Reconfigurable Wavelet-Based Architecture For Pattern Recognition Applications Using

A Field Programmable Gate Array

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

Abdulqadir Al-aqeeli

August, 1998
Acknowledgments

I wish to thank my thesis advisor, Dr. Janusz Starzyk, for his constant patience and great assistance. His way of teaching was the best I’ve ever seen. I think 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 wish to thank also Dr. Curtis, Dr. Celenk, Dr. Dill and Dr. Snyder for their precious time, valuable help and support with their guidance and ideas.

Special thanks go to my parents, and my wife, whose support gave me spiritual strength in pursuing my ambition and staying in the United States. My dear friend, Ahmad Alsolaim, can not be forgotten because he was the main supporter and his continuous advice helped me in completing my thesis. He also, helped me in all of my courses.

I am very thankful to Aman Sareen who supported me in my thesis. He provided me with his matlab codes' results that were used in developing the main architecture. Also, I wish to thank my friend Mohammed Alsharekh who helped me in the sampling issue of my thesis. Finally, thanks go to all my friends who assisted me upon request and were very helpful in their suggestions.
# Table of Content

<table>
<thead>
<tr>
<th>Approval Page</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgement</td>
<td></td>
</tr>
<tr>
<td>Table of Content</td>
<td>i</td>
</tr>
<tr>
<td>List of Figures</td>
<td>iii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>v</td>
</tr>
</tbody>
</table>

1. **INTRODUCTION** ........................................... 1  
   1.1 Pattern Recognition and Wavelets ................................ 1  
   1.2 Field Programmable Gate Arrays (FPGAs) .......................... 2  
   1.3 Overview ................................................................ 3  

2. **SRAM FPGAS** .................................................. 4  
   2.1 Introduction .................................................. 4  
   2.2 Xilinx 4000-Series FPGAs ........................................ 4  
      2.2.1 Configurable Logic Blocks .................................. 6  
      2.2.2 Input / Output Blocks ...................................... 8  
      2.2.3 Programmable Interconnect ................................... 9  
   2.3 Using Function Generators as RAM .................................. 10  
      2.3.1 Dual-Port Edge-Triggered Mode ............................... 14  

3. **WAVELET-BASED ALGORITHM FOR PATTERN RECOGNITION APPLICATIONS** ........................................ 17  
   3.1 The Problem ................................................... 17  
   3.2 The Algorithm .................................................. 18  
   3.3 FPGA Implementations for the Haar Wavelet Process ........ 21  
      3.3.1 Parallel Register-Based Architecture ....................... 21  
      3.3.2 Serial Register-Based Architecture ......................... 25  
   3.4 The Results of the Two Register-Based Architectures .......... 26  

4. **THE RECONFIGURABLE ARCHITECTURE USING FPGA'S DUAL-PORT RAM** ........................................ 29  
   4.1 Using Dual-Port RAM ........................................... 29  
   4.2 The RAM-Based Design .......................................... 31  
      4.2.1 The Processing Element (PE) .................................. 31  
      4.2.2 The Controllers ................................................ 34
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>General architecture of the Xilinx XC4000 FPGA</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Configurable logic block (CLB)</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>Memory lookup table used in F and G function generators</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>XC4000 input/output block (IOB)</td>
<td>9</td>
</tr>
<tr>
<td>2.5</td>
<td>High level routing diagram of the XC4000 FPGA</td>
<td>11</td>
</tr>
<tr>
<td>2.6</td>
<td>Programmable switch matrix</td>
<td>11</td>
</tr>
<tr>
<td>2.7</td>
<td>16x2 (or 16x1) edge-triggered single-port RAM</td>
<td>12</td>
</tr>
<tr>
<td>2.8</td>
<td>32x1 edge-triggered single-port RAM</td>
<td>13</td>
</tr>
<tr>
<td>2.9</td>
<td>16x1 edge-triggered dual-port RAM</td>
<td>14</td>
</tr>
<tr>
<td>2.10</td>
<td>Simplified mode of dual-port RAM</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>The block diagram of the problem</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td>Haar wavelet for m samples</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>Parallel architecture for the Haar wavelet (eight samples)</td>
<td>22</td>
</tr>
<tr>
<td>3.4</td>
<td>8-bit register mapped into eight IOBs</td>
<td>23</td>
</tr>
<tr>
<td>3.5</td>
<td>Computing the average with registered output</td>
<td>23</td>
</tr>
<tr>
<td>3.6</td>
<td>8-bit subtractor with registered output</td>
<td>24</td>
</tr>
<tr>
<td>3.7</td>
<td>Implementation of two one-bit adders using a CLB</td>
<td>25</td>
</tr>
<tr>
<td>3.8</td>
<td>Serial architecture for Haar wavelet (eight samples)</td>
<td>26</td>
</tr>
<tr>
<td>4.1</td>
<td>Block diagrams of the Haar wavelet for the 8 and 16 samples</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>Simplified block diagram of the Haar wavelet using dual-port RAM</td>
<td>31</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>Block diagram of processing element</td>
<td>32</td>
</tr>
<tr>
<td>4.4</td>
<td>Block diagram shows the architecture that computes the average</td>
<td>33</td>
</tr>
<tr>
<td>4.5</td>
<td>State machine representation of the controller block &quot;CONT1&quot;</td>
<td>34</td>
</tr>
<tr>
<td>4.6</td>
<td>The AHDL code of the controller block &quot;CONT1&quot;</td>
<td>35</td>
</tr>
<tr>
<td>4.7</td>
<td>Simplified block diagram of the system</td>
<td>37</td>
</tr>
<tr>
<td>4.8</td>
<td>The whole system</td>
<td>39</td>
</tr>
<tr>
<td>4.9</td>
<td>The AHDL code of the controller &quot;Main&quot;</td>
<td>41</td>
</tr>
<tr>
<td>5.1</td>
<td>Part of the waveform results of the functional simulation</td>
<td>44</td>
</tr>
<tr>
<td>5.2</td>
<td>Architecture of the 16x8 RAM</td>
<td>48</td>
</tr>
<tr>
<td>5.3</td>
<td>Architecture of the block that computes the average</td>
<td>49</td>
</tr>
<tr>
<td>5.4</td>
<td>Architecture of a 2x1 MUX</td>
<td>50</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Dual-port edge-triggered RAM signals</td>
<td>16</td>
</tr>
<tr>
<td>3.1</td>
<td>Hardware requirements and the performance of the parallel architecture</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>Hardware requirements and the performance of the serial architecture</td>
<td>28</td>
</tr>
<tr>
<td>4.1</td>
<td>Comparison between the parallel and the RAM-based architectures</td>
<td>36</td>
</tr>
<tr>
<td>4.2</td>
<td>The selected coefficients and their locations</td>
<td>40</td>
</tr>
<tr>
<td>4.3</td>
<td>The data stored in the three ROