities

- A high-dimensional problem space,
- Complex interactions between problem variables, and
- A solution space that may be empty, contain a unique solution, or contain a number of almost equally useful solutions.

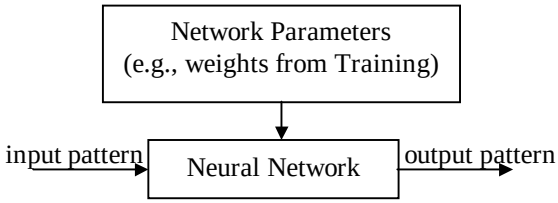


Fig. 1.3: Neural network approach

The objective of the thesis is to present the approach used for feature extraction of the HRR signals. It was demonstrated in this work that by selecting features based on evolutionary use of Haar wavelets, new features with better classification properties could be obtained. The convergence is very fast and produces results close to the Bayesian classification limits. The quality of the obtained transformation is measured using the entropy based information index. This index was developed for data sets with estimated probability density functions of different classes. Any mutual dependence of the extracted features is automatically accounted for by performing a Monte Carlo integration in multi-dimensional space. The confidence interval of the predicted performance is related to standard deviation of the information index and depends on the information level and the number of training points used. Chapter 3 discusses the algorithms used and the simulation results and graphs for some sample signals are shown in Chapter 4. Chapter 5 shows an architecture in which the discussed algorithm could be implemented on VLSI chips. The detailed implementation techniques and methods are discussed in the thesis, “Reconfigurable Wavelet-Based Architecture for Pattern

Recognition Applications Using a Field Programmable Gate Array" by Mr. Abdulqadir Alaqeeli. Chapter 6 lists the conclusions based on the work done and future work.

1.1 Tools used for the work:

Matlab was used to develop the algorithms and produce various results and graphs. The VHDL code was targeted to the Xilinx 4000 series FPGA, and the Xilinx Foundation tools were used for that.

Chapter 2

Statistical approaches to pattern recognition

There are two aspects to pattern recognition - developing a decision rule and using it. The actual recognition occurs in the use of the rule, the pattern is defined in the learning process by the labeled samples. Since ‘O’ could be seen as a ‘circle’ or as a character ‘O’, the pattern recognition problem thus begins with class definition and labeled samples of those classes in some workable representation. The problem is solved when a decision rule is derived which assigns a unique label to new patterns. Pattern recognition is concerned primarily with the description and analysis of measurements taken from processes [1]. For this preprocessing is often required to remove noise and measurement redundancy. An important step for any classification algorithm is feature selection and extraction from the training data.

2.1 Feature selection (or preprocessing):

Feature selection is the process by which a sample in the measurement space is described by a finite and usually smaller set of numbers called frames, which become components of the pattern space. Its based on the hypothesis that the training data follow some natural rules, which makes presence or absence of data from a given class in a specific location of the input space predictable. The role of feature extraction is to transform the input space in possibly a nonlinear fashion establishing separation boundaries between classes.

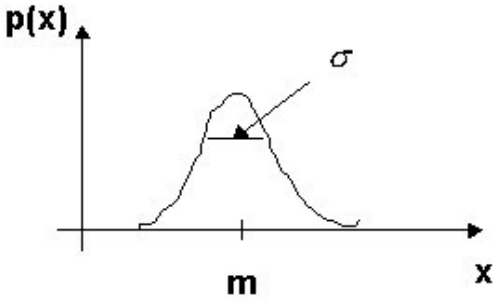
Extraction of features depends on selection of a transformation operator which can be linear (e.g. projection, wavelets, Fourier transform) or nonlinear [4].

2.2 Histograms:

This approach is one of the oldest to estimate probability density functions. The line in which the samples occur is divided into several intervals. The probability of a sample occurring in each interval is defined by the number of sample points in that interval divided by the total number of sample points. The probability density is the probability divided by the length of the interval. As the dimensionality increases the number of sample points needed to estimate the density functions increases enormously and so the histogram approach is seldom used for estimation of probability density functions.

2.3 Parametric Methods:

This approach defines the descriminant function by a class of probability densities defined by a relatively small number of parameters. Its generally assumed that each pattern class arises from a multivariate Gaussian (normal) distribution, where parameters are the mean and covariance.



$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-m)^2}{2\sigma^2}}$$

$$p(x_1, \ldots, x_n) = \frac{1}{\sqrt{2\pi}^n \sqrt{|\Sigma|}} e^{-\frac{1}{2}(\mathbf{X}-\mathbf{M})^T \Sigma^{-1} (\mathbf{X}-\mathbf{M})}$$

Fig 2.1: Determination of parameters for Gaussian distribution.

Two approaches are used for parametric estimation. They are:

- **Maximum Likelihood Estimation:** parameters are fixed but unknown i.e. it seeks for parameters which maximizes the probability of obtaining the given training set.

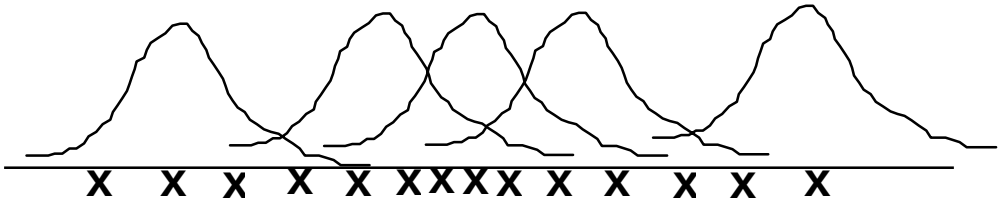


Fig. 2.2: Illustration of the maximum likelihood approach.

- **Bayesian approach:** models the to-be-estimated parameters as random variables with some assumed known distribution. It uses training set to update density function of known parameters.

Calculate $d_1^2(X) = (X - M_1)^T \Sigma_1^{-1}(X - M_1)$ and $d_2^2(X) = (X - M_2)^T \Sigma_2^{-1}(X - M_2)$

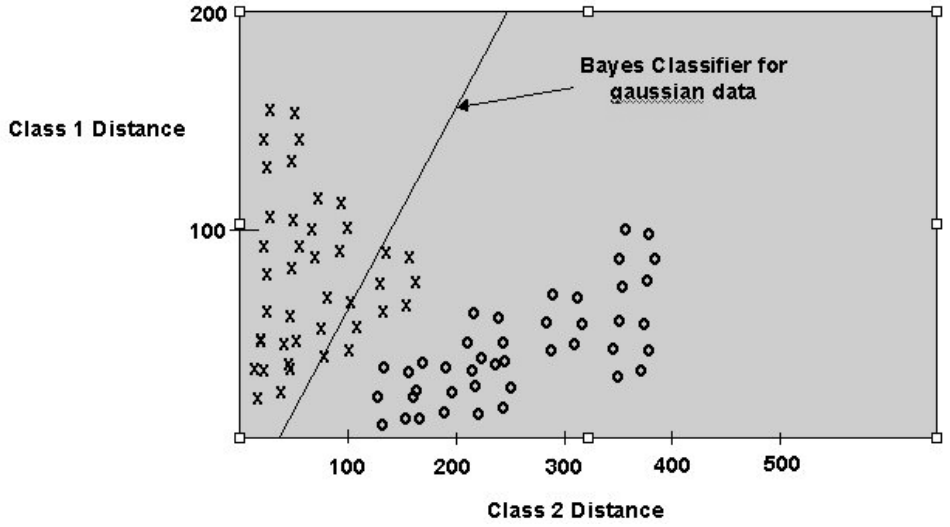


Fig. 2.3: Classification of Gaussian data sets using Bayes classifier

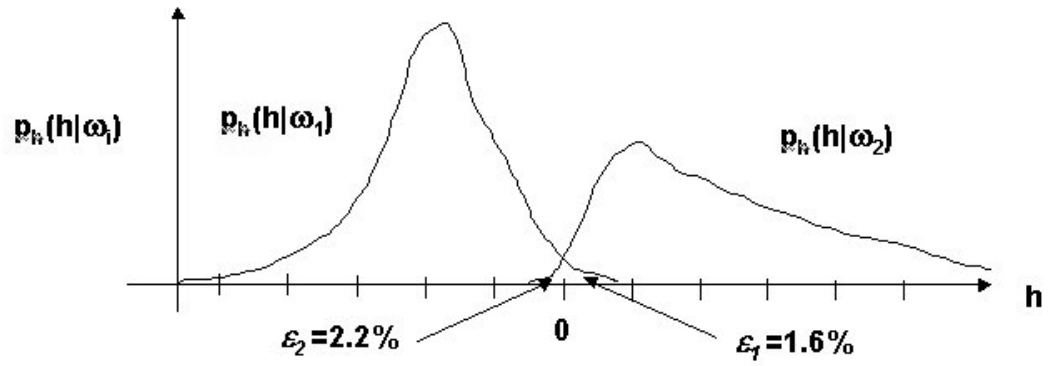


Fig. 2.4: Illustration of Bayes classification error.

Problem with parametric learning is

- It might be difficult to determine a specific form (e.g. Gaussian, Uniform) for the distribution.

- The chosen form does not fit one of the estimable formulations.

2.4 Non-parametric methods:

They use parameterized discriminant functions, like the coefficients of a multivariate polynomial of some degree. No conventional form of probability distribution is assumed. The following section describes in detail one of the non-parametric approach.

2.4.1 k-NN approach:

The rule is – classify a point as a member of the class to which majority of its k-nearest neighbors belong [2]. It assumes that distance between points is a legitimate measure of the similarity of the patterns they represent. The algorithm weights equally all k-nearest neighbors, no matter what their relative distance from the point in question. In implementation, nearest neighbor pattern classification requires storage of the entire sample points and computation of the distance from all to a point in question. Use of this method requires careful normalization of pattern space and choice of distance and is best for small number of samples of reasonably low dimension. Figure 2.5 illustrates the kNN approach.

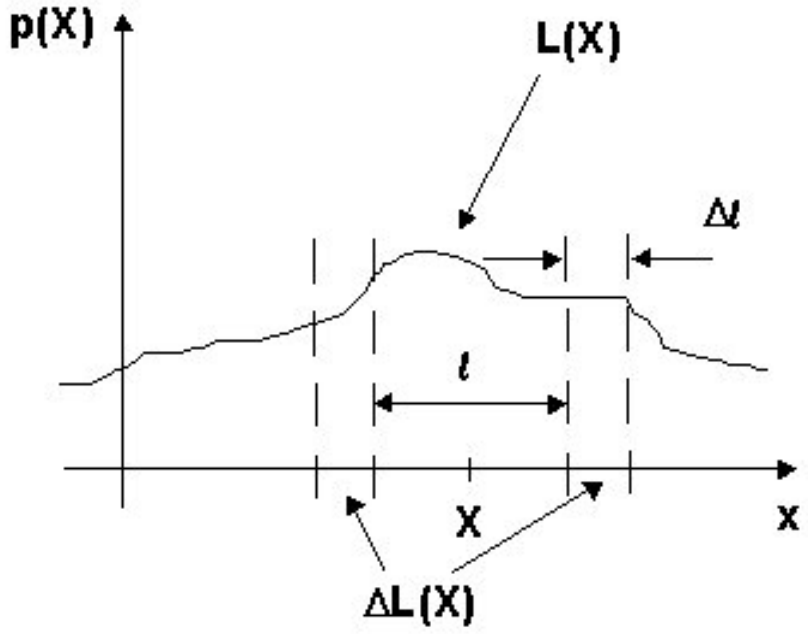


Fig. 2.5: Illustration of the kNN approach.

2.4.2 Parzen window approach:

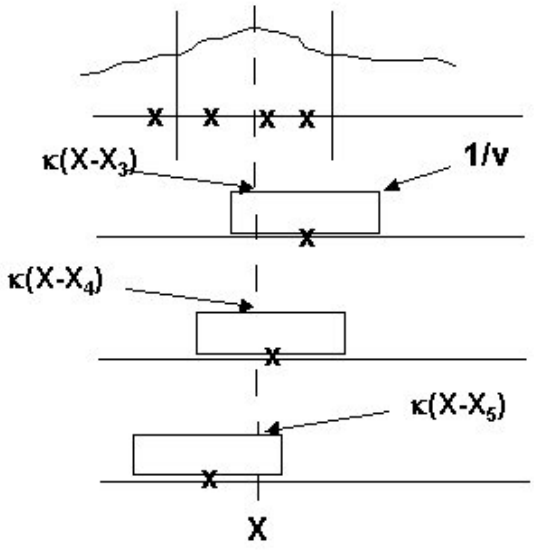
The Parzen-window approach is a well-know technology for estimating probability density functions. A density function (the kernel) $\Phi(x)$ is used for which: $\Phi(x) > 0$, $\int \Phi(x)dx = V$. Having samples x_i, then the estimated pdf $\hat{p}(x)$ is obtained from [6]:

$$\hat{p}(x) = \frac{1}{V_n n} \sum_{i=1}^{n} \Phi_n(x - x_i)$$

where n is the total number of data and V_n is the kernel volume:

$$V_n = \int \Phi_n(x)dx$$

The kernel is usually a function of $\frac{1}{\sqrt{(n)}}$ i.e. the larger the number of data the smaller the kernel. In practice, the number of data is limited and therefore the kernel can not be taken too small (otherwise, holes and spikes appear in the estimated pdf. In practice Gaussian kernel is often used and a width is chosen which optimizes the classification performance. Another practical problem is the computational time and storage for estimating a pdf value.



Original estimate

$$\hat{p}(X) = \frac{k}{Nv} = \frac{3}{Nv}$$

Parzen estimate

$$\hat{p}(X) = \frac{1}{N}\sum_{i=1}^{N}\kappa\left(X - X_i\right)$$

$$= \frac{1}{N}\left(\tfrac{1}{v} + \tfrac{1}{v} + \tfrac{1}{v}\right) = \frac{3}{Nv}$$

Fig. 2.6 Illustration of the Parzen window approach.

Chapter 3

Entropy based selection of optimum transformation of input data

In the classification task, a favorable occurrence is when an object (or signal) is correctly classified. If classification is based on given probability distribution functions, one possible approach to classify an observed signal is to evaluate pdf functions for different classes of this observed signal. The signal is classified as an object of the class with the maximum pdf value. Let us consider a two-class classification problem with pdf functions as shown in Figure 3.1.

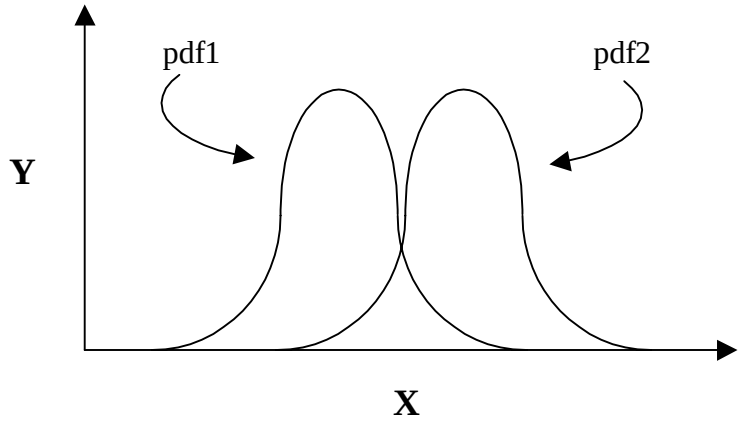


Fig. 3.1: Probability distribution function of two classes.

Probability that a signal from class 1 will be correctly classified under this scenario equals to the integral under pdf1 for all random variables X for which pdf1 > pdf2. Let us define S1 as a subspace of the input space for which pdf1 > pdf2. In order to obtain the value of integral $\int_{S1} (\text{pdf1})\, dx$, a number of schemes were used and are discussed further.

3.1 Cartesian Grid Method: The motivation for this approach arose from the simple way of integral approximation. The input space is divided into equally sized segments and integral over each segment is approximated by multiplying the volume of the each segment by the average value of the integrated function over this volume. Total integral is approximated by the sum of the segment integrals. Initially a unit grid centered at the origin is made. Each dimension is divided into equal number of parts given by 'numPoints'. In order to map the unit grid to the actual points we multiply the original grid points by maximum standard deviation of the original points and a constant named 'varstd'. The value of this constant is chosen so that statistically almost 98% of the points are covered. Having obtained the grid of points in cartesian coordinates we now perform QR factorization of the actual signal shifted to the origin of the coordinate system. The resulting two matrices Q and R will be used to obtain a linear transformation of the input space, where Q is the orthogonal matrix and R is the upper triangular matrix. Having Q and R we can easily obtain an N-dimensional ellipsoid which encloses the original points in the input space and can be used to describe n-dimensional probability density function (pdf) of the data points in the original input space. The original data points can be transformed to the orthogonal space by using multiplication of the input signals by R^{-1}. In the orthogonal space, all data points are equally spread around the origin. Their distribution is approximately Gaussian with the mean value equal to zero and standard deviation equal to 1/sqrt(m-1), where m is the number of data points (also equal to standard deviation of the matrix Q). To include most of the data points in an

enclosing n-dimensional sphere we multiply the standard deviation by 2.3 (includes close to 98% of all normally distributed samples).

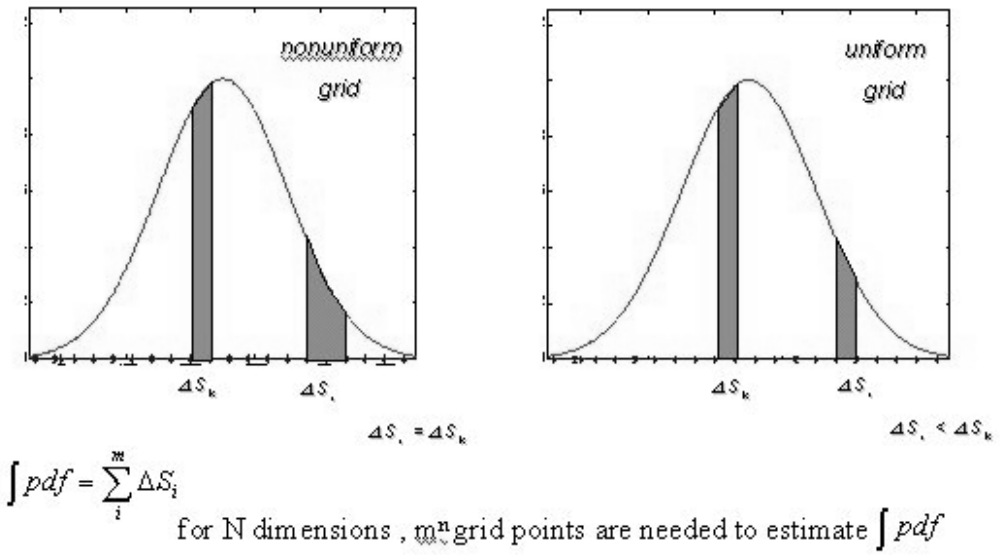


Fig 3.2: Illustration of uniform and non-uniform grid for direct integration in 1D.

The enclosing sphere (and the Gaussian probability density function) can be transformed back to the original input space by taking product of vectors in the orthogonal space with matrix R. Therefore, the enclosing sphere will be transformed to enclosing ellipsoid in the input space. Subsequently, all the grid points are transformed to the orthogonal space spanned by column vectors of matrix Q and points lying outside the enclosing n-dimensional sphere are discarded. Now, we have reduced the earlier generated grid to a selected set of points over which we are going to calculate the pdf. The program that generates the Cartesian grid is called cartplot.m. Figure 3.3 illustrates the 2-dimensional grid. It shows the original points in ‘o’ and the original grid points with '+'. The ellipse

which bounds the original set of points is shown in red with '-' and the grid points which are selected and lie within the ellipse are shown with '*'.

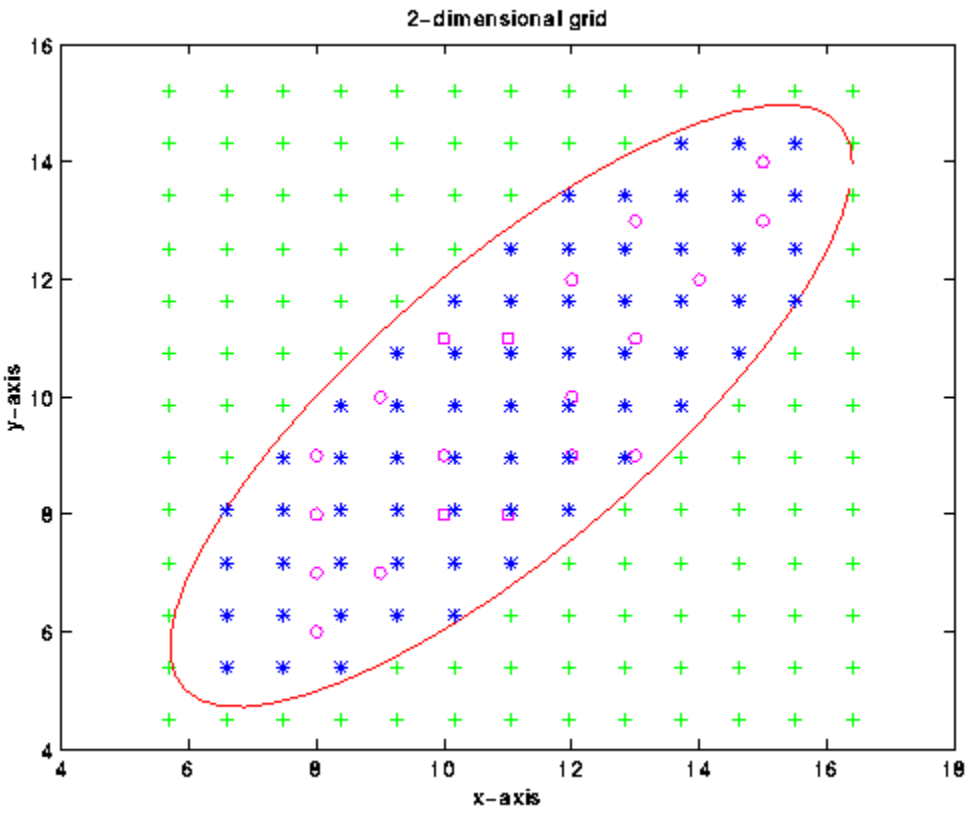


Fig. 3.3: 2-dimensional grid.

red	**(.):**	**ellipse bounding the signal**
green	**(+):**	**original set of points**
magenta	**(o):**	**original signal**
blue	**(*):**	**selected set of points**

Figures 3.4 and 3.5 illustrate the 3-dimensional process. The original set of points and the selected grid points are shown in Figure 3.4, while the original grid points and the ellipse which bounds the original set of data points are shown in Figure 3.5.

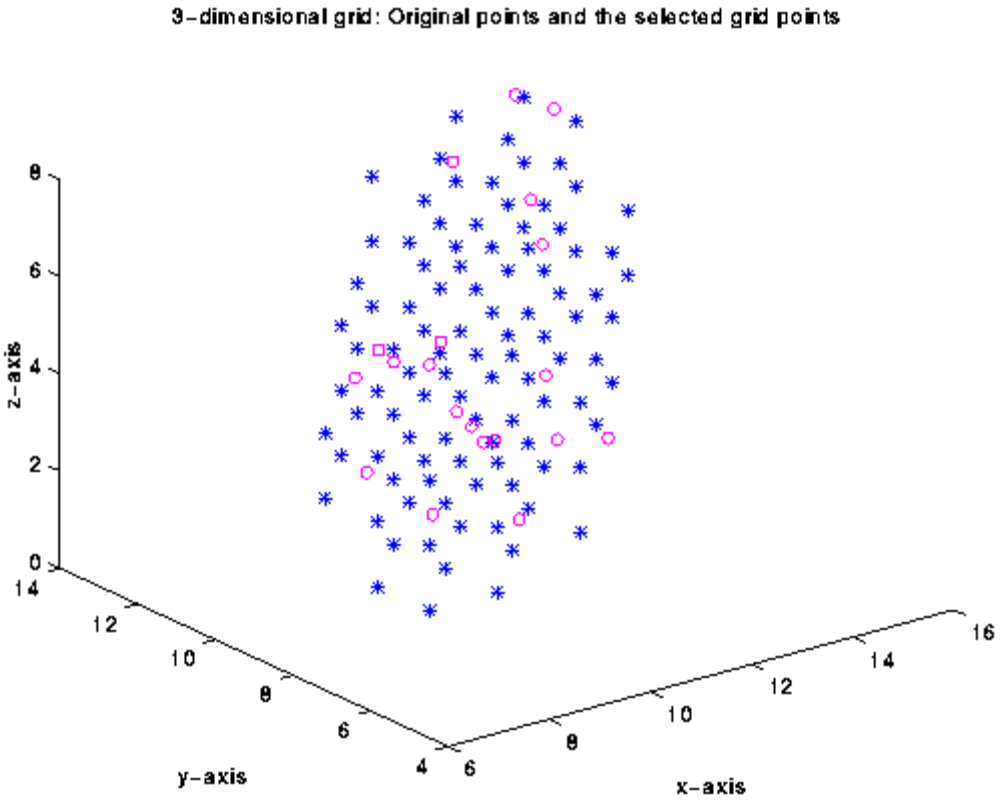


Fig. 3.4: 3-dimensional grid: Original points and the selected grid points.

magenta	**(o):**	**original set of points**
blue	**(*):**	**selected set of points**

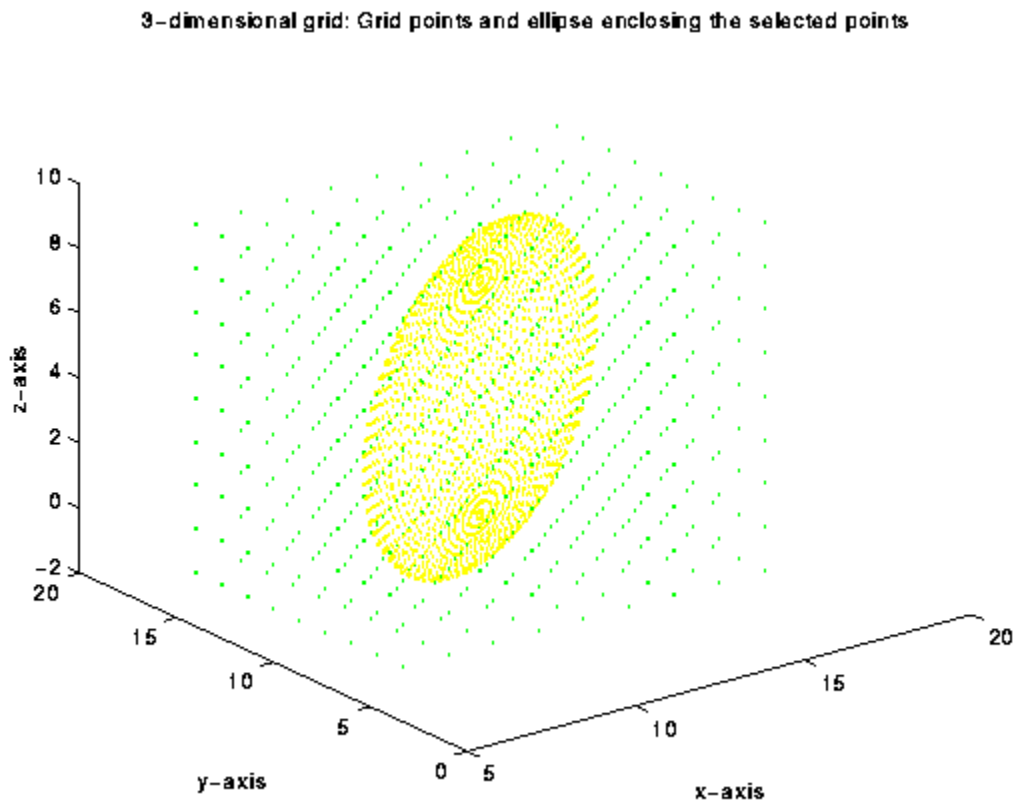


Fig. 3.5: 3-dimensional grid.

yellow (.): ellipse bounding the signal
blue (.): original set of points

3.2 Spherical Grid Method: Cartesian grid approach is simple to use, however it is not the most numerically efficient way of finding n-dimensional integrals. Considering specificity of Gaussian functions (exponentially declining values) significant savings in computations can be obtained. The same accuracy can be obtained with non-uniform distribution of the grid points and total number of grid points reduced. First, a regular n-dimensional grid is obtained using polar coordinates. Polar coordinates are described by specifying the radius and angles in all directions. The increment in radius Δr_1 is obtained from the function **intstep**. This function evaluates the steps for integration of normal 1-dimensional pdf under the assumption that discrete integration using selected points introduces equal increments of the integral values. Once the increments in radius are known we make the grid in polar form. The angles vary from 0 to π and radius varies from 0 to maximum value obtained from intstep function. Starting from the 2-dimensional case the projection of the radius vector is used to determine the radius of next higher dimension such that the resulting grid points are equally distributed on the surface of n-dimensional sphere with radius r_1.

To illustrate this process let us assume that we want to obtain a radius and a number of grid points on a single circle around a 3-dimensional sphere. We assume that the sphere radius is r_1 and the projection angle $\alpha 1$ is used. The radius r_2 is obtained from projection of r_2 using

$$r_2 = r_1 * \cos\alpha 1$$

as shown in Figure 3.6.

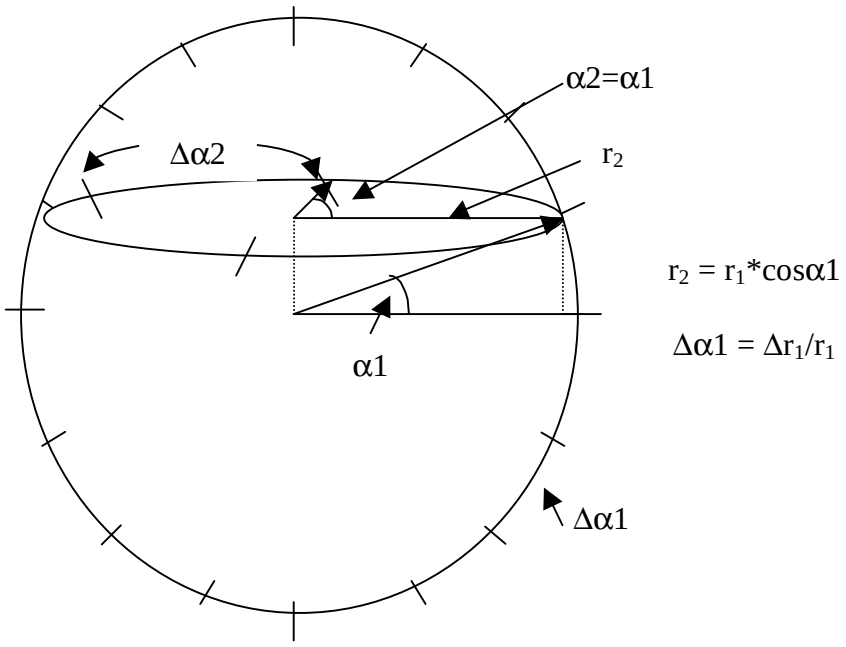


Fig. 3.6: Figure indicating how to obtain radius projection into other dimension.

Radius r_2 is needed to determine values of $\alpha 2$, where $\alpha 2$ has the increment

$$\Delta\alpha 2 = \Delta r_1/r_2$$

$\alpha 2$ starts from 0 and is incremented by $\Delta\alpha 2$ until it reaches π. The number of grid points on this circle is less than or equal to $2\pi r_2/\Delta r_1$. Subsequently r3, which is a radius of a circle around 4-dimensional sphere, is obtained using:

$$r_3 = r_2 * \cos\alpha 2$$

Values of $\alpha 3$ are also computed from increment in $\alpha 3$ which is given by:

$$\Delta\alpha 3 = \Delta r_1/r_3$$

This continues until the highest dimension is reached. Then with the radius r1 and varying angles from 0 to π in each dimension we obtain a set of grid points equally

distributed on the surface of n-dimensional sphere with radius r1. A function **Ndsph** was written which generates the entire grid points on this sphere by calling itself recursively. The procedure Ndsph is repeated for each r1 value which was determined by intstep. The result is a polar grid of points that fill the upper half of n-dimensional volume. Examples of generated polar grids for 2 and 3-dimensional volumes are as shown on Figure 3.7 and Figure 3.8.

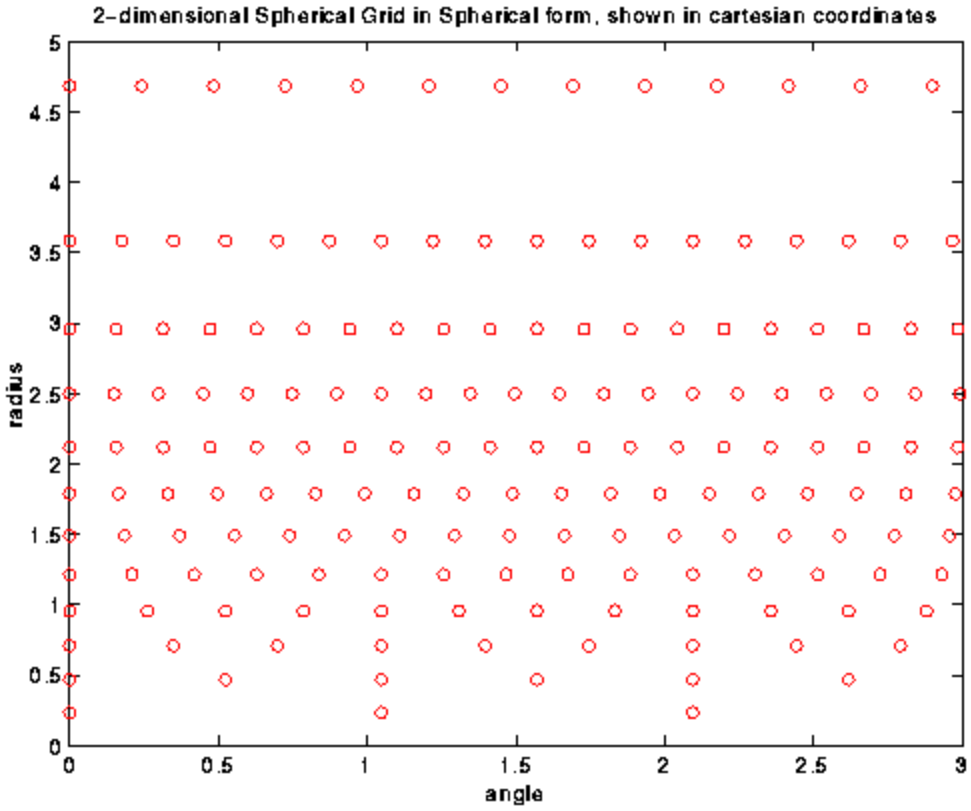


Fig. 3.7: 2-dimensional Spherical Grid in rectangular coordinates.

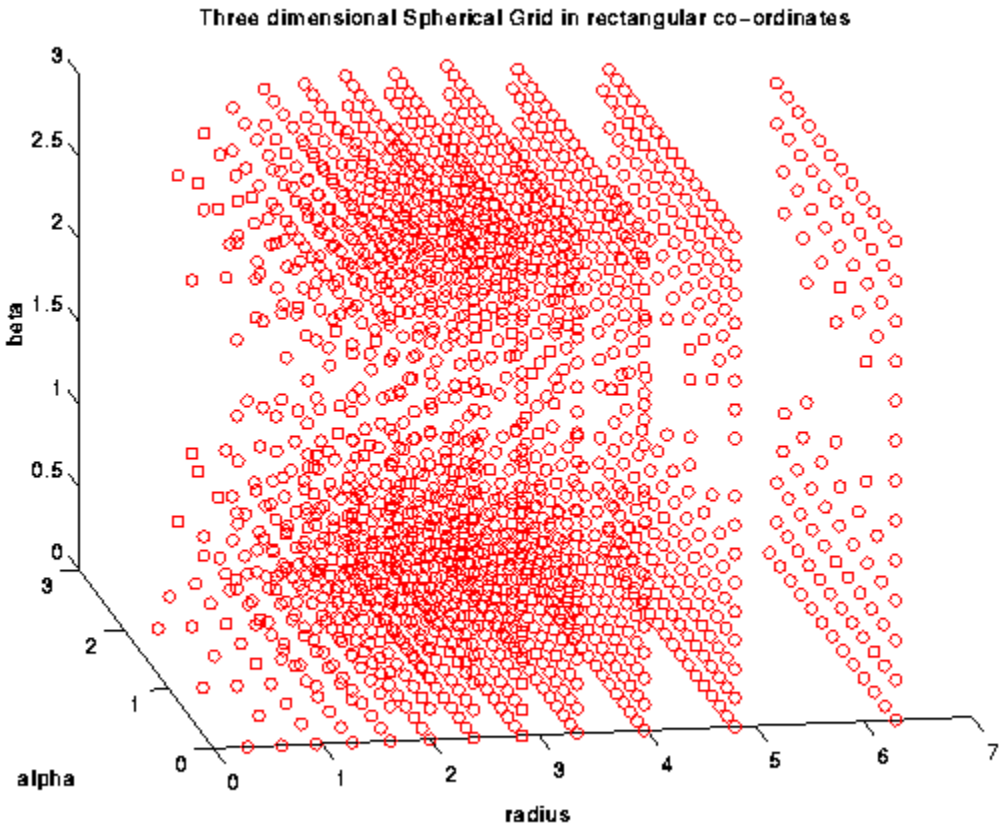


Fig. 3.8: 3-dimensional Spherical Grid in rectangular coordinates.

These spherical points are then converted to the Cartesian set of points. The formulas used to convert the spherical coordinates to the Cartesian form are as follows:

$$X_n = r * \sin\alpha_{n-1}$$

$$X_{n-1} = r * \cos\alpha_{n-1} * \sin\alpha_{n-2}$$

$$X_{n-2} = r * \cos\alpha_{n-1} * \cos\alpha_{n-2} * \sin\alpha_{n-3}$$

.

.

$$X_2 = r * \cos\alpha_{n-1} * \cos\alpha_{n-2} * \cos\alpha_{n-3} *.....* \cos\alpha_2 * \sin\alpha_1$$

$$X_1 = r * \cos\alpha_{n-1} * \cos\alpha_{n-2} * \cos\alpha_{n-3} *.....* \cos\alpha_2 * \cos\alpha_1$$

For instance, the Cartesian form of the 2-dimensional grid shown in Figure 3.7 is illustrated in Figure 3.9. This grid needs to be complemented to fill full n-dimensional volume by symmetrical projection around the coordinate center, as well as adding the center point. The resulting grid for 2-dimensional case is shown in Figure 3.10.

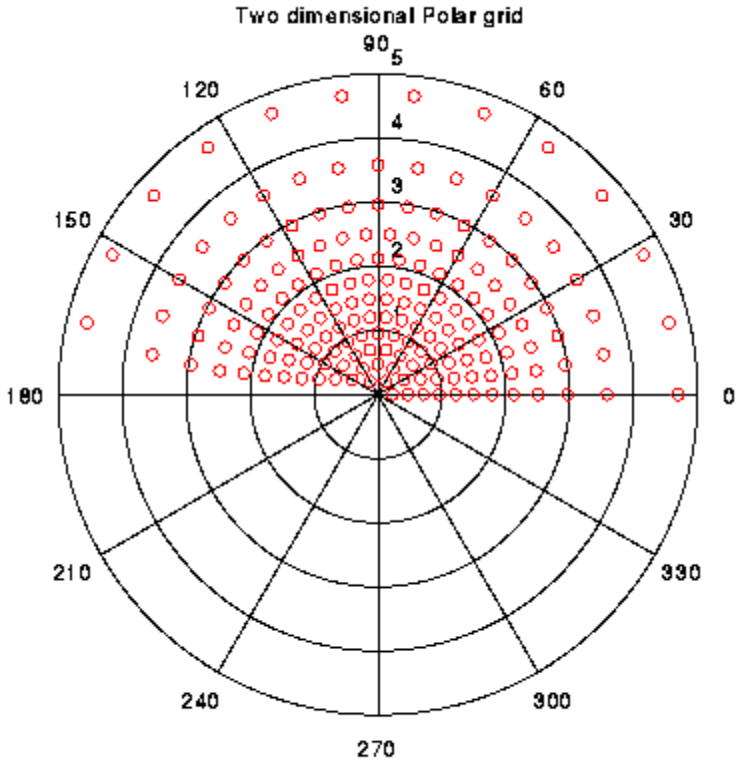


Fig. 3.9: 2-dimensional Spherical Grid in polar form.

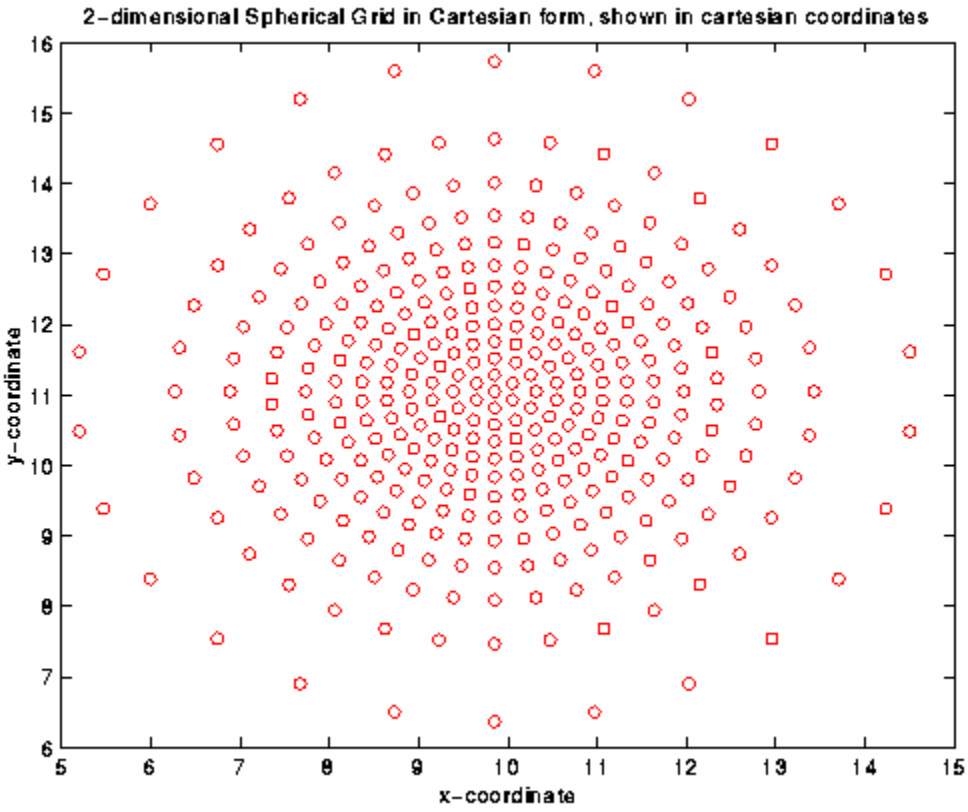


Fig. 3.10: 2-dimensional Spherical Grid in Cartesian from, shown in cartesian coordinates.

Once the grid is converted from polar coordinates to Cartesian form, transformation based on QR factorization is applied and the grid points lying outside the unit circle are discarded. This method of QR factorization and discarding the points outside the unit circle is the same as described in Cartesian grid described previously. The program that makes the spherical grid is called **sphnd.m** and it calls two routines **intstep** and **Ndsph**. These programs are attached in Appendix. Figure 3.11 illustrates the grid selection process in which the '+' sign shows the original grid points and 'o' shows the original signal points, '*' indicate the selected grid points and '-' in red shows the ellipse which bounds the original set of points and bounds the selected grid points.

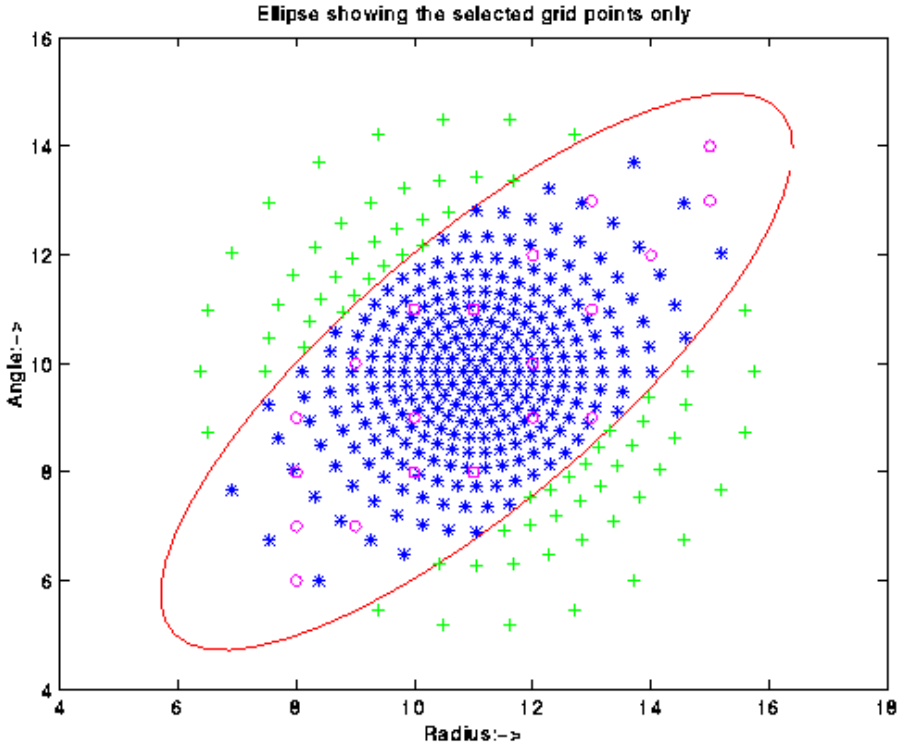


Fig. 3.11: 2-dimensional Spherical grid.

red	**(-):**	**ellipse bounding the signal**
green	**(+):**	**original set of points**
magenta	**(o):**	**original signal**
blue	**(*):**	**selected set of points**

Figures 3.12 and 3.13 show the 3-dimensional case. Figure 3.12 shows the ellipse and the original set of points, while Figure 3.13 shows the original grid and selected grid points

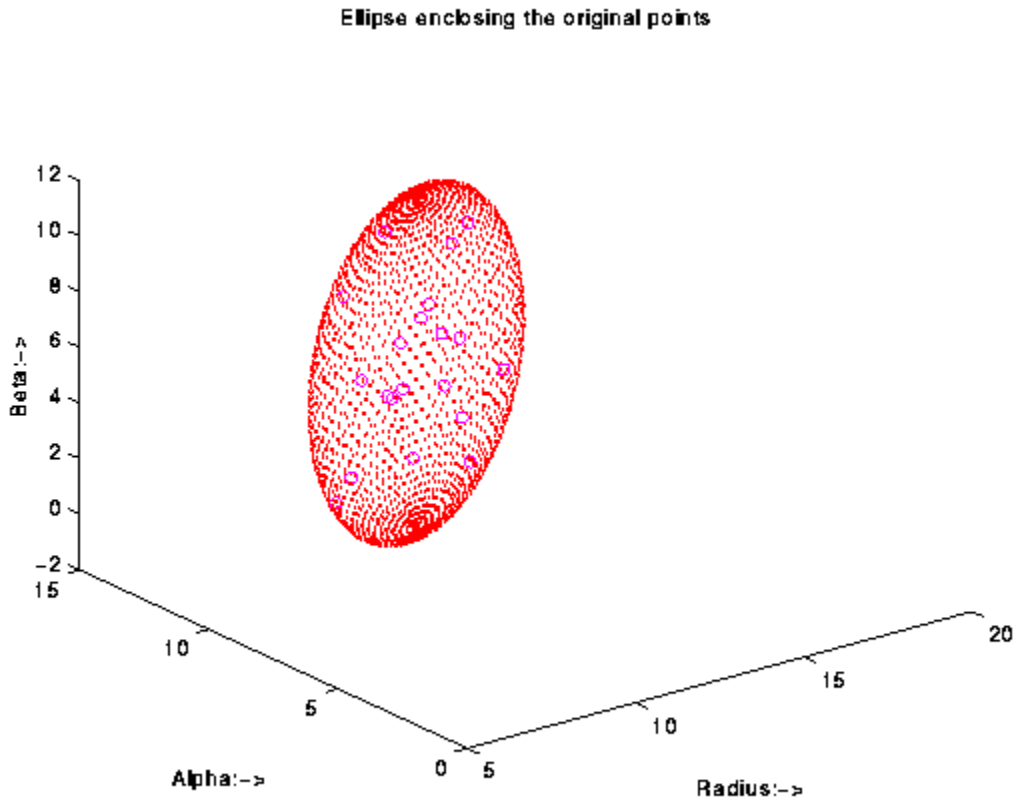


Fig. 3.12: Ellipse enclosing the original set of points.
Magenta (o): Original Points
Red (.): Ellipse enclosing the original points

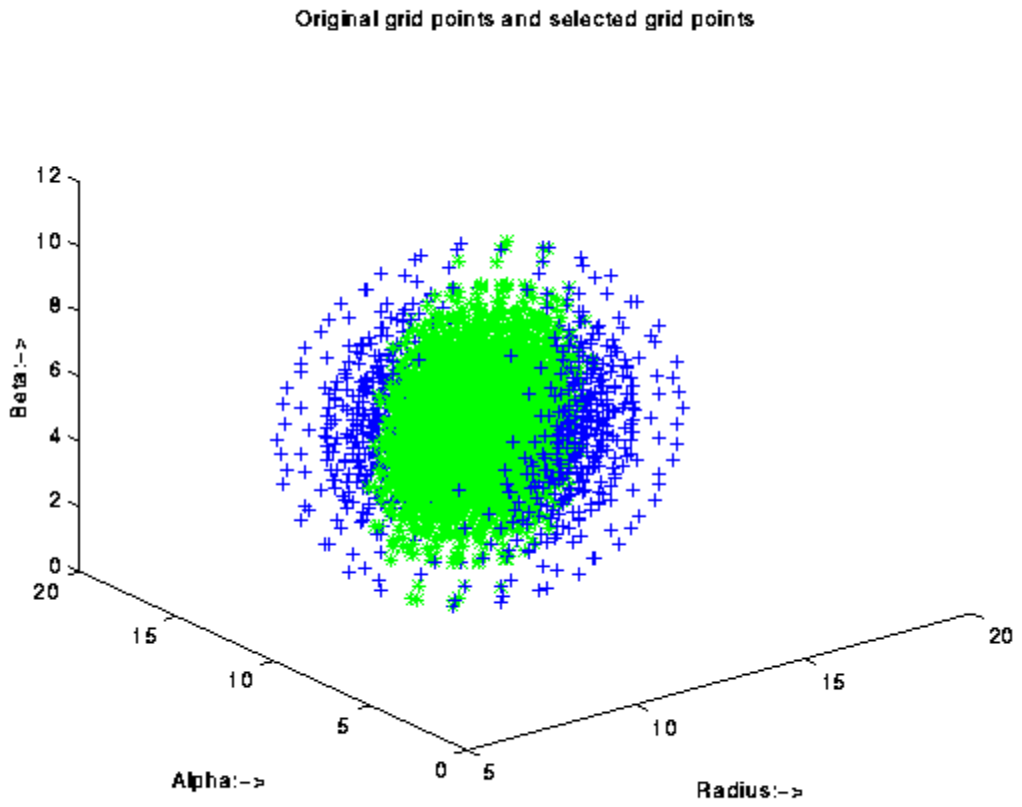


Fig. 3.13: Original and Selected Grid Points.

Blue (+): Original Grid Points
Green (*): Selected Grid Points

3.3 Monte Carlo Method: One of the major drawbacks of Cartesian and Spherical approach was that with increase in dimension, the number of grid points generated grew exponentially. Monte Carlo method eliminates this problem by generating a fixed known number of random points to approximate the integral. In Monte Carlo integration the estimate of integral $\int_{S1} (\text{pdf1})\, dx$, is obtained by counting how many random points generated by pdf1 have pdf1 > pdf2. Then the probability of correct classification for class 1 is approximated from:

$$\text{Probability} = \frac{\text{Total number of random points for which pdf1 > pdf2}}{\text{Total number of random points generated}}$$

As shown in figure 3.14 all points to the left of point Z have pdf value greater for pdf1 than for pdf2.

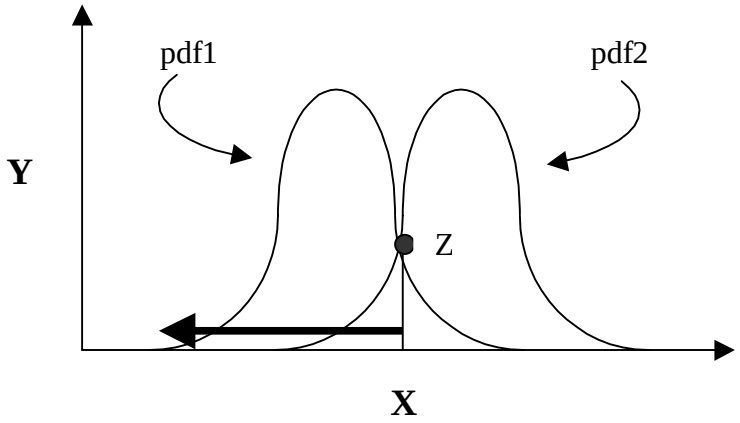


Fig. 3.14: Probability based on pdf1 > pdf2.

A slight modification of the above scheme is to use weighted pdfs. Now a point is classified as a point of class 1 with the believe level determined by:

$$\frac{\text{pdf}_1(x)}{\text{pdf}_1(x) + \text{pdf}_2(x)}$$

and as a point of class 2 with the believe level determined by:

$$\frac{pdf_2(x)}{pdf_1(x) + pdf_2(x)}$$

As a result of weighted pdf functions we need to estimate their integrals with weighted probabilities. Figure 3.15 shows the region of each weighted probability.

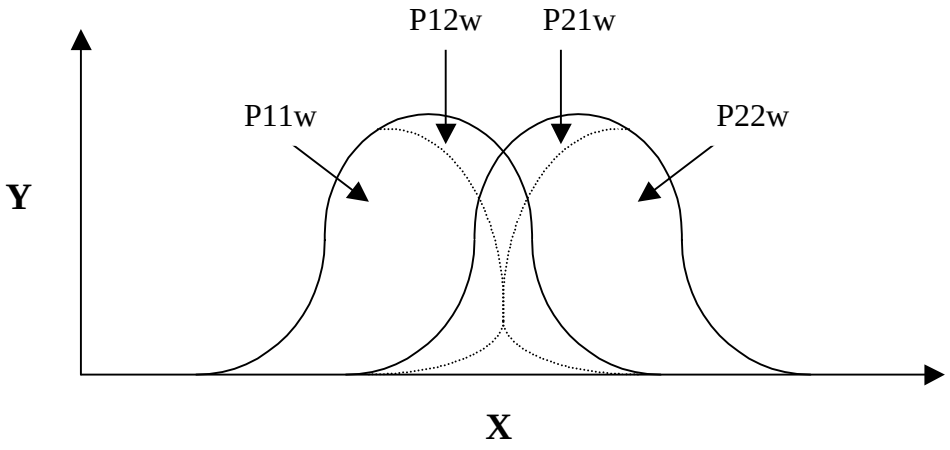


Fig. 3.15: Region of each weighted probability.

Figure 3.16 illustrates the generation of points in the Monte Carlo method.

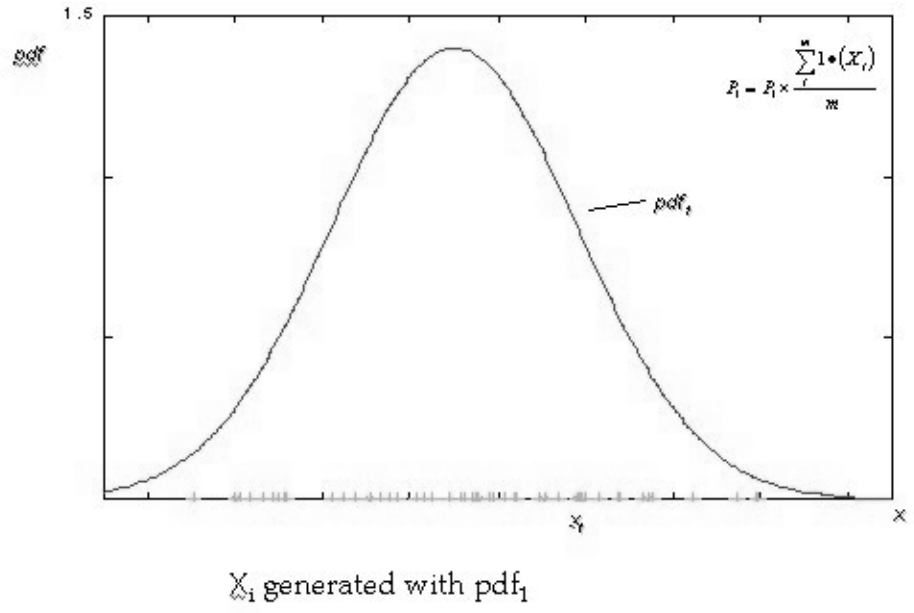


Fig 3.16 Illustration for Monte-Carlo integration approach – generation of points.

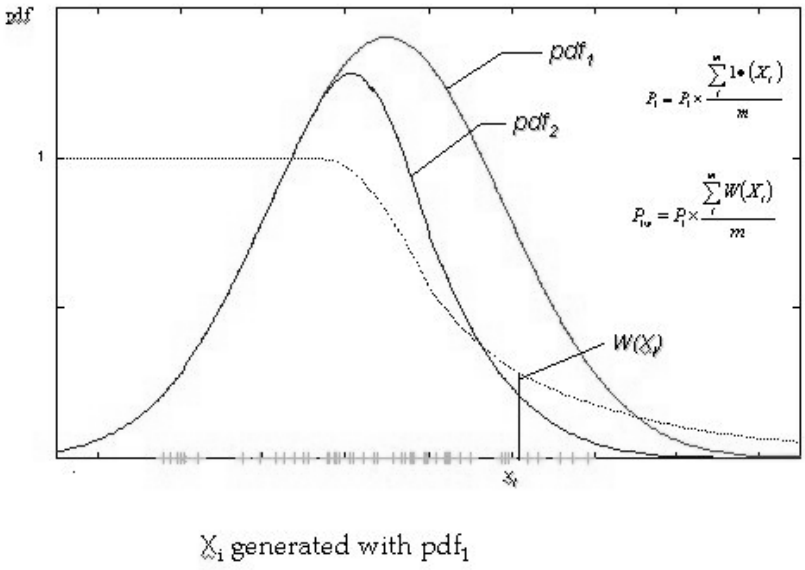


Fig. 3.17 Illustration for Monte-Carlo integration approach – probability calculation.

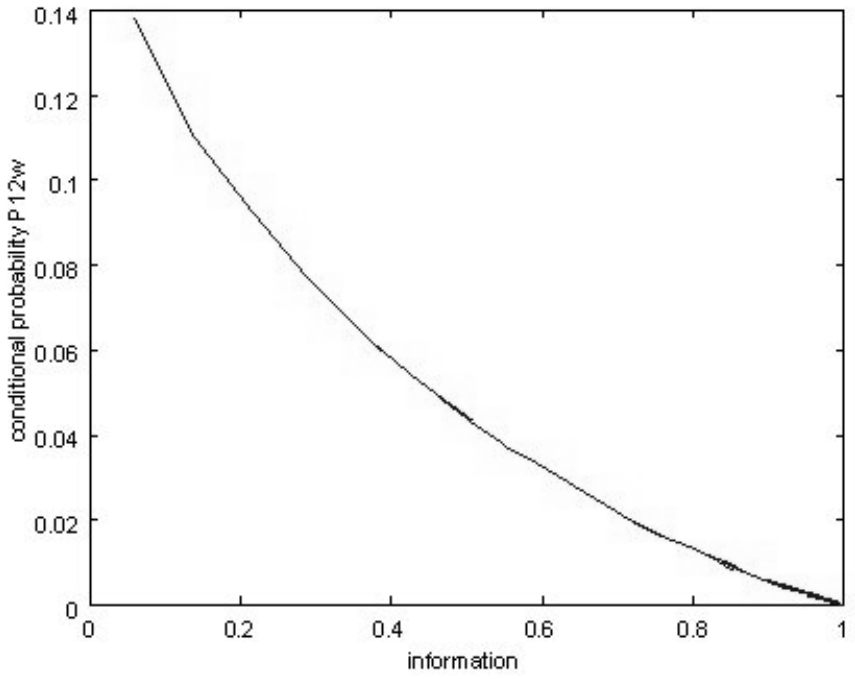


Fig. 3.18 Relationship between information index and probability of misclassification.

As is clear from figure 3.18, in order to achieve a high information index, we need to correctly classify the points i.e. probability of misclassification should be minimum.

The information is determined as

$$\text{info} = 1 - \frac{\text{entr}}{\text{maxentr}} \tag{3.1}$$

where

$$\text{entr} = -(P1.*\log(P1) + P2.*\log(P2)) + P1w.*\log(P1w) + P2w.*\log(P2w) + P12w.*\log(P12w) + P21w.*\log(P21w) \tag{3.2}$$

and

$$\text{maxentr} = -(P1.*\log(P1) + P2.*\log(P2)) + P11.*\log(P11) + P22.*\log(P22) + 2 * P12.*\log(P12) \tag{3.3}$$

where

$$P11 = \frac{P1^2}{P1 + P2} \qquad P22 = \frac{P2^2}{P1 + P2} \qquad P12 = \frac{P1 * P2}{P1 + P2}$$

newalgma.m was the main program written for this and it calls two routines **infopdfV.m** for one dimensional analysis and **infoND.m** for higher dimension analysis.

Chapter 4

Wavelet based transformation

Wavelets are a class of linear transformations of the input data for the purpose of data compression or representation. Wavelets cut up data into different frequency components, and then analyze each component with a resolution matched to its scale. The scale that we use to look at data plays a special role [8]. Wavelet algorithms process data at different scales or resolutions. They have advantages over traditional Fourier methods in analyzing physical situations where the signal contains discontinuities and sharp spikes. Fourier basis functions are localized in frequency but not in time. Small frequency changes in the Fourier transform will produce changes everywhere in time domain. Wavelets are local in both frequency and time domain. With wavelet analysis, we can use approximating functions that are contained neatly in finite domains. This makes wavelets useful in signal processing, data transmission and pattern recognition. Because the original signal or function can be represented in terms of a wavelet expansion (using coefficients in a linear combination of the wavelet functions), data operations can be performed using just the corresponding wavelet coefficients. Moreover, if we choose the best wavelet coefficients adapted to our data, (or truncate the wavelet coefficients below a specified threshold), the data is sparsely represented and require fewer resources for storage, transmission and processing. Due to this sparse coding, FBI has standardized the use of wavelets in digital fingerprint image compression.

4.1 The Haar Wavelet: The Haar wavelet is the simplest and oldest of all wavelets. It is generated by the step function taking values 1 and –1, on $[0, |)$ and $[|, 1)$, respectively. Figure 4.1 shows the graph of Haar wavelet.

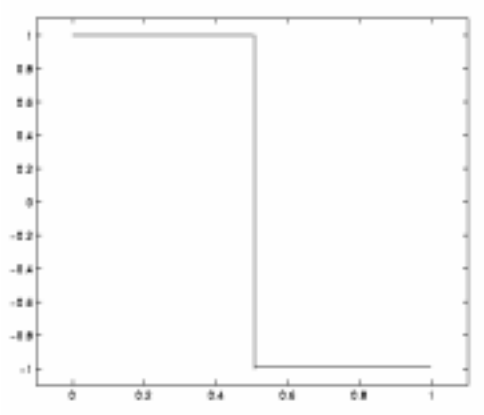


Fig. 4.1: Haar Wavelet.

One property of the Haar wavelet is that it has compact support, which means that it vanishes outside of a finite interval. Its discrete value makes it easy to represent and manipulate on a digital computer. It is also very easy to build a dedicated processing hardware that implements Haar wavelet. Unfortunately, Haar wavelets are not continuously differentiable which somewhat limits their applications.

In order to see how Harr transform works, we first consider two numbers a and b. The average and difference is given as

$$\text{avg} = \frac{(a + b)}{2} \qquad \text{diff} = (a - b)$$

Now in order for these numbers (avg and diff) to be useful, we must be able to reconstruct the original numbers from them. In the Haar transform the above two equations are the basic equations used. Let us assume an input signal of length N which

we assume to be a power of 2 for simplicity, a_0 a_1 a_2 a_3 .... a_{N-1}. The Haar transform is defined as

$$H_i = \frac{(a_{2*i} + a_{2*i+1})}{2} \qquad \text{where } i = 0 \ldots (N/2) - 1 \quad \text{(average components)}$$

$$H_{(N/2)+i} = (a_{2*i} - a_{2*i+1}) \qquad \text{where } i = 0 \ldots (N/2) - 1 \quad \text{(difference components)}$$

The following table shows an example of a signal represented by 8 discrete values and results on the first level transformation obtained by the presented equations.

(a) Haar Transform Example								
Input Signal	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
	0.0	0.5	1.0	0.5	0	-0.5	-1	-0.5
Haar Coefficients	0.25	0.75	-0.25	-0.75	-0.5	0.5	0.5	-0.5
	average components $H_i=(a_{2*i}+a_{2*i+1})/2$ where i = 0 ... 3				difference components $H_{4+i}= a_{2*i} - a_{2*i+1}$ where i = 0 ... 3			

Table 4.1: Haar Transform example.

In the preceding example, the Haar transform was run once (or one level) over the input signal. The difference component of the first level tells us where dramatic local changes in the input signal occur. Large differences indicate large change between adjacent input

signal values. Differences near zero tell us that adjacent input signal values are somehow similar. By running the Haar transform again on the first level average and difference components we can generate information about a larger part of the input signal since each new coefficient contains information about 4 adjacent input signal values. We repeat the process until we obtain single average and difference components (which are defined on the all input signal values). Stated succinctly, each successive transform level reveals courser frequency (change) information about a larger part of the input signal. If the signal has N coefficients, then the Haar transform can be run $\log_2$ (N) times producing N coefficients in each run. Table 2 shows the value of Haar wavelet coefficients at each level and Fig. 4.2 show waveforms that represent the obtained coefficients after each loop of the Haar transform algorithm. Notice that in this waveform interpretation coefficients represent the inner product of the input signal and the corresponding waveform.

Input Signal	0	0.5	1	0.5	0	-0.5	-1	-0.5
Level 1	0.25	0.75	-0.25	-0.75	-0.5	0.5	0.5	-0.5
Level 2	0.5	-0.5	-0.5	0.5	0	0	-1	1
Level 3	0	1	0	-1	0	0	0	-2

Table 4.2: Haar wavelet coefficients at each level.

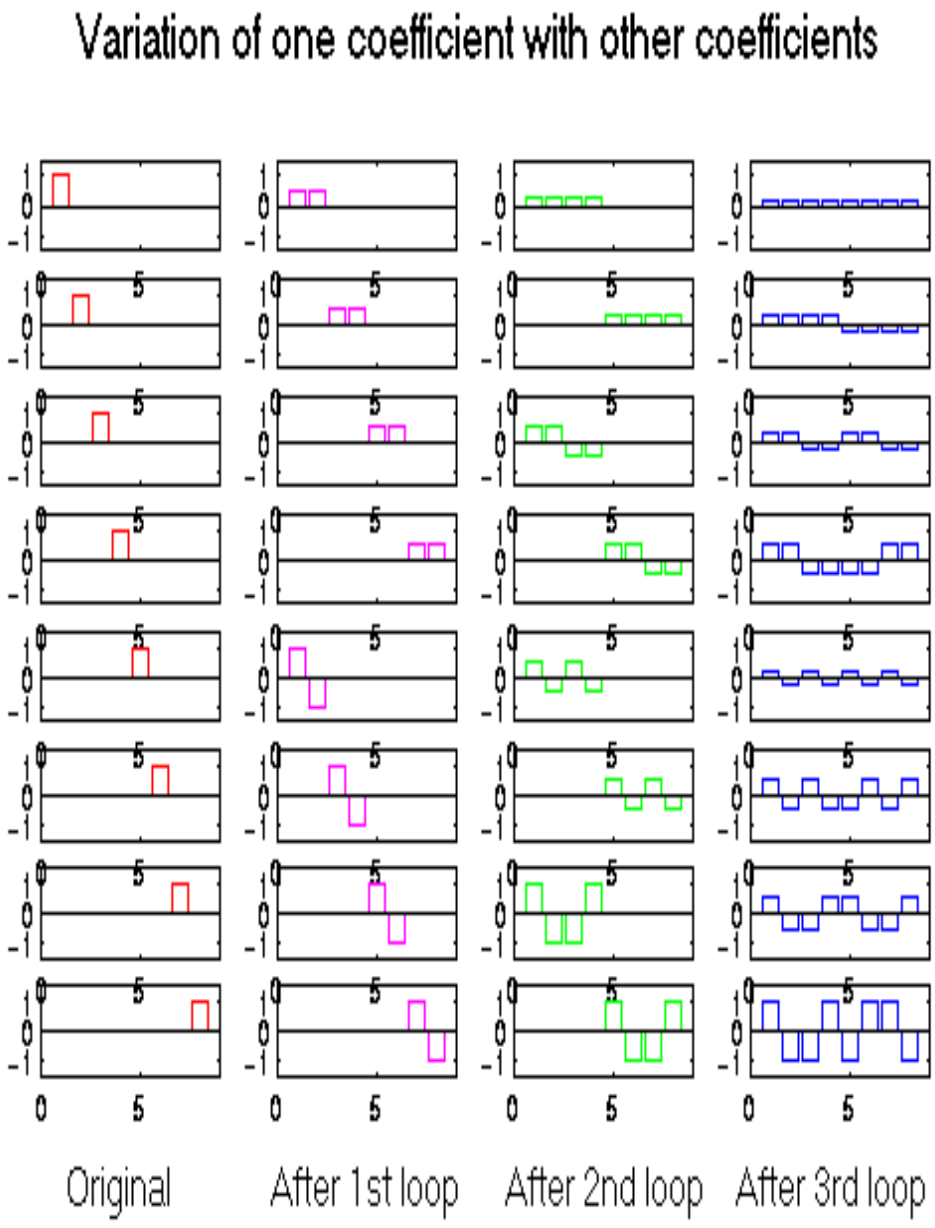


Fig. 4.2: Waveform interpretation of Haar coefficients.

The following section explains how the Haar transform was used to determine the input space transformation which maximized information in 1-dimension.

4.2 1 Dimensional Analysis: We start with two classes of signals and perform Haar transform on them. Once the Haar transform has been performed, the next step is to evaluate the entropy measure. The entropy measure and its evaluation will be discussed in the following section. The information thus obtained shows which coefficient if used in signal preprocessing, will most differentiate the input signals for classification. The coefficients are reordered in descending order of information content and we select N coefficients with the maximum information content. Now we are ready for the next iteration. The program iterates until the change in the maximum information in successive iterations is less than some epsilon value. At the end of all iterations, we have used Haar wavelet to determine the best coefficients needed to obtain maximum information for the classification problem. The obtained transformation of the input signals is now much more complex than the original Haar wavelets and in fact represents a dedicated transformation of the input data optimum for a given classification problem. It was automatically generated by repetitive use of the Haar wavelet. The main program for the 1-dimensional analysis is **onedim.m** and it calls two routines, **constant.m**, which declares all the constants and, **infopdfV.m** which determines the entropy.

4.3 Entropy Determination: First the input signals are divided into classes and each class probability density function is estimated using its mean and standard deviation by:

$$pdf = \frac{P * \exp(-0.5 * (\frac{x - \bar{x}}{\sigma})^2)}{\sqrt{2\pi}\sigma} \tag{4.1}$$

$\bar{x}$: mean , σ : standard deviation

The weighted probability density functions pdf_{xy} are determined based on the probability density functions of signals from different classes. The information index is determined as

$$\text{info} = 1 - \frac{\text{entr}}{\text{maxentr}} \tag{4.2}$$

where

$$\text{entr} = -(P1.*\log(P1) + P2.*\log(P2)) + P1w.*\log(P1w) + P2w.*\log(P2w) + P12w.*\log(P12w) + P21w.*\log(P21w) \tag{4.3}$$

and

$$\text{maxentr} = -(P1.*\log(P1) + P2.*\log(P2)) + P11.*\log(P11) + P22.*\log(P22) + 2 * P12.*\log(P12) \tag{4.4}$$

The following figures show some examples. Figure 4.3 shows the original signal, figure 4.4 the signal after noise is added, figure 4.5 the measure of information during each iteration, and figure 4.6 the way the coefficients were mixed to obtain the maximum information.

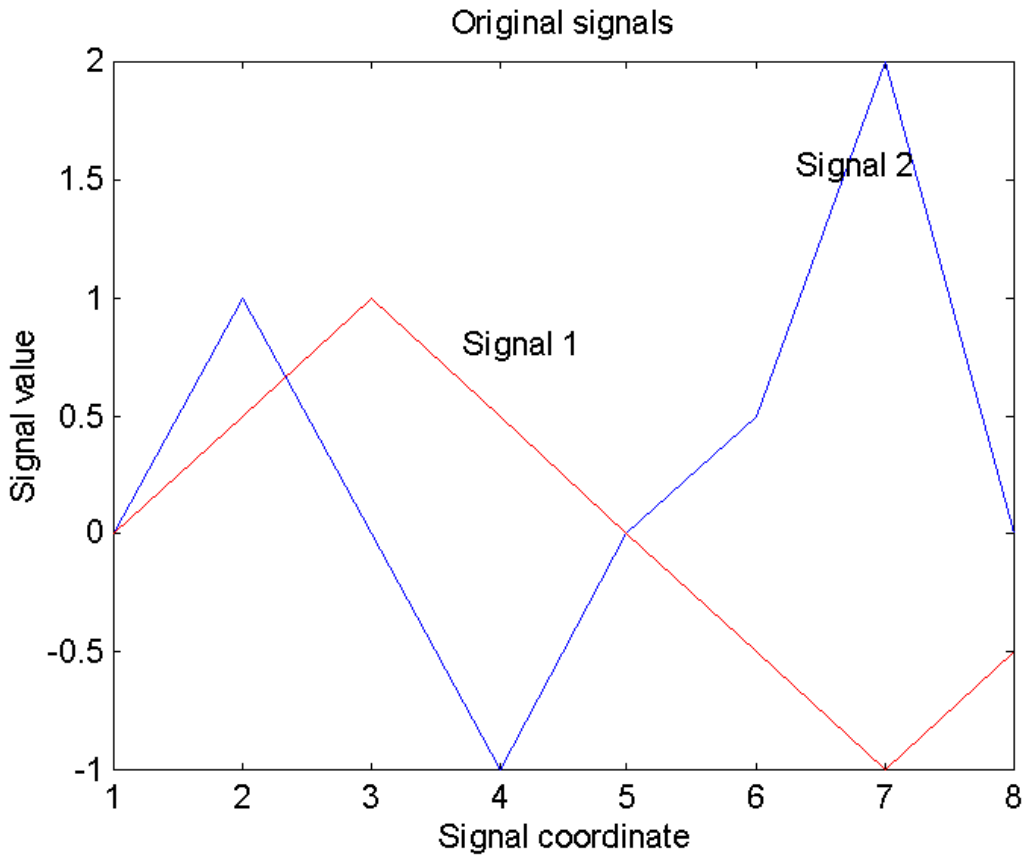


Fig. 4.3: Original signals.

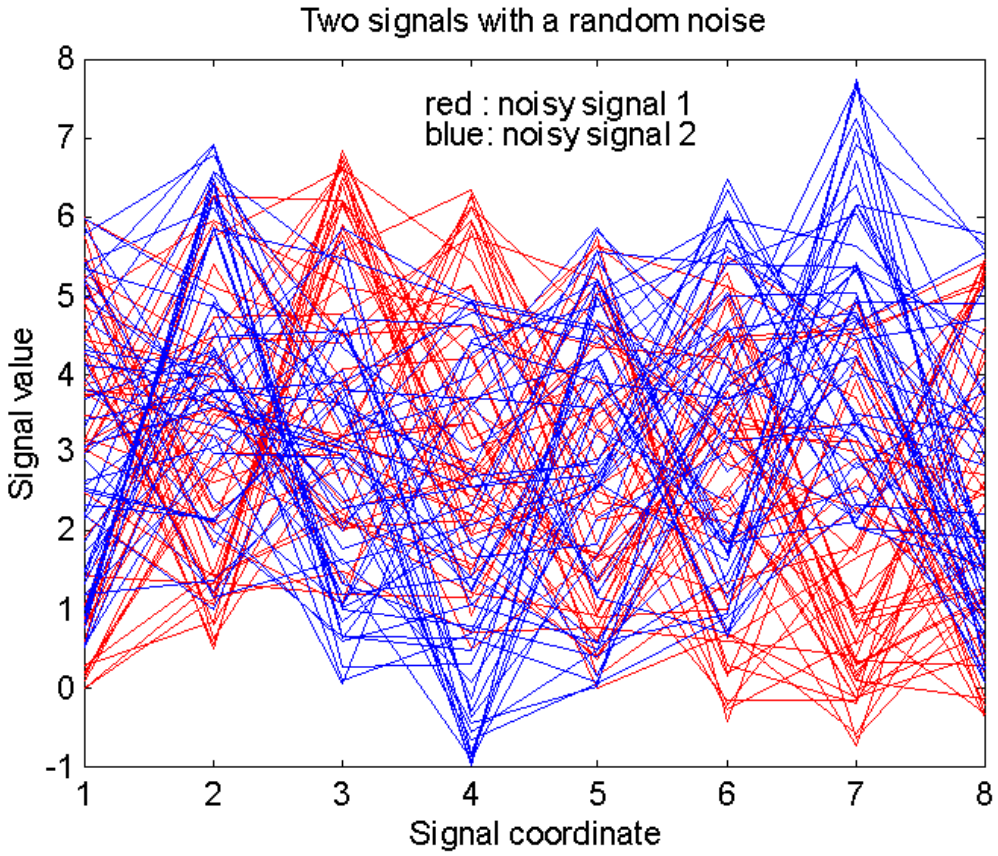


Fig. 4.4: Original signals with noise.

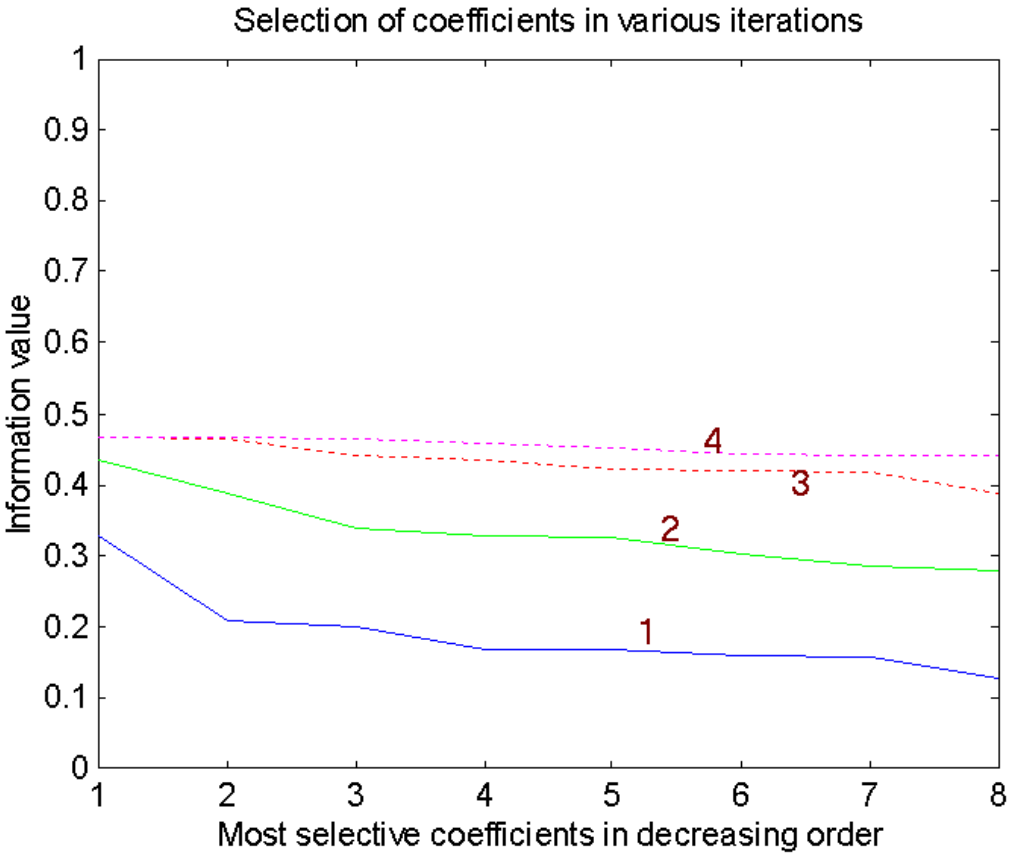


Fig. 4.5: Information measure after each iteration.

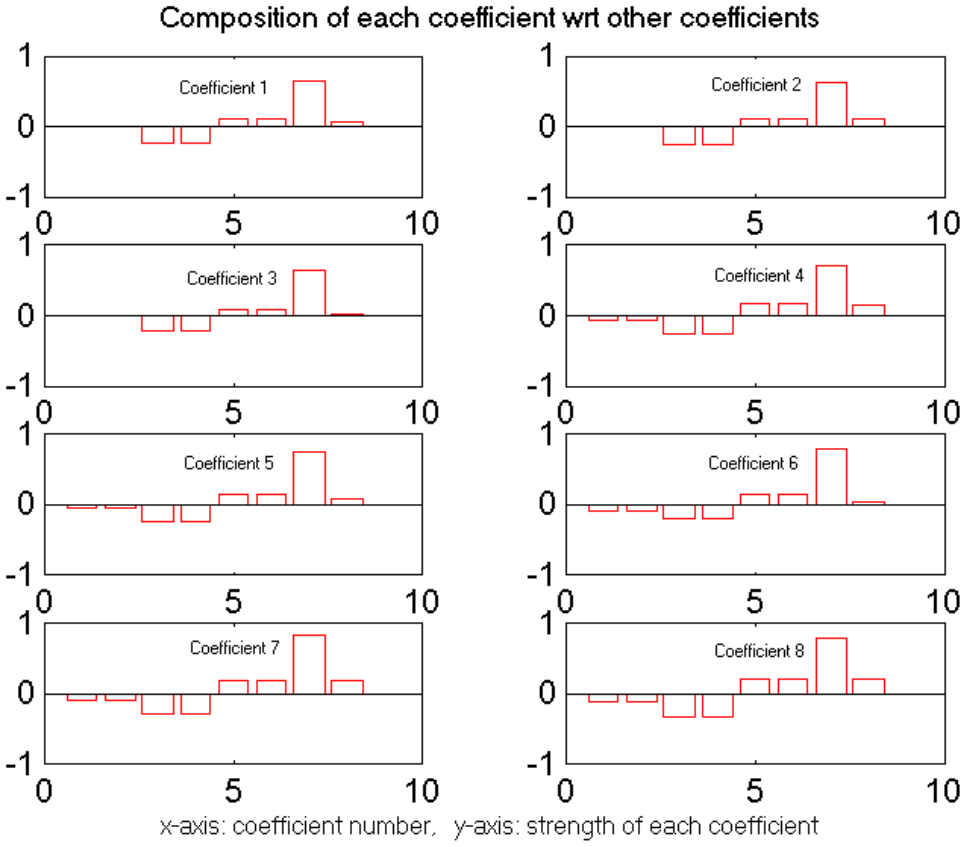


Fig. 4.6: Waveform interpretation of the selected coefficients.

In the next two examples, the two signals contain 128 coefficients each. A class of input signals was then generated from the two input signals by adding random noise. Then Haar wavelet transformation and the selection of coefficients was performed and repeated iteratively until there was no further improvement in the information index. Figure 4.7 and 4.11 shows the original signals and the waveform interpretation of the obtained transformation of the input signals that best differentiates the two signals. As we can see, the optimized transformation correctly learned where the two signals are different and it applies different weight to different areas of the input signals to stress the varying importance of the signals difference. Figure 4.8 and 4.12 shows the information index after each iteration for only 8 best coefficients, figure 4.9 and 4.13 shows the information after each iteration for all 128 coefficients, and figure 4.10 and 4.14 shows the way first four coefficients were composed.

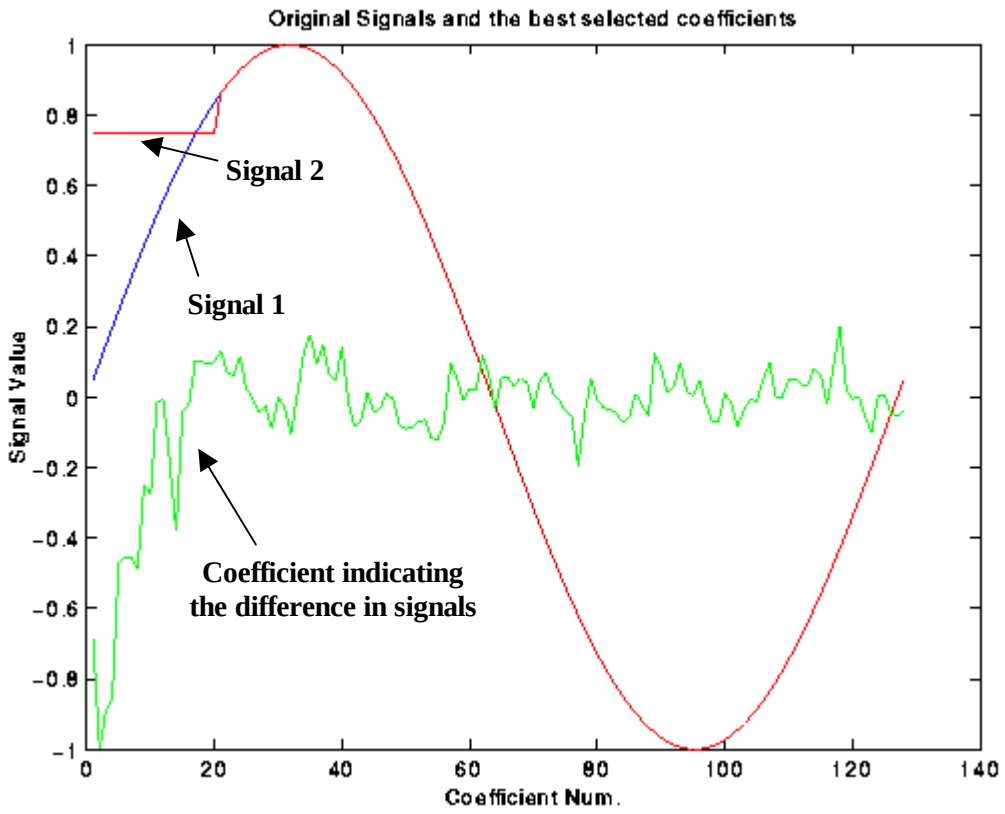


Fig. 4.7: Original signals and the evolved waveform that best separates the two signals.

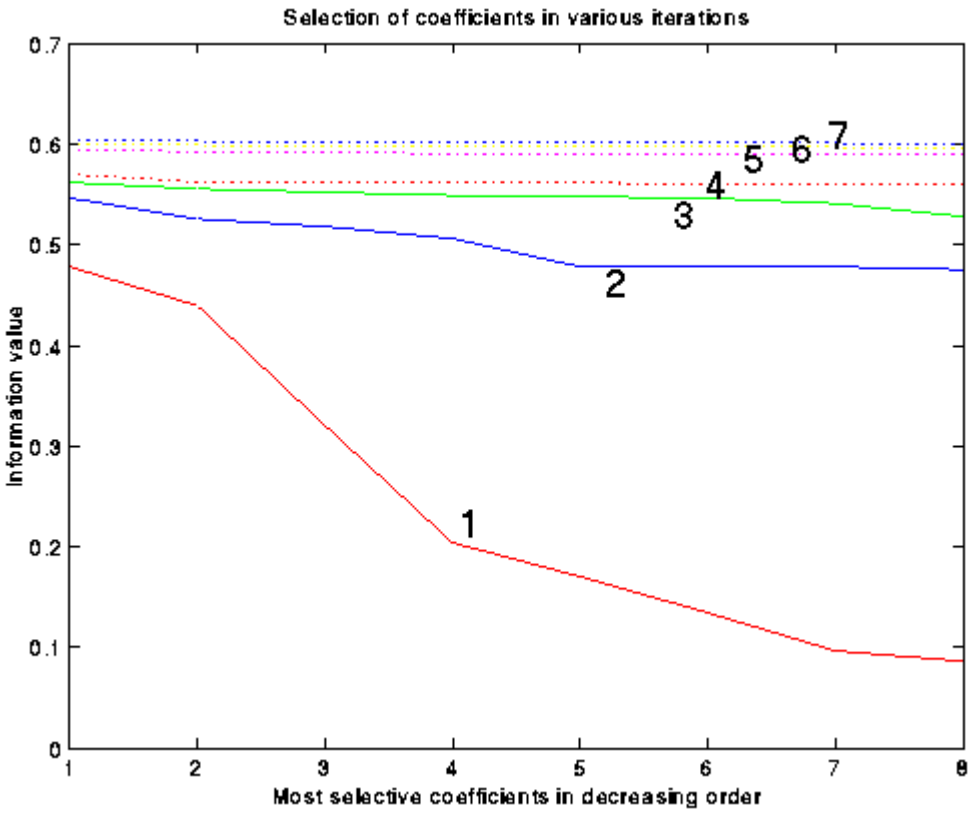


Fig. 4.8: Entropy measure for best 8 coefficients after each iteration.

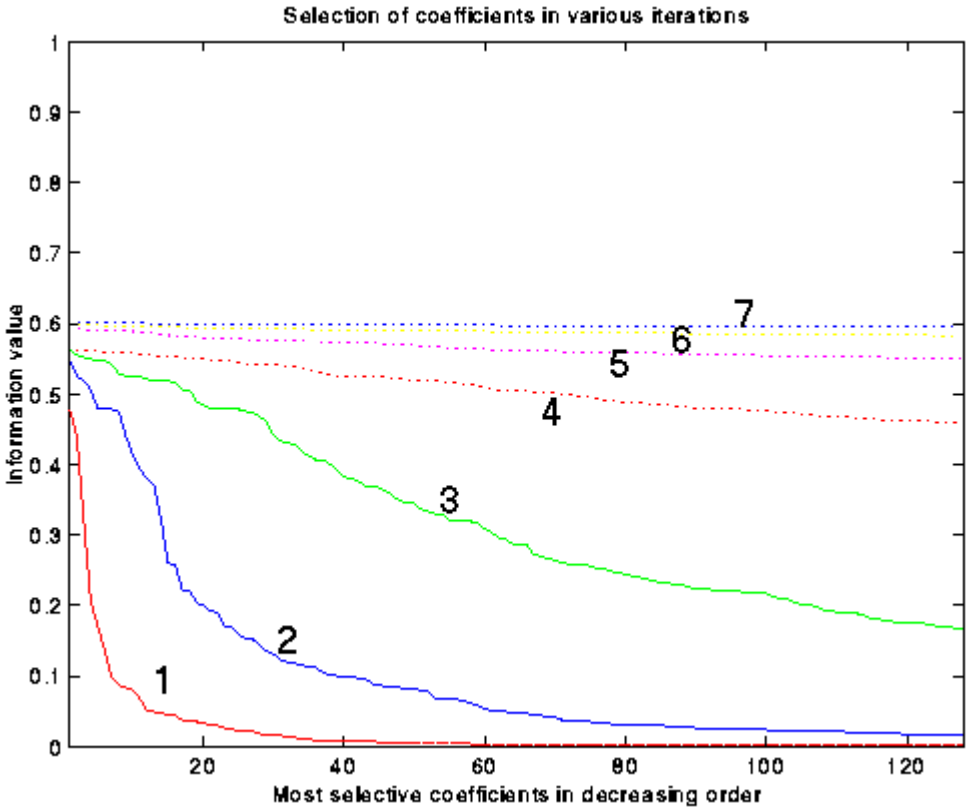
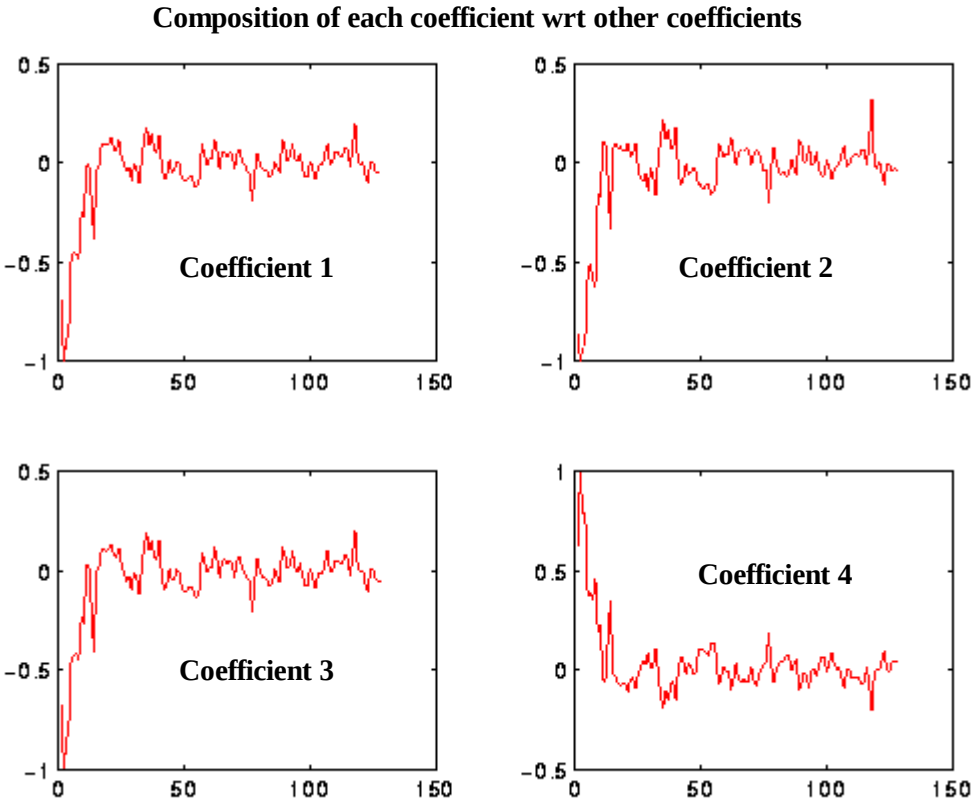


Fig. 4.9: Entropy measure for all the coefficients after each iteration.



x-axis: coefficient number, y-axis: strength of each coefficient

Fig. 4.10: Waveform interpretation of the selected coefficients.

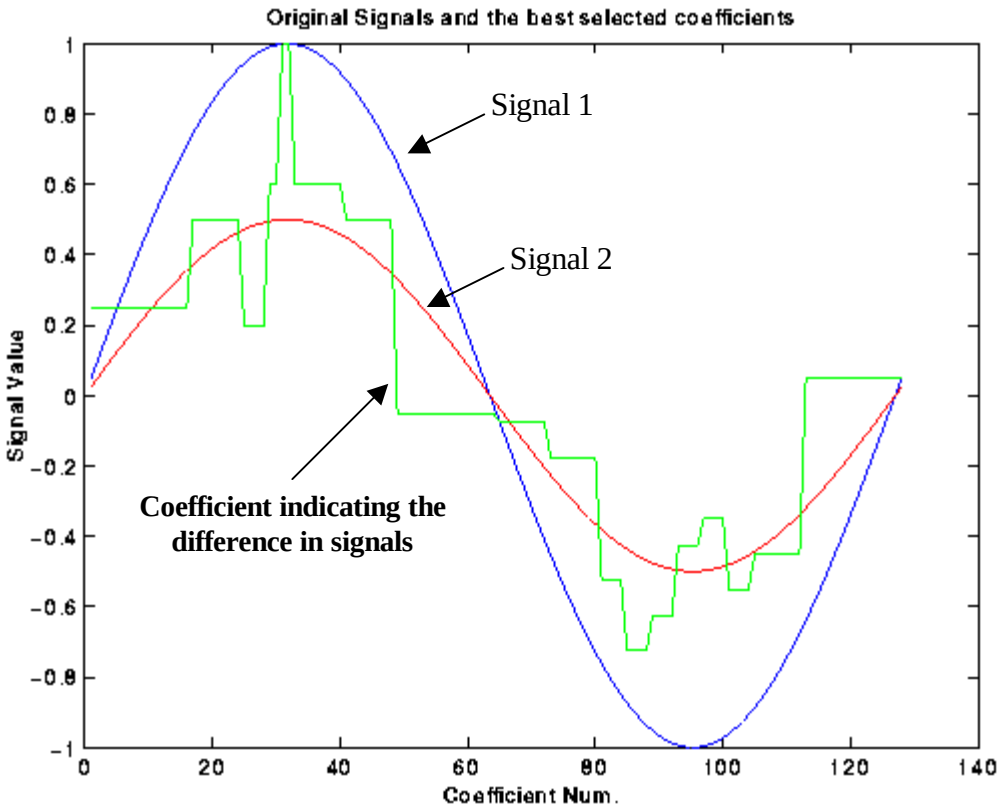


Fig. 4.11: Original signals and the evolved waveform that best separates the two signals.

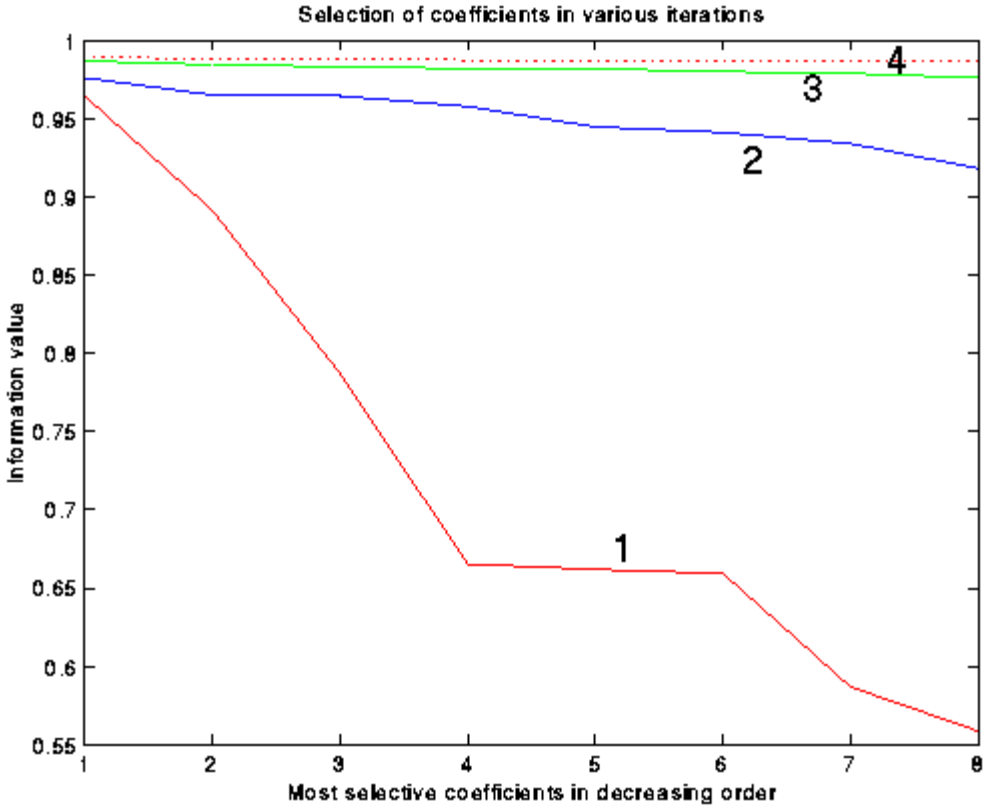


Fig. 4.12: Entropy measure for best 8 coefficients after each iteration.

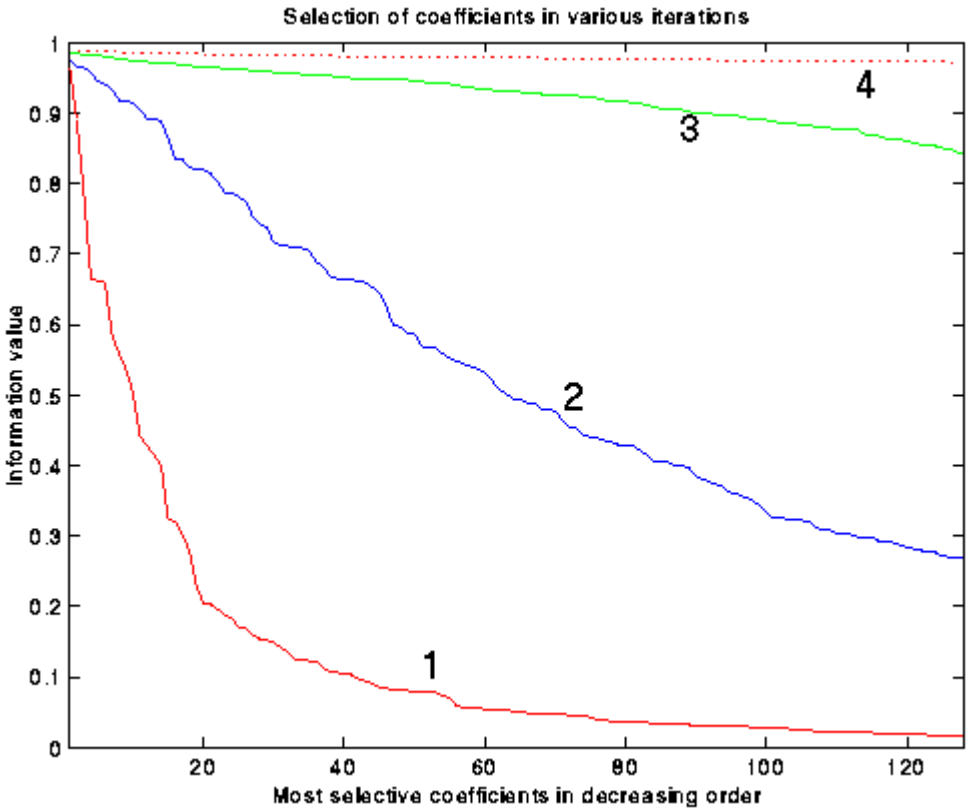


Fig. 4.13: Entropy measure for all the coefficients after each iteration.

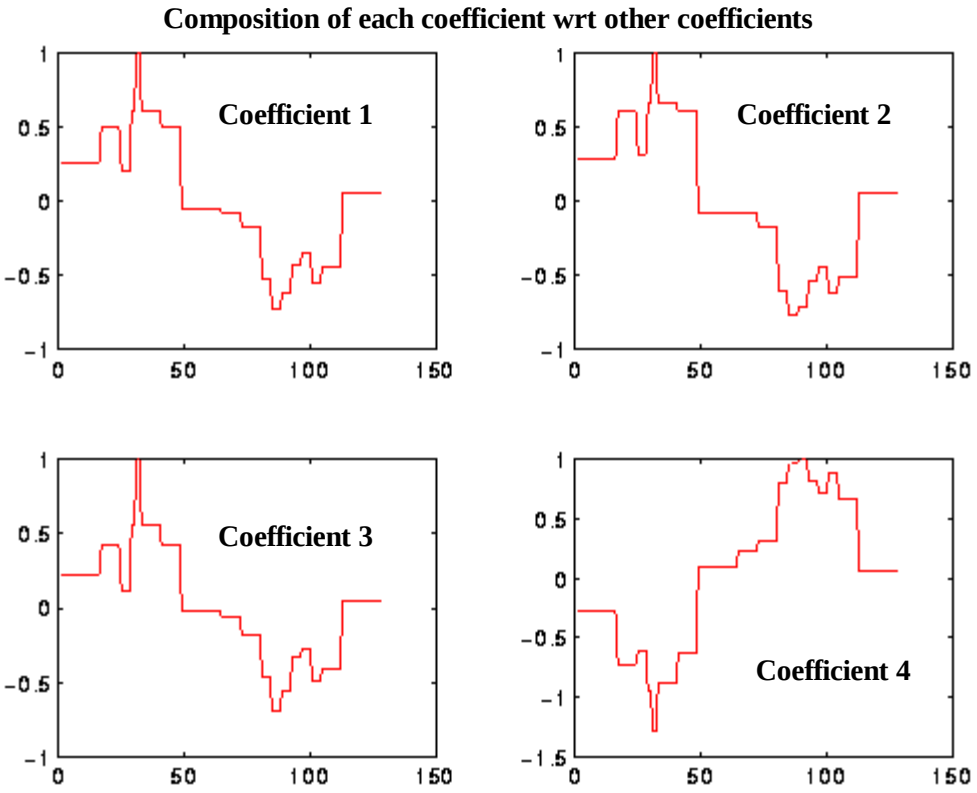


Fig. 4.14: Waveform interpretation of the selected coefficients.

Chapter 5

FPGA realization of Haar Wavelet

Field-Programmable Gate Arrays (FPGAs) are flexible and reusable high-density circuits that can be (re)configured by the designer, enabling the VLSI design/validation/simulation cycle to be performed more quickly and cheaply. FPGAs can implement thousands of logic gates in a single IC and it can be programmed by users at their site in a few seconds or less depending on the device type used. The design risk is low and the development time is short. These advantages have made FPGAs very popular for prototype development, custom computing, digital signal processing, and logic emulation [9].

XC4000 Series devices are implemented with a regular, flexible, programmable architecture of Configurable Logic Blocks (CLBs), interconnected by a powerful hierarchy of versatile routing resources, and surrounded by a perimeter of programmable Input/Output Blocks (IOBs). They have generous routing resources to accommodate the most complex interconnect patterns. Because Xilinx FPGAs can be reprogrammed an unlimited number of times, they can be used in innovative designs where hardware is changed dynamically, or where hardware must be adapted to different user applications. FPGA devices can be reconfigured to change logic function while resident in the system. This capability gives the system designer a new degree of freedom not available with any other type of logic. Hardware can be changed as easily as software. Design updates or

modifications are easy, and can be made to products already in the field. An FPGA can even be reconfigured dynamically to perform different functions at different times. Reconfigurable logic can be used to implement system self-diagnostics, create systems capable of being reconfigured for different environments or operations, or implement multi-purpose hardware for a given application. As an added benefit, using reconfigurable FPGA devices simplifies hardware design and debugging and shortens product time-to-market.

5.1 Basic Functional Blocks: The basic architecture of the XC4000 series is shown in Figure 5.1. The Xilinx XC4000 series contain three major building blocks [12]:

- Configurable Logic Block (CLB)
- Input/Output Block (IOB)
- Programmable Interconnect

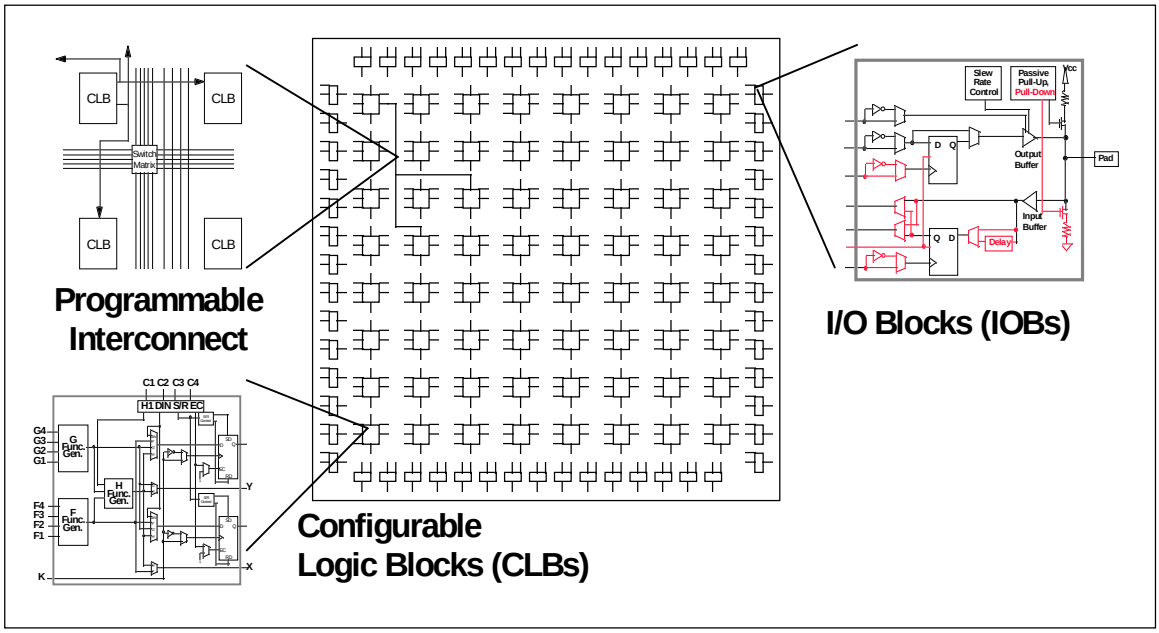


Fig. 5.1: General architecture of the Xilinx XC4000 FPGA.

5.1.1 Configurable Logic Block: This is the main, logic building block in a FPGA. It consists of two 4-input function generators (F and G) and one 3-input function generator (H). Together they are capable of implementing a 9-variable function. Each CLB consist of two storage elements that can either store the output of function generators or can be independently configured as flip-flops or latches. Figure 5.2 shows the block diagram of a CLB [12].

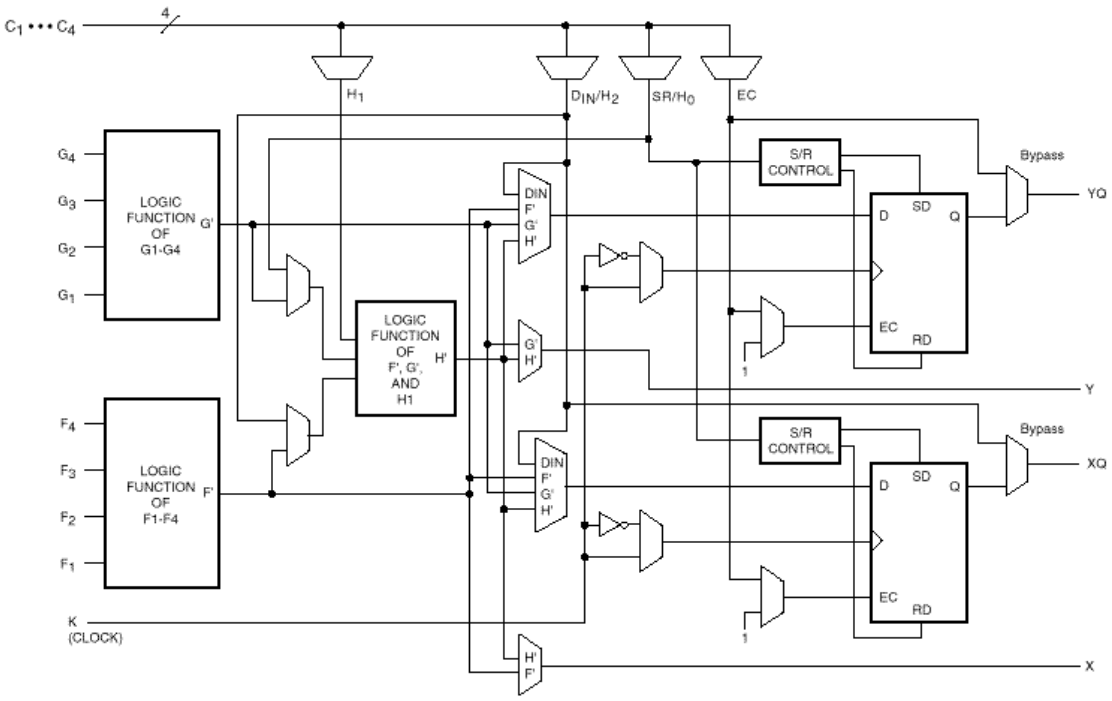


Fig. 5.2: Block diagram of a Configurable Logic Block (CLB).

5.1.2 Input/Output Block: User configurable input/output blocks (IOBs) provide the interface between external package pins and the internal logic. Each IOB controls one

package pin and can be configured for input, output, or bi-directional signals. Figure 5.3 shows the block diagram of an IOB [12].

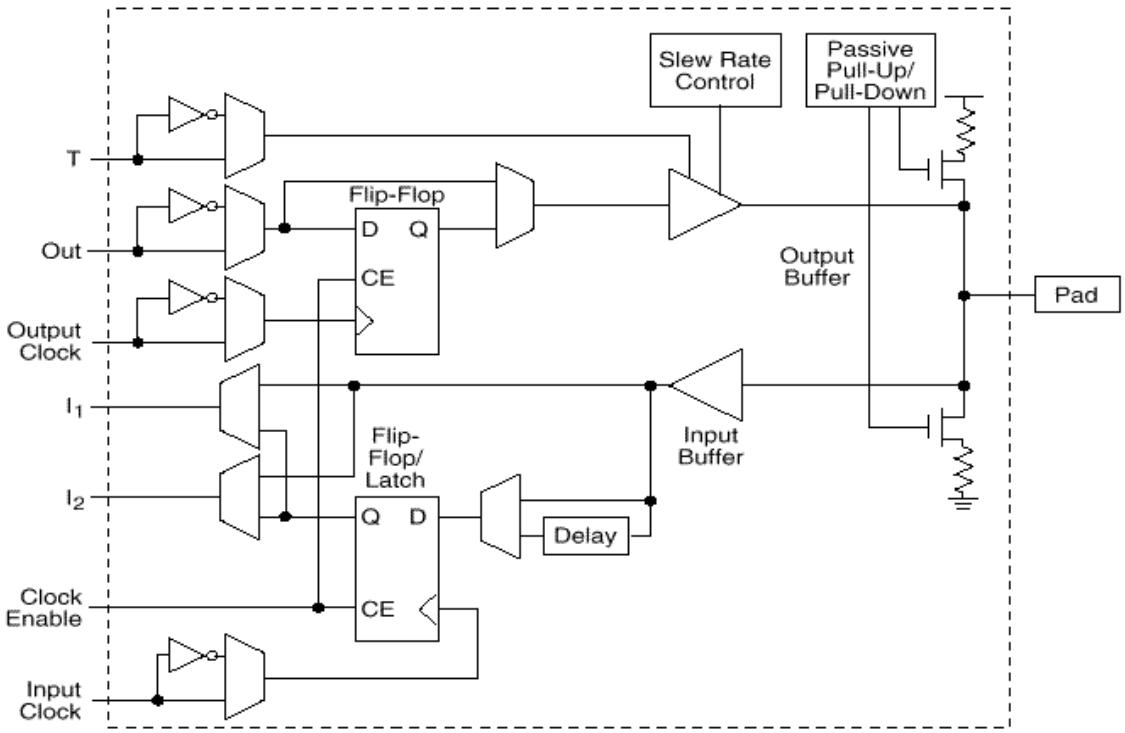


Fig. 5.3: Block diagram of an Input/Output Block (IOB).

5.1.3 Programmable Interconnect: A high level diagram of the routing resources associated with one CLB is shown in Figure 5.4. Five interconnect types are distinguished by the relative length of their segments: single-length lines, double-length lines, quad lines, octal lines and long lines. In the XC4000 FPGAs, direct connects allow fast data flow between adjacent CLBs, and between IOBs and CLBs. The horizontal and vertical single- and double-length lines intersect at a box called a programmable switch

matrix (PSM). Each switch matrix consists of programmable pass transistors used to establish connections between the lines (Figure 5.5).

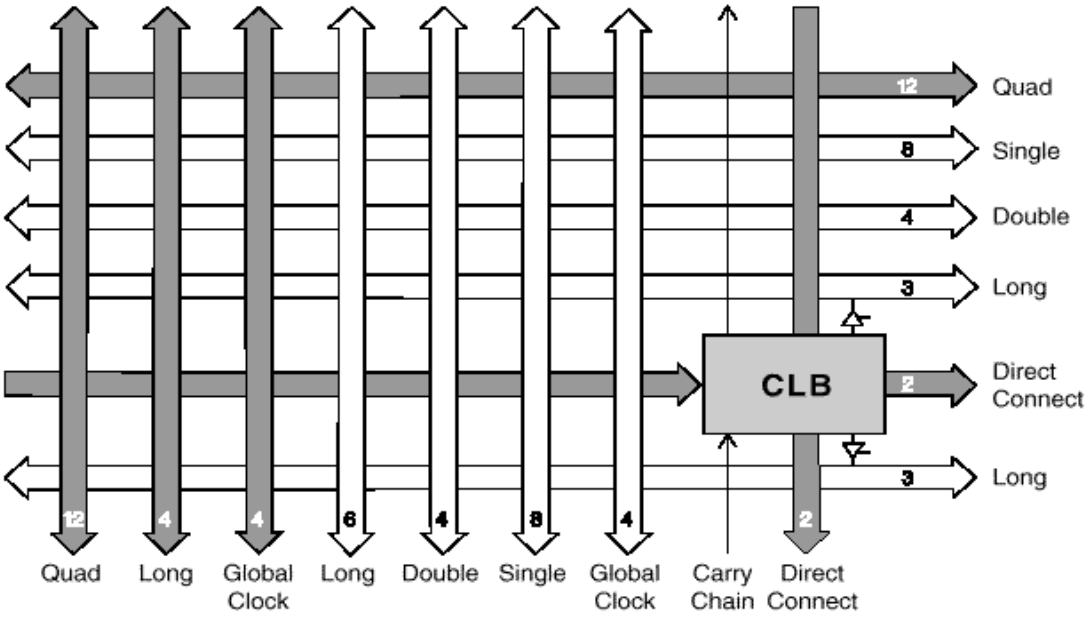


Fig. 5.4: High level routing diagram of the XC4000 FPGA

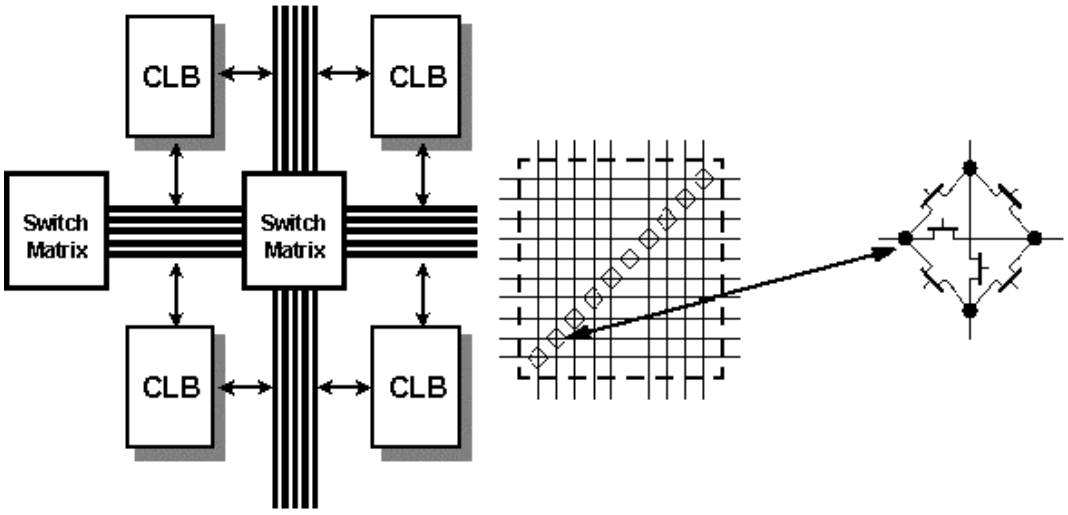


Fig. 5.5: Programmable switch matrix

5.2 FPGA Implementation of Haar Wavelet: The speed of a neural network can be significantly increased if data is fed in parallel. For parallel implementation of data, FPGAs can serve as an ideal source. Hence, a simple parallel architecture was thought of. Figure 5.6 shows the block diagram for this architecture for an 8-coefficient Haar wavelet. The registers, adders and difference blocks are triggered at the rising edge of the clock. At the first clock edge the data is loaded and on the 2nd edge gets out of the registers to the adder and the difference blocks. At the 3rd clock edge the adder and the difference blocks perform their respective operations which are described in equations as:

$$H_i = \frac{(a_{2*i} + a_{2*i+1})}{2} \qquad \text{where } i = 0 \ldots (N/2) - 1 \quad \text{(average components)}$$

$$H_{(N/2)+i} = (a_{2*i} - a_{2*i+1}) \qquad \text{where } i = 0 \ldots (N/2) - 1 \quad \text{(difference components)}$$

The 4th clock edge performs the Haar transform on the data out of the stage 1 of the Haar transform. This data output is known as stage 2 data and is fed to stage 3, which at the 5th clock edge produces the final transformed data. The top level description of the architecture was done in structural flow in VHDL and each block of the register, adder and difference block were mapped to their respective entities. The code is shown in Appendix B at page B-1. As is seen, the register block output the input data on clock edge, the adddiv block computes the average of the two input data and outputs it on the clock edge. The difference block computes the difference between the two input data and outputs the difference on the clock edge. The haar entity then stitches together all the blocks in figure 5.6. Xilinx foundation tools and Synopsis FPGA Express were used to simulate and synthesize the code and map it to the FPGA. The simulation results are shown in figure 5.7 and Fig 5.8. Fig 5.7 shows a set of input data and output of each stage

at the rising edge of clock. Figure 5.8 shows the parallel nature of the architecture. The data is continuously fed in parallel and each block performs its operation on the rising edge and the data appears at the output in parallel. Note that at any clock edge each block is processing data that was processed by the previous stage at the pervious clock edge, hence the parallel architecture.

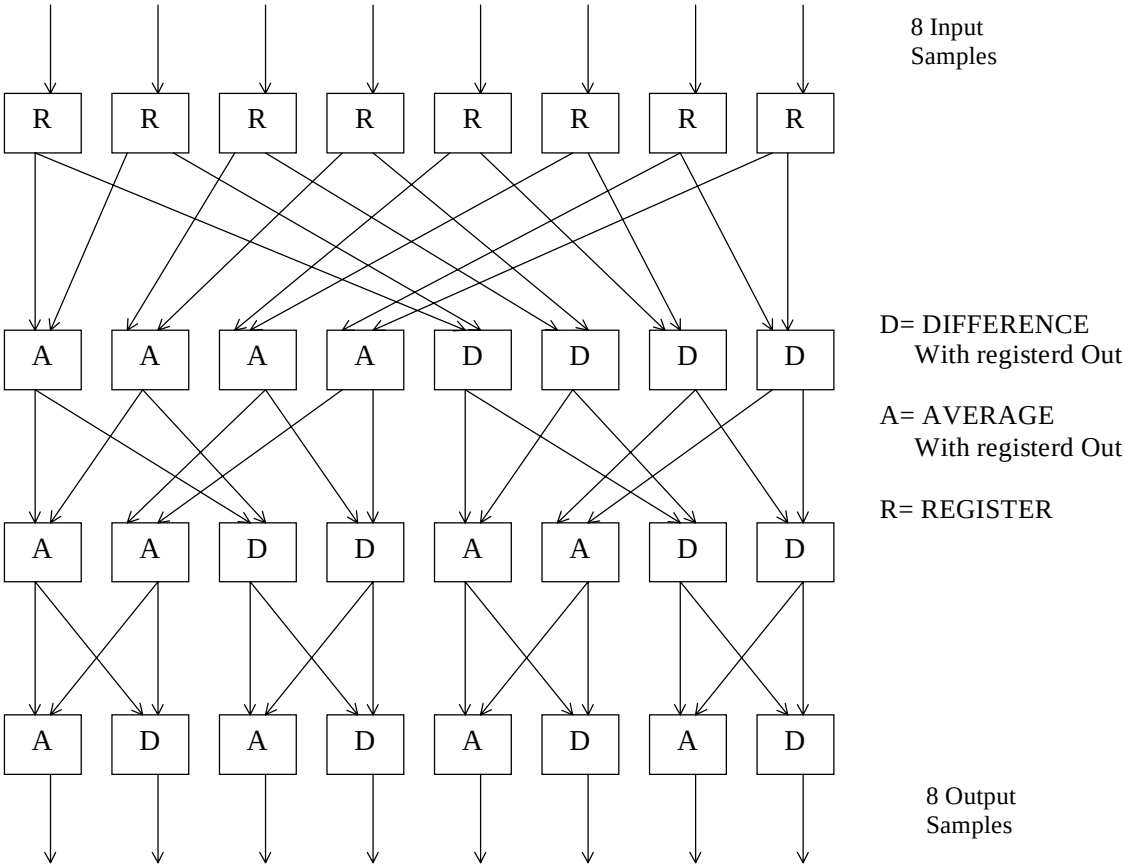


Fig. 5.6: Parallel architecture of Haar Wavelet (8-inputs).

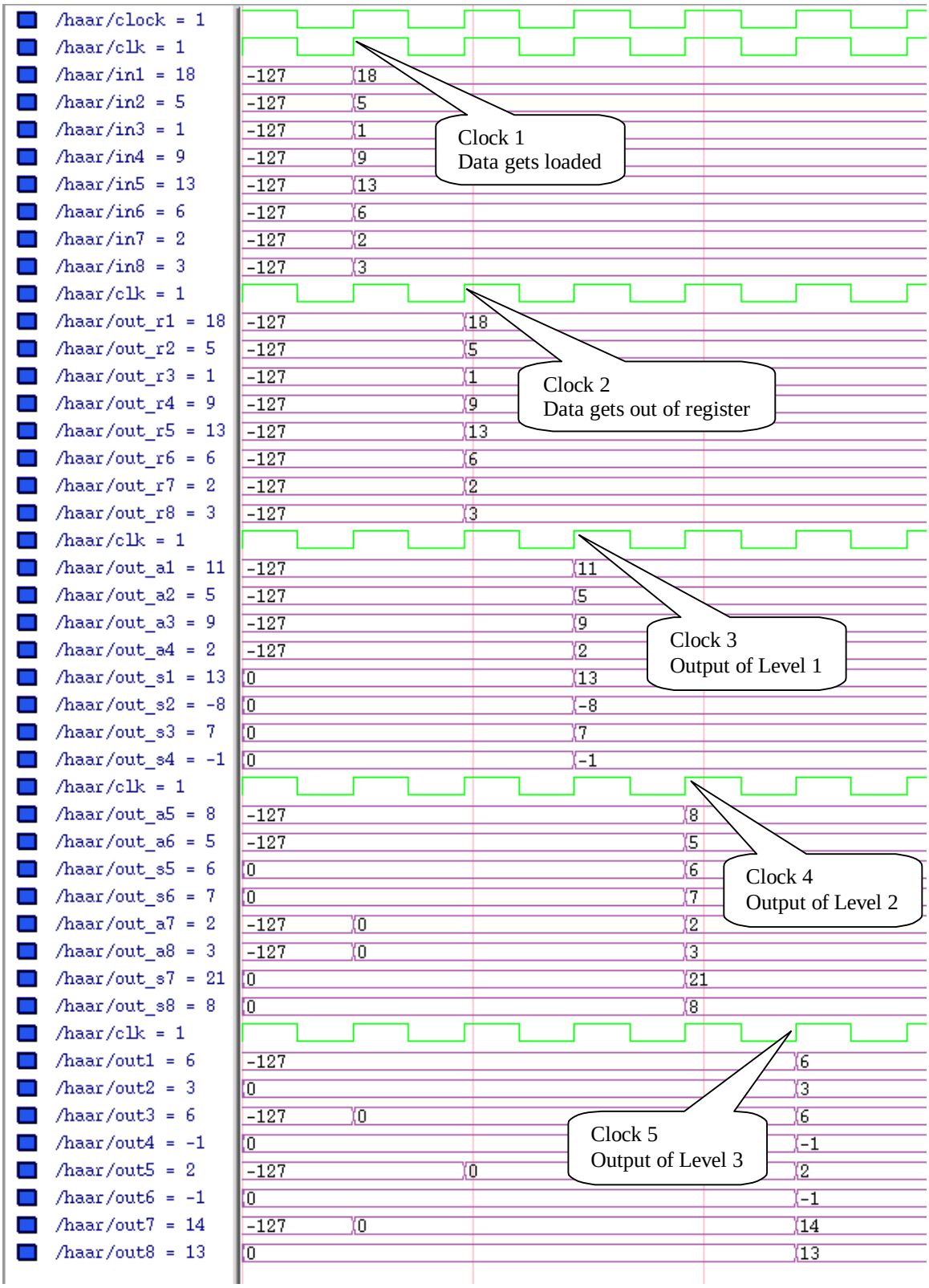


Fig. 5.7: Simulation Result showing data at various stages at each clock edge.

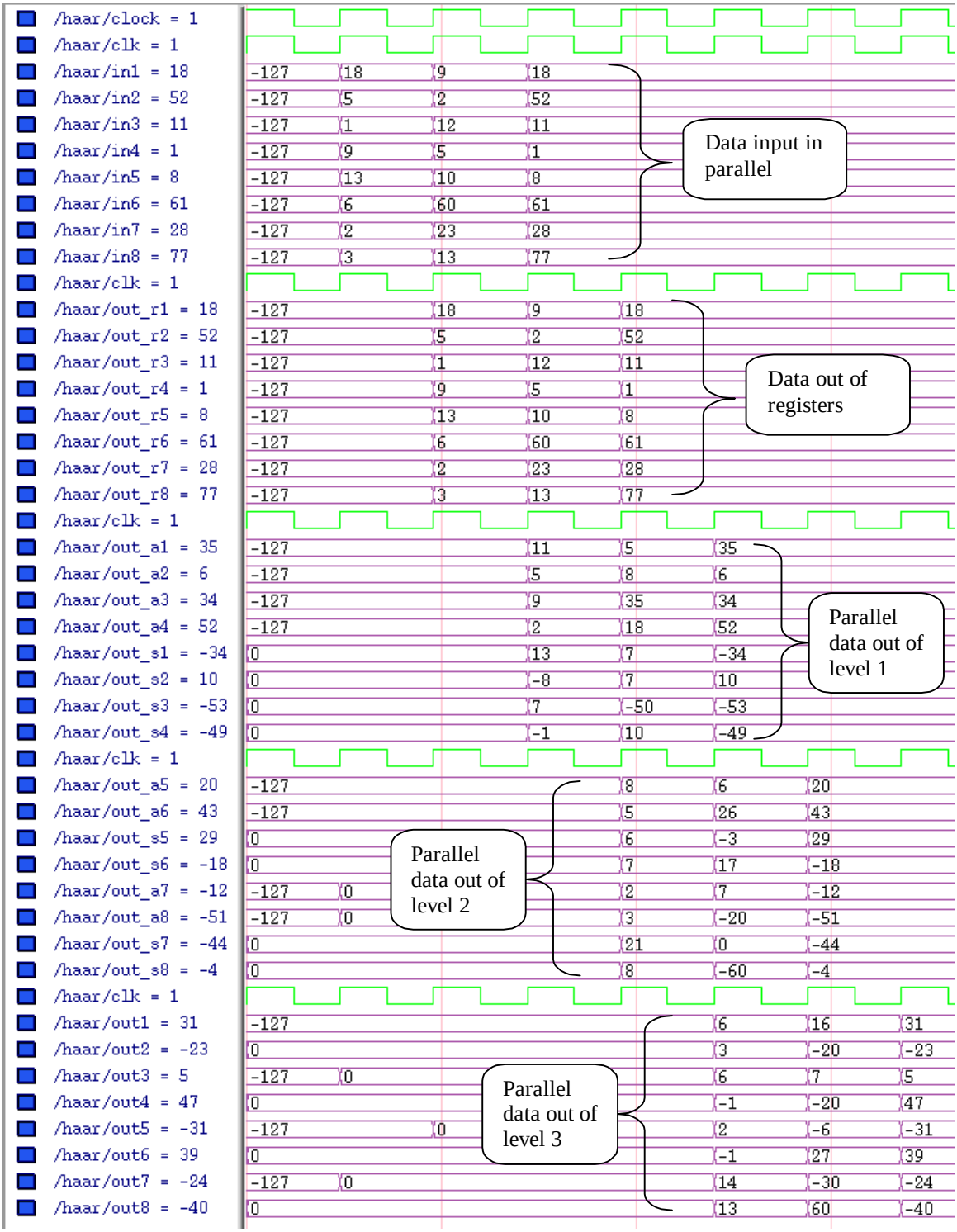


Fig. 5.8: Simulation Result showing the parallel nature of the architecture.

This thesis was focused on design methodology for automatic learning of the optimum transformation of the input space based on Haar wavelet structure. Although the described architecture implements only the Haar wavelet, it can be used as a core subsystem in a preprocessing system which iteratively uses Harr wavelet, selects corresponding coefficients and reapply the Haar wavelet transform. Full realization of the presented concept in FPGA architecture was a subject of another MS thesis [10].

5.3 **Facts and figures of the implementation:** Figure 5.7 shows the layout on the XC4010XLPQ160 FPGA and Table 5.1 shows the facts about various resources in the FPGA used.

Number of CLBs	120/400 (30%)
Number of bonded IOBs	129/129 (100%)
Number of global buffers	1/12 (8%)
Total equivalent gate count	3948
Minimum period	25.405 ns
Maximum frequency	39.362 MHz
Maximum net delay	10.365 ns
Average Connection Delay	3.494 ns
Average Connection Delay on critical nets	0.000 ns
Average Clock Skew	0.248 ns
The Maximum Pin Delay	10.365 ns
Average Connection Delay on the 10 Worst Nets	9.184 ns

Table 5.1: Facts about the resources of the FPGA used.

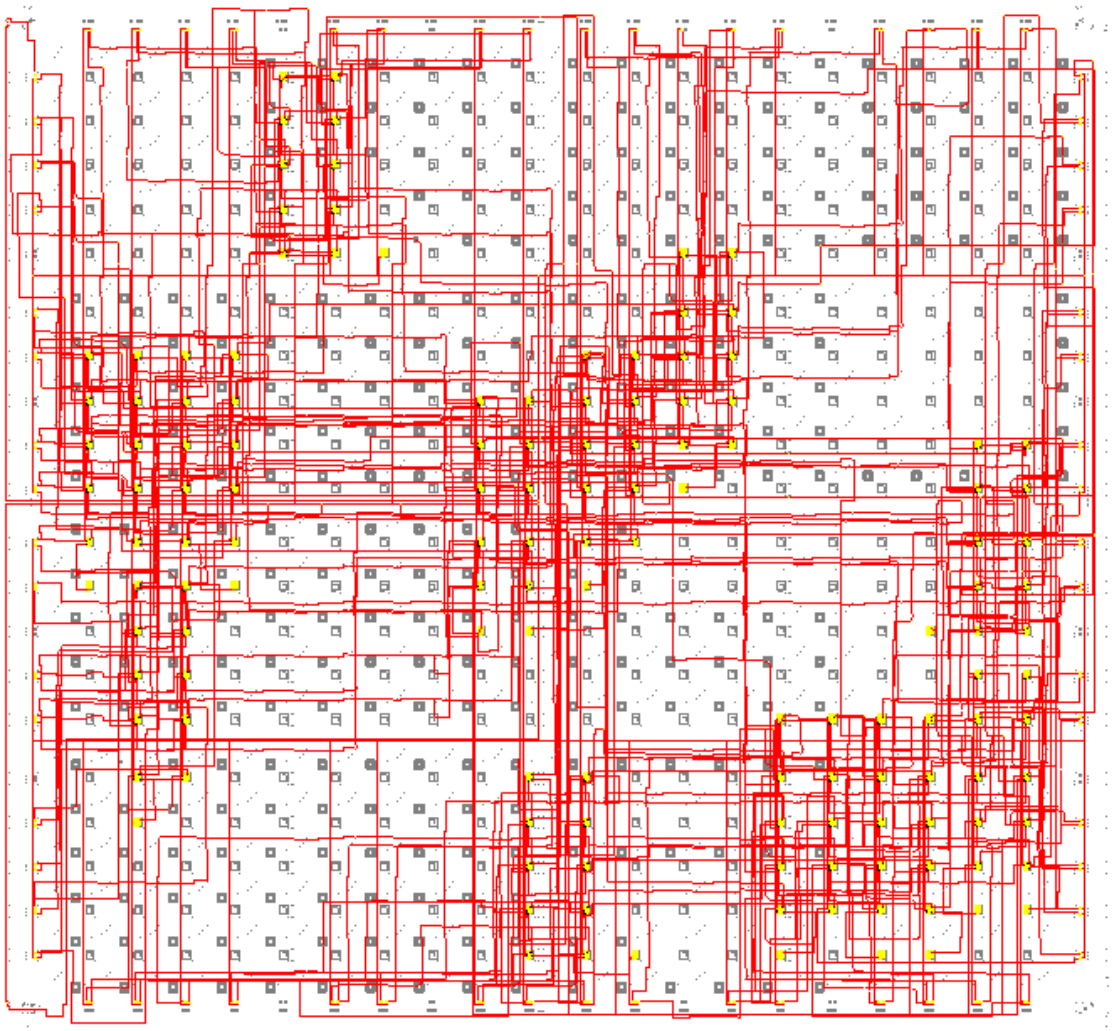


Fig. 5.9: Layout of the Haar Wavelet Parallel Architecture on the XC4010XLPQ160 FPGA.

5.4 **Advantages and limitations of the parallel architecture:** This architecture takes advantage of the fact that FPGAs can process parallel data very fast. Because of this, large amount of data can be processed at the same time and this increases the speed of computation. In addition, the architecture is very simple so it could be easily extended for 128 inputs. However, the number of input/output pins is limited on a given chip. This puts a limitation on the number of inputs the architecture can have.

Chapter 6

Summary and Future Work

Pattern recognition techniques are often an important component of intelligent systems. With the increase in size and complexity of the feature selection undertaken in pattern recognition and other areas, people pay more attention to the application of evolutionary and adaptive learning techniques in this field. It was demonstrated in this work that by selecting features based on evolutionary use of Haar wavelets, new features with better classification properties could be obtained. The convergence is very fast. The quality of the obtained transformation is measured using the entropy based information index. This index was developed for data sets with estimated probability density functions of different classes. Any mutual dependence of extracted features is automatically accounted for by performing Monte Carlo integration in 2 and 3 dimensional space. The confidence interval of the predicted performance is related to standard deviation of the information index and depends on the information level and the number of training points used.

Future work will include extension of the proposed information measure to k-class problem, training and testing using some real data. In the FPGA implementation, 8-signal approach could easily be extended to 128 signals. To make a real time reconfigurable architecture the XC6200 can be the best choice. Besides the parallel architecture discussed in this thesis, other architectures could be explored and implemented.

References

[1] Tzay Y. Young, King-Sun Fu, *Handbook of Pattern Recognition and Image Processing*, Academic Press Inc., 1986.

[2] William S. Meisel, *Computer Oriented Approaches to Pattern Recognition*, Academic Press Inc., 1972.

[3] Robert Schalkoff, *Pattern Recognition: Statistical, Structural and Neural Approaches, John Wiley & Sons*, Inc. 1992.

[4] Dr. Janusz Starzyk, *Evolutionary Feature Extraction for SAR Air to Ground Moving Target Recognition – Statistical Approach*, August 1998.

[5] Keinosuke Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, 1972.

[6] Xiaohui Liu Cohen, Michael Berthold, *Advances in Intelligent Data Analysis, Reasoning about Data*, Second International Symposium, IDA-97, London, UK, August, 1997

[7] Chi-hau Chen, *Statistical Pattern Recognition*, Hayden Book Company, Inc. 1978.

[8] Graps Amara, *An Introduction to Wavelets, IEEE Computational Science and Engineering*, Vol. 2, No. 2, Summer 1995.

[9] Xilinx, *The Programmable Logic Data Book*, Xilinx, 1998.

[10] Abdulqadir Al-aqeeli, *Reconfigurable Wavelet-Based Architecture for Pattern Recognition Applications using A Field Programmable Gate Array*, MS Thesis,

School of Electrical Engineering and Computer Science, Ohio University, Athens, Ohio, 1998.

[11] XC4000 Data Sheet, at http://www.xilinx.com.

[12] Presentation Materials, at http://www.xilinx.com/univ.htm.

Appendix

Appendix A

A.1 Program: haarintp.m

```
%plotting the interpretation of haar transform
numCoeff=8;
% determine levels????
levls=round(log10(numCoeff)/log10(2));
x=1:numCoeff;

rows=numCoeff;
coeffMatrix=diag(ones(1,numCoeff));
coeffD=coeffMatrix;
for i=1:levls
  coeffC=[];
  begn=1;
  while begn<numCoeff
    oddr=(begn:2:begn+rows-1);
    evnr=(begn+1:2:begn+rows-1);
    coeffA=[];
    coeffB=[];
    for j=1:length(oddr)
      coeffA(j,:)=( coeffMatrix(oddr(j),:) + coeffMatrix(evnr(j),:) )/2;
      coeffB(j,:)=coeffMatrix(oddr(j),:) - coeffMatrix(evnr(j),:);
    end;
    coeffC=[coeffC;coeffA;coeffB];
    begn=begn+rows;
  end; % while
  coeffMatrix=coeffC;
  coeffD=[coeffD;coeffC];
  rows=round(rows/2);
end; %for i

coeffMatrix=coeffD;

%plotting the interpretation of haar transform
range=[0,9,-1.5,1.5];
subplot(10,4,1);
text(0,1,'Variation of one coefficient with other coefficients');
axis('off');
```

```
for i=1:numCoeff
  subplot(10,4,4*(i-1)+4+1);
  bar(x,coeffMatrix(numCoeff*0+i,:),'r');axis(range);
  subplot(10,4,4*(i-1)+4+2);
  bar(x,coeffMatrix(numCoeff*1+i,:),'m');axis(range);
  subplot(10,4,4*(i-1)+4+3);
  bar(x,coeffMatrix(numCoeff*2+i,:),'g');axis(range);
  subplot(10,4,4*(i-1)+4+4);
  bar(x,coeffMatrix(numCoeff*3+i,:),'b');axis(range);
end;

subplot(10,4,4*(numCoeff+1)+1);
text(0,0,'Original');axis('off');

subplot(10,4,4*(numCoeff+1)+2);
text(0,0,'After 1st loop');axis('off');

subplot(10,4,4*(numCoeff+1)+3);
text(0,0,'After 2nd loop');axis('off');

subplot(10,4,4*(numCoeff+1)+4);
text(0,0,'After 3rd loop');axis('off');
```

A.2 Program: onedim.m

```
clear all; % clears all variables and functions from the workspace
pack; % Consolidate workspace memory


constant;

%================================
%     Add noise to each signal
%================================
for i=1:50
  Matrix(:,i)=signal1+sig1noiselevel*(2*rand(numCoeff,1)-1);
end;
for i=51:100
  Matrix(:,i)=signal2+sig2noiselevel*(2*rand(numCoeff,1)-1);
end;

x=1:numCoeff;
```

```
% plot original signal.
if plotOriginal==1
  figure;
  plot(x,signal1,'r');
  hold on;
  plot(x,signal2,'b');
  xlabel('Signal coordinate');
  ylabel('Signal value');
  title('Original signals');
end;

% plot noisy signal.
if plotNoisy==1
  figure;
  plot(x,Matrix(:,1:50),'r');
  hold on;
  plot(x,Matrix(:,51:100),'b');
  xlabel('Signal coordinate');
  ylabel('Signal value');
  title('Two signals with a random noise');
end;

figure;

%==================================
% Main loop starts from here
%==================================

% start the main loop and continue till the desired accuracy is achieved.
while abs(currentInfo - previousInfo) > absinfoincr

  IterationNumber=IterationNumber+1;
  fprintf(fid,'Iteration Number = %i\n',IterationNumber);

  % check the iteration being done. if more than 8 then reset it to 1.
  % the reason is that we have only 8 ways of plotting the wave form.
  strindex=mod(IterationNumber,8);
  if strindex==0
    strindex=8;
  end;

%==================================
%  Perform Haar Transform
%==================================
```

```
rows=numCoeff;
D=Matrix;
for i=1:levls
  C=[];
  begn=1;
  while begn<numCoeff
    oddr=(begn:2:begn+rows-1);
    evnr=(begn+1:2:begn+rows-1);
    % this normalization is used to balance Walsh coefficients
    % of a raw signal
    %A=(Matrix(oddr,:)+Matrix(evnr,:))/2;
    A=(Matrix(oddr,:)+Matrix(evnr,:))/2;
    % subtract pairwise
    B=Matrix(oddr,:)-Matrix(evnr,:);
    C=[C;A;B];
    begn=begn+rows;
  end; % while
  Matrix=C;
  D=[D;C];
  rows=round(rows/2);
end; %for i

Matrix=D;
[rowsMatrix colsMatrix] = size(Matrix);

% make other matrices to easy out some other computations.
tMatrix = Matrix';
sig1Matrix = Matrix(:,1:50);
sig2Matrix = Matrix(:,51:100);

% determine the minimum and maximum value of the signal.
rmin = min(tMatrix);
rmax = max(tMatrix);

% determine the standard deviation of each signal.
D1=std (sig1Matrix');
D2=std (sig2Matrix');

% determine mean of each signal.
E1=mean(sig1Matrix');
E2=mean(sig2Matrix');

% set probability of each signal to be 1.
```

```
  P1=1;
  P2=1;
  spread=5;
  resolve=200;
  numSteps = 300;
% calculate the information

  info1=infopdfV(P1,P2,E1,E2,D1,D2,spread,numSteps);

% sort the information matrix to extract the best "numCoeff" number of elements.
  [info1 index] = sort(info1);
  orderedinf=fliplr(info1);
  index=fliplr(index);

  plotarray1D = [plotarray1D;orderedinf(1:numCoeff)];
  infoarray=[infoarray;orderedinf(1:numCoeff)];

  if plotInfo1D==1
    axis([1,max(x),0,1]);
    plot(x,orderedinf(1:numCoeff),stringmat(strindex,:));
    xlabel('Most selective coefficients in decreasing order');
    ylabel('Information value');
    title('Selection of coefficients in various iterations');
    hold on;
  end;

% update the values of the information matrix
  previousInfo=currentInfo;
  currentInfo=orderedinf(1);

% reset the Matrix before going into the other loop.
  rownum = index(1:numCoeff);
  Matrix=Matrix(rownum,:);
  selectedcoeffMatrix=[selectedcoeffMatrix;coeffMatrix(rownum,:)];
  fprintf(fid,'Iteration %i complete. Hit any key...\n',IterationNumber);
  keyboard;
end; %while iterations

printVec(infoarray,'infoarray',fidInfo);

%==================================
%  Plot the Haar Interpretation
%==================================
```

```
if plotHarr==1
  M=selectedcoeffMatrix(1:numCoeff,:);
  for i=2:IterationNumber
    M=selectedcoeffMatrix(numCoeff*(i-1)+1:numCoeff*i,:)*M;
  end;

  printVec(M,'Haar Interpretaion',fidHarr);
  %plotting the interpretation of haar transform as bar graph
  figure;
  for i=1:4
    subplot(2,2,i);
    bar(x,M(i,:),'r');
  end;

  %plotting the interpretation of haar transform as signals
  figure;
  plot(signal1,'b')
  hold on
  plot(signal2,'r')
  e=M(1,:);
  en=e/norm(e)*((norm(signal1)+norm(signal2))/2);
  plot(en(1,:),'g')

end; % if plotHarr

if fidHarr ~= 1
  fclose(fidHarr);
  fclose(fidInfo);
end;
```

A.3 Program: constant.m

```
numCoeff=128;
i=1:numCoeff;
signal1=sin(2*pi/(numCoeff-1)*i);

signal2=0.75*ones(1,20);
i=21:numCoeff;
signal2=[signal2 sin(2*pi/(numCoeff-1)*i)];
clear('i');
% setup the way each plot of information matrix is to be plotted.
stringmat=['r-';'b-';'g-';'r:';'m:';'y:';'b:';'g:';'r:';'m-'];
```

```
% noise level to be added to the signal.
sig1noiselevel=1;
sig2noiselevel=1;

% Some epsilons
epsil=0.01;
epsil2=-0.02;

% choice of which plots to plot.
plotOriginal=1;
plotNoisy=1;
plotInfo1D=1;
plotHarr=1;

% initialise the variables
absinfoincr=.005;
previousInfo=0;
currentInfo=1;
IterationNumber=0;
plotarray1D=[];
infoarray=[];

%==================================
% Some file identifier definitions to indicate where to dump data
%==================================
fid=1; % file identifier, used for printing results on the screen.
fidInfo  = fopen('c:\users\aman\matlab\work\plotinfo.txt','w');
fidHarr  = fopen('c:\users\aman\matlab\work\harrinfo.txt','w');

%==================================
%  Make Haar Coefficient Lookup Table
%==================================

levls=round(log10(numCoeff)/log10(2));   % determine levels ????
rows=numCoeff;
coeffMatrix=diag(ones(1,numCoeff));
coeffD=coeffMatrix;
for i=1:levls
  coeffC=[];
  begn=1;
  while begn<numCoeff
    oddr=(begn:2:begn+rows-1);
    evnr=(begn+1:2:begn+rows-1);
```

```
    coeffA=[];
    coeffB=[];
    for j=1:length(oddr)
      coeffA(j,:)=( coeffMatrix(oddr(j),:) + coeffMatrix(evnr(j),:) )/2;
      coeffB(j,:)=coeffMatrix(oddr(j),:) - coeffMatrix(evnr(j),:);
    end;
    coeffC=[coeffC;coeffA;coeffB];
    begn=begn+rows;
  end; % while
  coeffMatrix=coeffC;
  coeffD=[coeffD;coeffC];
  rows=round(rows/2);
end; %for i
coeffMatrix=coeffD;
selectedcoeffMatrix=[];
```

A.4 Program: infopdfV.m

```
function info=infopdfV(P1,P2,E1,E2,D1,D2,spread,numSteps)



%  This function finds the entropy measure between two pdf functions
%  The sum of probabilities is constant and each probability
%  obtained from its pdf is known a priori
%  each random variable is described by normal pdf
%  pdf=P/(sqrt(2*pi)*D)*exp(-0.5*(x-E)^2./D^2)
%  cdf specifies the normal cumulative distribution function


plotPdf1=0;
plotPdf2=0;
entries=length(E1);
case1=[];
case2=[];
case3=[];
for i=1:entries
  if D1(i)~=0 | D2(i)~=0
    case1=[case1,i];
  end;
  if D1(i)==0 & D2(i)==0 & E1(i)==E2(i)
    case2=[case2,i];
  end;
```

```
  if D1(i)==0 & D2(i)==0 & E1(i)~=E2(i)
    case3=[case3,i];
  end;
end;

xmin=min(E1(case1)-spread*D1(case1) , E2(case1)-spread*D2(case1));
xmax=max(E1(case1)+spread*D1(case1) , E2(case1)+spread*D2(case1));

x=[];
step=(xmax-xmin)/numSteps;
for i=1:length(case1)
  temp=[];
  tempx=xmin(i):step(i):xmax(i);
  if length(tempx) > numSteps
    x(i,:)=tempx(1:numSteps);
  else
    x(i,:)=tempx;
  end;
end;

% now calculate the numerator of pdf1 and pdf2.

numD1=(x - E1(case1)'*ones(1,size(x,2))) ./ (D1(case1)'*ones(1,size(x,2)));
numD2=(x - E2(case1)'*ones(1,size(x,2))) ./ (D2(case1)'*ones(1,size(x,2)));

% calculate pdf1 and pdf2

pdf1=P1*exp(-0.5*numD1.*numD1)./(sqrt(2*pi)*D1(case1)'*ones(1,size(x,2)));
pdf2=P2*exp(-0.5*numD2.*numD2)./(sqrt(2*pi)*D2(case1)'*ones(1,size(x,2)));

if plotPdf1==1
  for i=1:size(pdf1,1)
    plot(x,pdf1(i,:));
    hold on;
  end;
end;

if plotPdf2==1
  for i=1:size(pdf2,1)
    plot(x,pdf2(i,:));
    hold on;
  end;
end;
```

```
P1=sum(pdf1').*(step);
P2=sum(pdf2').*(step);

%  find weight functions p1xx+p2xx=1

p1xx=pdf1./(pdf1+pdf2);
p2xx=1-p1xx;

%  find weighted pdfs

pdf1x=pdf1.*p1xx;
pdf2x=pdf2.*p2xx;
pdf12x=pdf1.*p2xx;
pdf21x=pdf2.*p1xx;

%  evaluate weighted probabilities

P1w=sum(pdf1x').*step;
P2w=sum(pdf2x').*step;
P12w=sum(pdf12x').*step;
P21w=sum(pdf21x').*step;

%  find the entropy measure

entr = -(P1.*log(P1)+P2.*log(P2)) + P1w.*log(P1w)+P2w.*log(P2w)+
P12w.*log(P12w)+ P21w.*log(P21w);

%  find information

P11=P1.^2./(P1+P2);
P22=P2.^2./(P1+P2);
P12=P1.*P2./(P1+P2);

maxentr=-(P1.*log(P1)+P2.*log(P2))+P11.*log(P11)+P22.*log(P22)+2*P12.*log(P12);
info(1,case1)=1-entr./maxentr;

% now we find the indeces where D1==D2==0 and E1==E2

case2=[];
for i=1:entries
  if D1(i)==0 & D2(i)==0 & E1(i)==E2(i)
    case2=[case2,i];
  end;
end;
```

```
info(case2)=zeros(1,length(case2));
info(case3)=ones(1,length(case3));
return;
```

A.5 Program: cartplot.m

```
clear;

innumPoints=25;

% any two dimensional matrix
X=[8 6;8 7;8 8;8 9;9 7;9 10;10 8;10 9;10 11;11 8;11 11; ...
   12 9;12 10; 12 12;13 9;13 11;13 13;14 12;15 13;15 14];
%X=[X 10*rand(size(X,1),1)];

%  find center
X0=mean(X);
n=size(X,2);
X=X-ones(size(X,1),1)*X0;  % X shifted to the origin
numPoints=2*ceil((innumPoints+n^2)^(1/n))+1;

%=================================
% Make Grid of Cartesian points
%=================================

% a unit grid centered at origin
delta=(2)/(numPoints-1);
Gridf=[-1:delta:1]';
temp=-1:delta:1;
for dim=2:n
  x_=[];
  for i=1:length(temp)
    x_=[x_; Gridf ones(size(Gridf,1),1)*temp(i)];
  end;
  Gridf=x_;
end;

varstd=2.3;
[Q,R]=qr(X);
R=R(1:n,:);
TX=Gridf*varstd*max(std(X));
```

```
TX=TX*R^(-1);

%  set a circle with the radius
ellradius=varstd*std(Q(1,:));


TX=TX;        % Transforming unit grid into the actual coordinates
Gridf=Gridf*varstd*max(std(X));  % Transforming unit grid into the actual coordinates

%  relative distances of the test points
TX1=TX.^2;
distances=sqrt(sum(TX1'));

select=find(distances<ellradius);
Gridell=Gridf(select,:);

%==================================
% Make an ellipse which will enclose the original points
%==================================
Z=[ellradius];
t=2*pi*(0:50)/51;
for dim=2:n
  tempZ=[];
  for rows=1:size(Z,1)
    z_=[];
    for cols=1:length(t)
      z_=[z_;Z(rows,:)];
    end;
    tempZ=[tempZ;z_ t'];
  end;
  Z=tempZ;
end;

% convert cartesian points of ellipse to spherical coordinates
Angle=Z(:,2:n);
if n==2
  cosVector=[cos(Angle) ones(size(Angle,1),1)];
else
  cosVector=[fliplr( cumprod(cos(Angle'))' ) ones(size(Angle,1),1)];
end;
sinVector=[ones(length(Angle),1) fliplr(sin(Angle))];
for i=1:size(Z,1)
  Z(i,:)=Z(i,1)*(cosVector(i,:).*sinVector(i,:));
```

```
end;
Z=Z*R;
Z=Z+ones(size(Z,1),1)*X0;

X=X+ones(size(X,1),1)*X0;
Gridf=Gridf+ones(size(Gridf,1),1)*X0;
Gridell=Gridell+ones(size(Gridell,1),1)*X0;
hold off;
if n==2
  plot(Gridf(:,1),Gridf(:,2),'g+');
  hold on;
  plot(Gridell(:,1),Gridell(:,2),'b*');
  plot(Z(:,1),Z(:,2),'r-');
  plot(X(:,1),X(:,2),'mo');
  title('2-dimensional grid');
  xlabel('x-axis')
  ylabel('y-axis')
elseif n==3
  plot3(Gridf(:,1),Gridf(:,2),Gridf(:,3),'g.');
  hold on;
  plot3(Z(:,1),Z(:,2),Z(:,3),'y.');
  figure;
  plot3(Gridell(:,1),Gridell(:,2),Gridell(:,3),'b*');
  hold on;
  plot3(X(:,1),X(:,2),X(:,3),'mo');
  title('3-dimensional grid');
  xlabel('x-axis')
  ylabel('y-axis')
  zlabel('z-axis')
end;
```

A.6 Program: sphnd.m

```
%function [Gridf, stepinfo, legntvar]=sphnd(X,fid);

%if (nargin<1)|(nargin>2)
%  error('Error using sphnd..[Gridf, stepinfo, legntvar]=sphnd(X<,fid>');
%end;

%if nargin==1
  fid=1;
%end;
```

```
X=[8 6;8 7;8 8;8 9;9 7;9 10;10 8;10 9;10 11;11 8;11 11; ...
12 9;12 10; 12 12;13 9;13 11;13 13;14 12;15 13;15 14];
X=[X 10*rand(size(X,1),1)];

%  find center
X0=mean(X);
n=size(X,2);
X=X-ones(size(X,1),1)*X0;

%plotting variables..
plotRadiusAngle=1;
plotGrid=0;
plot_confined_ellipse=1;

%  del specifies step size wrt to the standard deviation
del=0.1;
variable=intstep(del);
legntvar=length(variable);
spread=max(std(X));
variable=variable*spread;
radius=variable(2:length(variable));
deltaAlpha=((radius-variable(1:size(variable,2)-1))./radius);
deltaAlpha=pi./(round(pi./deltaAlpha));        % here deltaAlpha is in radians..

%=================================================%
% Start making the Grid
%=================================================%
Gridf=[];
stepinfo=[];
for k=1:length(radius)
  if k==1
    dr=radius(k);
  else
    dr=radius(k)-radius(k-1);
  end;
  Angle=[0 :deltaAlpha(k): pi-deltaAlpha(k)];
  prev_rows=size(Gridf,1);
  if n==2
    Gridf=[Gridf;radius(k)*ones(length(Angle),1) Angle'];
  else
    Gridf=[Gridf; Ndsph(radius(k),Angle,dr,n,radius(k))];
  end;
  curr_rows=size(Gridf,1);
```

```
  stepinfo=[stepinfo;dr*ones(curr_rows-prev_rows,1)];
end;  % for k

if plotRadiusAngle==1
  figure;
  if n==2
    polar(Gridf(:,2),Gridf(:,1),'ro');
    title('Two dimensional Polar grid');
    xlabel('angle');
    ylabel('radius');
  elseif n>=3
    plot3(Gridf(:,1),Gridf(:,3),Gridf(:,2),'ro');
    title('Three dimensional Spherical Grid in rectangular co-ordinates');
    view(-10,20);
    xlabel('radius');
    ylabel('alpha');
    zlabel('beta');
  end;
end;

% convert cartesian grid points to spherical coordinates system.
Angle=Gridf(:,2:n);
if n==2
  cosVector=[cos(Angle) ones(size(Angle,1),1)];
else
  cosVector=[fliplr( cumprod(cos(Angle'))' ) ones(size(Angle,1),1)];
end;
sinVector=[ones(length(Angle),1) fliplr(sin(Angle))];
for i=1:size(Gridf,1)
  Gridf(i,:)=Gridf(i,1)*(cosVector(i,:).*sinVector(i,:));
end;

Gridf=[-flipud(Gridf);zeros(1,n);Gridf];
stepinfo=[flipud(stepinfo);stepinfo(1);stepinfo];

if plotGrid==1
  figure;
  if n==2
    plot(Gridf(:,2),Gridf(:,1),'ro');
    title('two dimensional polar grid');
    xlabel('x-coordinate');
    ylabel('y-coordinate');
  elseif n>=3
    onlyOneSphere=0;
```

```
    if onlyOneSphere==1
      eps=.01;
      Gridpl=[];
      radius1=norm(Gridf(1,:));
      for j=1:size(Gridf,1)
        if radius1-eps<norm(Gridf(j,:)) & norm(Gridf(j,:))<radius1+eps
          Gridpl=[Gridpl; Gridf(j,:)];
        end; % if
      end; % for j
      plot3(Gridpl(:,3),Gridpl(:,2),Gridpl(:,1),'ro');
      title('three dimensional sphere');
    else
      plot3(Gridf(:,3),Gridf(:,2),Gridf(:,1),'ro');
      title('three dimensional polar grid');
    end;
    view(-10,20);
    xlabel('beta');
    ylabel('alpha');
    zlabel('radius');
  end;
end;


[Q,R]=qr(X);
R=R(1:n,:);
X=X+ones(size(X,1),1)*X0;
%  make an enclosing ellipsoid
t=2*pi*(0:50)/51;
TX=Gridf;
TX=TX*R^(-1);
%  set a circle with the radius of two std
ellradius=2.3*std(Q(1,:));
%  relative distances of the test points
TX=TX.^2;
distances=sqrt(sum(TX'))/ellradius;

%=================================================%
% Make an ellipse enclosing the original signal and selected grid points.
%=================================================%
Z=[ellradius];
for dim=2:n
  tempZ=[];
  for rows=1:size(Z,1)
    z_=[];
```

```
    for cols=1:length(t)
      z_=[z_;Z(rows,:)];
    end;
    tempZ=[tempZ;z_ t'];
  end;
  Z=tempZ;
end;

% convert cartesian ellipse points to spherical coordinates
Angle=Z(:,2:n);
if n==2
  cosVector=[cos(Angle) ones(size(Angle,1),1)];
else
  cosVector=[fliplr( cumprod(cos(Angle'))' ) ones(size(Angle,1),1)];
end;
sinVector=[ones(length(Angle),1) fliplr(sin(Angle))];
for i=1:size(Z,1)
  Z(i,:)=Z(i,1)*(cosVector(i,:).*sinVector(i,:));
end;

Z=Z*R;
Z=Z+ones(size(Z,1),1)*X0;

select=find(distances<1);
Gridell=Gridf(select,:);
stepinfo=stepinfo(select);
Gridell=Gridell+ones(size(Gridell,1),1)*X0;
if plot_confined_ellipse == 1
  currFigNO=gcf+1;
  figure(currFigNO);
  hold off;
  if n==2
    Gridf=Gridf+ones(size(Gridf,1),1)*X0;
    plot(Gridf(:,1),Gridf(:,2),'g+');
    hold on;
    plot(Gridell(:,1),Gridell(:,2),'b*');
    plot(Z(:,1),Z(:,2),'r-');
    plot(X(:,1),X(:,2),'mo');
    xlabel('Radius:->');
    ylabel('Angle:->');
    title('Ellipse showing the selected grid points only');
  else
    plot3(Z(:,1),Z(:,2),Z(:,3),'r.');
    hold on;
```

```
    plot3(X(:,1),X(:,2),X(:,3),'mo');
    xlabel('Radius:->');
    ylabel('Alpha:->');
    zlabel('Beta:->');
    title('Ellipse enclosing the original points');
    figure;
    Gridf=Gridf+ones(size(Gridf,1),1)*X0;
    plot3(Gridf(:,1),Gridf(:,2),Gridf(:,3),'b+');
    hold on;
    plot3(Gridell(:,1),Gridell(:,2),Gridell(:,3),'g*');
    xlabel('Radius:->');
    ylabel('Alpha:->');
    zlabel('Beta:->');
    title('Original grid points and selected grid points');
   end;
end; % if plot_confined_ellipse==1
%Gridf=Gridell;
%return;
```

A.7 Program: intstep.m

```
function x=intstep(del)
%
%  This function evaluates steps for integration of normal pdf
%  under the assumption that discrete integration using
%  selected points introduces equal integral values.
%  intgrl=pdf(x1)*del
%  where x2=x1+del is the next selection point, and x1=0, eps<<1
%  In order to solve this equation an error function is calcultaed
%  del is obtained using an iterative process
%  intgrl=cpdf(x2)-cpdf(x1)
%  del=x2-x1
%

ddel=1;
x1=0;
x=x1;
incr=[];

%  evaluate the epsilon value

eps=normcdf(x1+del,0,1)-normcdf(x1,0,1);
```

```
%  set the min % increment for del

epsdel=.01;

%  find sequence of x variable values
while abs(x1)<=5
        while ddel>abs(epsdel*del)
                ferr=normcdf(x1+del,0,1)-normcdf(x1,0,1)-eps;
                derferr=normpdf(x1+del,0,1);
                if derferr==0
                        break;
                end;
                deldel=(-ferr/derferr);
                del=del+deldel;
                ddel=abs(deldel);
        end; % while ddel
        x1=x1+del;
        ddel=1;
        incr=[incr del];
        x=[x x1];
end; % while x1

if derferr==0
        incr=incr(1:size(incr,2)-1);
        x=x(1:size(x,2)-1);
end;
```

A.8 Program: Ndsph.m

```
function Gridf=Ndsph(radius,alpha,dr,iter,inGrid)

inGrid;

if iter==2
        Gridf=inGrid;
else
        Gridf=[];
        for i=1:length(alpha)
                tempGrid=[inGrid alpha(i)];
                r1=radius*abs(cos(alpha(i)));
                dAlpha1=dr/r1;
```

```
                Angle1=[0 :dAlpha1: pi-dAlpha1];
                if iter-1==2
                        for j=1:size(tempGrid,1)
                                outGrid=[ones(length(Angle1),1)*tempGrid(j,:) Angle1'];
                        end;
                        Gridf=[Gridf;outGrid];
                else
                        Gridf=[Gridf;Ndsph(radius,Angle1,dr,iter-1,tempGrid)];
                end;
        end; % for i
end;
return;
```

A.9 Program: newalgma.m

```
% this program generates noisy signals out of two
% prototypes and applies pdf based information measure
% to identify the best disciminants for the generated
% signals using repeated Haar transform

% some environment setups..
clear all; % clears all variables and functions from the workspace
pack; % Consolidate workspace memory
clf; % clears the figure
clc;

start=cputime;
hold off;
absinfoincr=.005;
rreddesinfo=.1;


computer=4;
fid=1;
if computer==1
  %fid      = fopen('/home/homer/1/c/asareen/matlab/dumpdata.txt','w');
  fiddata  = fopen('/home/homer/1/c/asareen/matlab/plotdata.txt','w');
  fiderror = fopen('/home/homer/1/c/asareen/matlab/error.txt'   ,'w');
  fidPs    = fopen('/home/homer/1/c/asareen/matlab/p1p2.txt'    ,'w');
  fidinfo  = fopen('/home/homer/1/c/asareen/matlab/plotinfo.txt','w');
elseif computer==2
  %fid      = fopen('d:\users\aman\matlab\work\dumpdata.txt','w');
  fiddata  = fopen('d:\users\aman\matlab\work\plotdata.txt','w');
```

```
  fiderror = fopen('d:\users\aman\matlab\work\error.txt'   ,'w');
  fidPs    = fopen('d:\users\aman\matlab\work\p1p2.txt'    ,'w');
  fidinfo  = fopen('d:\users\aman\matlab\work\plotinfo.txt','w');
elseif computer==3
  %fid      = fopen('c:\users\aman\matlab\work\dumpdata.txt','w');
  fiddata  = fopen('c:\users\aman\matlab\work\plotdata.txt','w');
  fiderror = fopen('c:\users\aman\matlab\work\error.txt'   ,'w');
  fidPs    = fopen('c:\users\aman\matlab\work\p1p2.txt'    ,'w');
  fidinfo  = fopen('c:\users\aman\matlab\work\plotinfo.txt','w');
else
fiddata=1;
fiderror=1;
fidPs=1;
fidinfo=1;
end;

% setup the way each plot of information matrix is to be plotted.

%stringmat=['kx-';'b*-';'g+-';'ro:';'m*:';'kv:';'bd:';'gs:';'rh:';'mp-'];
stringmat=['r-';'b-';'g-';'r:';'m:';'b:';'g:';'r:';'m-'];


load c:\users\aman\matlab\work\sigFile

numCoeff=size(N,1);
Matrix=N/max(max(N));
cols=size(Matrix,2);
signal1=Matrix(:,1);
signal2=Matrix(:,cols/2+1);

% determine levels ????
levls=round(log10(numCoeff)/log10(2));
x=1:numCoeff;

rows=numCoeff;
coeffMatrix=diag(ones(1,numCoeff));
coeffD=coeffMatrix;
for i=1:levls
  coeffC=[];
  begn=1;
  while begn<numCoeff
    oddr=(begn:2:begn+rows-1);
    evnr=(begn+1:2:begn+rows-1);
    coeffA=[];
```

```
    coeffB=[];
    for j=1:length(oddr)
      coeffA(j,:)=( coeffMatrix(oddr(j),:) + coeffMatrix(evnr(j),:) )/2;
      coeffB(j,:)=coeffMatrix(oddr(j),:) - coeffMatrix(evnr(j),:);
    end;
    coeffC=[coeffC;coeffA;coeffB];
    begn=begn+rows;
  end; % while
  coeffMatrix=coeffC;
  coeffD=[coeffD;coeffC];
  rows=round(rows/2);
end; %for i

coeffMatrix=coeffD;
selectedcoeffMatrix=[];

% choice of which plots to plot.
plotOriginal=1;
plotNoisy=1;
plotInfo1D=1;
plotInfo2D=1;

getsigs=0;
if getsigs==1
  % define signals here.
  signal1=[0.05, 0.53, 1.01, 0.56, 0.07, -0.501, -1.03, -0.52]';
  signal2=[0.0033, 1.04, 0.002, -1.09, 0.03, 0.504, 2.2, 0.006]';

  %signal2=[1, .5 -.5 .5 1 -.5 0 .5]';
  epsil=0.01;
  epsil2=-0.02;
  %signal1=epsil+[0. 0.25, 0.5, 0.75, 1, 0.75, 0.5, 0.25, 0, -0.25, -0.5, -0.75, -1, -0.75, -
0.25, 0]';
  %signal2=epsil2+[0, 0, -0.2, -0.5, -1, -0.85, -0.75, -0.1, 0.1, 0.5, 1, 0.1, -0.4, -0.85, -
0.75, 0]';
  %signal1=epsil+[0, 0, -0.1, 0, 0, 0.05, 0, -0.1, 0.1, 0.5, 1, 0.1, -0.05, 0.05, 0.05, 0]';

  % Number of coefficients of the Signal.
  numCoeff=length(signal1);

  % add noise to each signal

  sig1noiselevel=6;
  sig2noiselevel=6;
```

```
  Matrix=[];
  for i=1:50
    Matrix(:,i)=signal1+sig1noiselevel*rand(numCoeff,1);
  end;
  for i=51:100
    Matrix(:,i)=signal2+sig2noiselevel*rand(numCoeff,1);
  end;
end;

% plot original signal.
if plotOriginal==1
  figure;
  plot(signal1,'r');
  hold on;
  plot(signal2,'b');
  xlabel('Signal coordinate');
  ylabel('Signal value');
  title('Original signals');
end;

% plot noisy signal.
if plotNoisy==1
  figure;
  plot(Matrix(:,1:cols/2),'r');
  hold on;
  plot(Matrix(:,cols/2+1:cols),'b');
  xlabel('Signal coordinate');
  ylabel('Signal value');
  title('Two signals with a random noise');
end;

% initialise the variables
previousInfo=0;
currentInfo=1;
iterations=1;
numOfLoops=0;
plotarray1D=[];
plotarray2D=[];
infoarray=[];
figure;

% start the main loop and continue till the desired accuracy is achieved.
%while abs(currentInfo - previousInfo) > absinfoincr & abs(currentInfo-previousInfo)/(1-
previousInfo)>rreddesinfo
```

```
while abs(currentInfo - previousInfo) > absinfoincr
  IterationNumber=iterations+8*numOfLoops;
 fprintf(fid,'Iteration Number = %i\n',IterationNumber);
 iterations=iterations+1;
 % check the iteration being done. if more than 8 then reset it to 1.
 % the reason is that we have only 8 ways of ploting the wave form.
 if iterations>8
   numOfLoops=numOfLoops+1;
   iterations=1;
 end;

 rows=numCoeff;
 D=Matrix;
 for i=1:levls
   C=[];
   begn=1;
   while begn<numCoeff
     oddr=(begn:2:begn+rows-1);
     evnr=(begn+1:2:begn+rows-1);
     % this normalization is used to balance Walsh coefficients
     % of a raw signal
     %A=(Matrix(oddr,:)+Matrix(evnr,:))/2;
     A=(Matrix(oddr,:)+Matrix(evnr,:))/2;
     % subtract pairwise
     B=Matrix(oddr,:)-Matrix(evnr,:);
     C=[C;A;B];
     begn=begn+rows;
   end; % while
   Matrix=C;
   D=[D;C];
   rows=round(rows/2);
 end; %for i

 Matrix=D;
 [rowsMatrix colsMatrix] = size(Matrix);

 % make other matrices to easy out some other computations.
 tMatrix = Matrix';
 sig1Matrix = Matrix(:,1:cols/2);
 sig2Matrix = Matrix(:,cols/2+1:cols);

 % determine the minimum and maximum value of the signal.
 rmin = min(tMatrix);
 rmax = max(tMatrix);
```

```
  % determine the standard deviation of each signal.
  D1=std (sig1Matrix');
  D2=std (sig2Matrix');

  % determine mean of each signal.
  E1=mean(sig1Matrix');
  E2=mean(sig2Matrix');

% set probability of each signal to be 1.
  P1=1;
  P2=1;

  spread=5;
  resolve=200;
  numSteps = 300;
% calculate the information

  info1=infopdfV(P1,P2,E1,E2,D1,D2,spread,numSteps);

% sort the information matrix to extract the best "numCoeff" number of elements.
  [info1 index] = sort(info1);
  orderedinf=fliplr(info1);
  previnfo = orderedinf(1);
  index=fliplr(index);
  plotarray1D = [plotarray1D;orderedinf(1:numCoeff)];
  infoarray=[infoarray;orderedinf(1:numCoeff)];
  if plotInfo1D==1
    subplot(1,2,1);
    axis([1,max(x),0,1]);
    plot(x,orderedinf(1:numCoeff),stringmat(iterations,:));
    xlabel('Most selective coefficients in decreasing order');
    ylabel('Information value');
    title('Selection of coefficients in various iterations');
    hold on;
  end;
  maxinfo=orderedinf(1);

  numPoints=200;
  dimension=3;
  for i=2:dimension
    fprintf(fid,'Working on %i dimensions\n',i);
    selectMatrix=Matrix(index(1:i-1),:);
    selectedIndex=index(1:i-1);
```

```
    for j=1:size(Matrix,1)
      if length(find(selectedIndex==j)) > 0
        info_ = previnfo;
        tempinfo(j) = info_;
      else
        tempMatrix=[selectMatrix;Matrix(j,:)];
        sig1Matrix = tempMatrix(:,1:cols/2);
        sig2Matrix = tempMatrix(:,cols/2+1:cols);
        Mean1=mean(sig1Matrix');
        Mean2=mean(sig2Matrix');
        Cov1=cov(sig1Matrix');
        Cov2=cov(sig2Matrix');
        P1=1;P2=1;
        eps=10^(-12);
        if abs(det(Cov1))<eps & abs(det(Cov2))<eps
          tempinfo(j)=maxinfo;
        else
          %keyboard;
          %info_1=newalgof(sig1Matrix',sig2Matrix')
          info_=infoND(sig1Matrix',sig2Matrix',Mean1,Mean2,Cov1,Cov2,P1,P2);
          tempinfo(j) = info_;
        end;
      end; % if length(find(selectedIndex==j)) > 0
    end; % for j
    [info1 index] = sort(tempinfo);
    orderinf=fliplr(info1);
    index=fliplr(index);
    infoarray=[infoarray;orderinf(1:numCoeff)];
    maxinfo=orderinf(1);
  end; % for i=2:dimension
  plotarray2D=[plotarray2D;orderinf(1:numCoeff)];
%  printVec(plotarray1D,'plotarray1D',fiddata);
%  printVec(plotarray2D,'plotarray2D',fiddata);
%  printVec(infoarray,'infoarray',fidinfo);

% update the values of the information matrix
  previousInfo=currentInfo;
  currentInfo=orderinf(1);
% plot the information matrix.
  if plotInfo2D==1
    subplot(1,2,2);
    axis([1,max(x),0,1]);
    plot(x,orderinf(1:numCoeff),stringmat(iterations,:));
    xlabel('Most selective coefficients in decreasing order');
```

```
    ylabel('Information value');
    title('Selection of coefficients in various iterations');
    hold on;
  end;
% reset the Matrix before going into the other loop.
  rownum = [selectedIndex index(1:numCoeff-1)];
  Matrix=Matrix(rownum,:);
  selectedcoeffMatrix=[selectedcoeffMatrix;coeffMatrix(rownum,:)];
fprintf(fid,'Iteration %i complete. Hit any key...\n',IterationNumber);
%pause;
%if IterationNumber==2 dbstop; end;
end; %while iterations

if fiddata==1
else
  fclose(fiddata);
  fclose(fiderror);
  fclose(fidPs);
  fclose(fidinfo);
end;


TimeTaken=cputime-start


plotharr=1;
if plotharr==1
  M=selectedcoeffMatrix(1:numCoeff,:);
  for i=2:IterationNumber
    M=selectedcoeffMatrix(numCoeff*(i-1)+1:numCoeff*i,:)*M;
  end;

  figure;
  %plotting the interpretation of haar transform

  for i=1:4
    subplot(2,2,i);
    bar(x,M(i,:),'r');
  end;
end; % if plotHarr

figure;
plot(signal1,'b')
hold on
```

```
plot(signal2,'r')
e=M(1,:);
en=e/norm(e)*((norm(signal1)+norm(signal2))/2);
plot(en(1,:),'g')
```

A.10 Program: infoND.m

```
function info=infoND(signal1,signal2,Mean1,Mean2,Cov1,Cov2,P1,P2)

%  this function evaluates the information index for two n-dimensional
%       distributions
%       signal1, and signal2 represent two sets of random signals with
%       normal distribution and mean values Mean1 and Mean2
%       as well as covariance matrices Cov1 and Cov2
%       P1 and P2 are apriori probabilities of the two signals

%       generate points by gausian(normal)distribution
%       points=randn(m,n);
%       generate two signals
%       signal1=points*R1+ones(m,1)*V1;
%       signal2=points*R2+ones(m,1)*V2;

m=size(signal1,1);
n=size(signal1,2);

%       this function finds information for n-dimensional pds functions
%       described by Mean1, Mean2, vectors and Cov1 and Cov2 matrices
%       signal1 are random points generated by pdf1 distribution
%       and signal2 are points generated by pdf2

x1_mean1=signal1-ones(m,1)*Mean1;
x2_mean2=signal2-ones(m,1)*Mean2;
x2_mean1=signal2-ones(m,1)*Mean1;
x1_mean2=signal1-ones(m,1)*Mean2;

x_ic_xt=inv(Cov1) * x1_mean1';
nexp=zeros(size(x1_mean1,1),1);
for jj=1:size(Cov1,1)
  nexp=nexp+x1_mean1(:,jj).*x_ic_xt(jj,:)';
end;

pdf11=P1*exp(-0.5 * nexp) / sqrt( (2*pi)^n*abs(det(Cov1)));
```

```
x_ic_xt=inv(Cov2) * x2_mean2';
nexp=zeros(size(x1_mean1,1),1);
for jj=1:size(Cov1,1)
  nexp=nexp+x2_mean2(:,jj).*x_ic_xt(jj,:)';
end;

pdf22=P2*exp(-0.5 * nexp) / sqrt( (2*pi)^n*abs(det(Cov2)));

x_ic_xt=inv(Cov1) * x2_mean1';
nexp=zeros(size(x1_mean1,1),1);
for jj=1:size(Cov1,1)
  nexp=nexp+x2_mean1(:,jj).*x_ic_xt(jj,:)';
end;
pdf21=P1*exp(-0.5 * nexp) / sqrt( (2*pi)^n*abs(det(Cov1)));

x_ic_xt=inv(Cov2) * x1_mean2';
nexp=zeros(size(x1_mean1,1),1);
for jj=1:size(Cov1,1)
  nexp=nexp+x1_mean2(:,jj).*x_ic_xt(jj,:)';
end;

pdf12=P2*exp(-0.5 * nexp) / sqrt( (2*pi)^n*abs(det(Cov2)));

P1w=P1/m*sum(pdf11./(pdf11+pdf12));
P2w=P2/m*sum(pdf22./(pdf21+pdf22));
P12w=P1/m*sum(pdf12./(pdf11+pdf12));
P21w=P2/m*sum(pdf21./(pdf21+pdf22));

if P1w==0 P1w=1; end;
if P2w==0 P2w=1; end;
if P12w==0 P12w=1; end;

%  find the entropy measure

entr=P1w*log(P1w)+P2w*log(P2w)+P12w*log(P12w)+P21w*log(P21w) -
(P1*log(P1)+P2*log(P2));

%  find the information

P11=P1^2/(P1+P2);
P22=P2^2/(P1+P2);
P12=P1*P2/(P1+P2);
```

```
maxentr=P11*log(P11)+P22*log(P22)+2*P12*log(P12) -(P1*log(P1)+P2*log(P2));
info=1-entr/maxentr;
```

Appendix B

B.1 Program: Haar.vhd

```
LIBRARY ieee;
USE ieee.ALL;

ENTITY haar IS
   PORT (
      clock : IN  bit;

      -- Inputs
      in1   : IN  integer RANGE -127 TO 127;
      in2   : IN  integer RANGE -127 TO 127;
      in3   : IN  integer RANGE -127 TO 127;
      in4   : IN  integer RANGE -127 TO 127;
      in5   : IN  integer RANGE -127 TO 127;
      in6   : IN  integer RANGE -127 TO 127;
      in7   : IN  integer RANGE -127 TO 127;
      in8   : IN  integer RANGE -127 TO 127;

      --Outputs
      out1  : OUT integer RANGE -127 TO 127;
      out2  : OUT integer RANGE -127 TO 127;
      out3  : OUT integer RANGE -127 TO 127;
      out4  : OUT integer RANGE -127 TO 127;
      out5  : OUT integer RANGE -127 TO 127;
      out6  : OUT integer RANGE -127 TO 127;
      out7  : OUT integer RANGE -127 TO 127;
      out8  : OUT integer RANGE -127 TO 127
   );
END haar;

ARCHITECTURE haar OF haar IS

   COMPONENT bufgs
      PORT (
         i : IN  bit;
         o : OUT bit
      );
   END COMPONENT;

   COMPONENT reg
```

```
    PORT (
      input  : IN  integer RANGE -127 TO 127;
      clk    : IN  bit;
      output : OUT integer RANGE -127 TO 127
    );
  END COMPONENT;

  COMPONENT adddiv
    PORT (
      a   : IN  integer RANGE -127 TO 127;
      b   : IN  integer RANGE -127 TO 127;
      clk : IN  bit;
      c   : OUT integer RANGE -127 TO 127
    );
  END COMPONENT;

  COMPONENT difference
    PORT (
      a   : IN  integer RANGE -127 TO 127;
      b   : IN  integer RANGE -127 TO 127;
      clk : IN  bit;
      c   : OUT integer RANGE -127 TO 127
    );
  END COMPONENT;


  SIGNAL   out_r1, out_r2, out_r3, out_r4, out_r5, out_r6, out_r7,
    out_r8 : integer RANGE -127 TO 127;
  SIGNAL   out_a1, out_a2, out_a3, out_a4, out_a5, out_a6, out_a7,
    out_a8 : integer RANGE -127 TO 127;
  SIGNAL   out_s1, out_s2, out_s3, out_s4, out_s5, out_s6, out_s7,
    out_s8 : integer RANGE -127 TO 127;
  SIGNAL   clk : bit;

BEGIN
  xbufgs: bufgs
    PORT MAP (
      clock,
      clk
    );
  --  Input to Register here
  r1: reg
    PORT MAP ( in1, clk, out_r1 );
  r2: reg
```

```
   PORT MAP ( in2, clk, out_r2 );
r3: reg
   PORT MAP ( in3, clk, out_r3 );
r4: reg
   PORT MAP ( in4, clk, out_r4 );
r5: reg
   PORT MAP ( in5, clk, out_r5 );
r6: reg
   PORT MAP ( in6, clk, out_r6 );
r7: reg
   PORT MAP ( in7, clk, out_r7 );
r8: reg
   PORT MAP ( in8, clk, out_r8 );

--  Input to Add_Divide and Difference for stage 1
a1: adddiv
   PORT MAP ( out_r1, out_r2, clk, out_a1 );
a2: adddiv
   PORT MAP ( out_r3, out_r4, clk, out_a2 );
a3: adddiv
   PORT MAP ( out_r5, out_r6, clk, out_a3 );
a4: adddiv
   PORT MAP ( out_r7, out_r8, clk, out_a4 );
s1: difference
   PORT MAP ( out_r1, out_r2, clk, out_s1 );
s2: difference
   PORT MAP ( out_r3, out_r4, clk, out_s2 );
s3: difference
   PORT MAP ( out_r5, out_r6, clk, out_s3 );
s4: difference
   PORT MAP ( out_r7, out_r8, clk, out_s4 );

--  Input to AddDiv and Difference for stage 2
a5: adddiv
   PORT MAP ( out_a1, out_a2, clk, out_a5 );
a6: adddiv
   PORT MAP ( out_a3, out_a4, clk, out_a6 );
s5: difference
   PORT MAP ( out_a1, out_a2, clk, out_s5 );
s6: difference
   PORT MAP ( out_a3, out_a4, clk, out_s6 );

a7: adddiv
   PORT MAP ( out_s1, out_s2, clk, out_a7 );
```

```
   a8: adddiv
      PORT MAP ( out_s3, out_s4, clk, out_a8 );
   s7: difference
      PORT MAP ( out_s1, out_s2, clk, out_s7 );
   s8: difference
      PORT MAP ( out_s3, out_s4, clk, out_s8 );

   --  Input to AddDiv and Difference for stage 3
   a9: adddiv
      PORT MAP ( out_a5, out_a6, clk, out1 );
   s9: difference
      PORT MAP ( out_a5, out_a6, clk, out2 );

   a10: adddiv
      PORT MAP ( out_s5, out_s6, clk, out3 );
   s10: difference
      PORT MAP ( out_s5, out_s6, clk, out4 );

   a11: adddiv
      PORT MAP ( out_a7, out_a8, clk, out5 );
   s11: difference
      PORT MAP ( out_a7, out_a8, clk, out6 );

   a12: adddiv
      PORT MAP ( out_s7, out_s8, clk, out7 );
   s12: difference
      PORT MAP ( out_s7, out_s8, clk, out8 );

END haar;
```

B.2 Program: Reg.vhd

```
LIBRARY ieee;
USE ieee.ALL;

ENTITY reg IS
   PORT (
      input  : IN  integer RANGE -127 TO 127;
      clk    : IN  bit;
      output : OUT integer RANGE -127 TO 127
   );
END reg;
```

```
ARCHITECTURE reg OF reg IS
BEGIN
   PROCESS(clk)
   BEGIN
      IF clk'EVENT AND clk = '1' THEN
         output <= input;
      END IF;
   END PROCESS;
END reg;
```

B.3 Program: Add_Divide.vhd

```
LIBRARY ieee;
USE ieee.ALL;

ENTITY adddiv IS
   PORT (
      a   : IN  integer RANGE -127 TO 127;
      b   : IN  integer RANGE -127 TO 127;
      clk : IN  bit;
      c   : OUT integer RANGE -127 TO 127
   );
END adddiv;

ARCHITECTURE adddiv OF adddiv IS
BEGIN
   PROCESS (clk)
   BEGIN
      IF clk'EVENT AND clk = '1' THEN
         c <= (a + b)/2;
      END IF;
   END PROCESS;
END adddiv;
```

B.4 Program: Subractor.vhd

```
LIBRARY ieee;
USE ieee.ALL;
```

```
ENTITY difference IS
   PORT (
      a   : IN  integer RANGE -127 TO 127;
      b   : IN  integer RANGE -127 TO 127;
      clk : IN  bit;
      c   : OUT integer RANGE -127 TO 127
   );
END difference;

ARCHITECTURE difference OF difference IS
BEGIN
   PROCESS (clk)
   BEGIN
      IF clk'EVENT AND clk = '1' THEN
         c <= a - b;
      END IF;
   END PROCESS;
END difference;
```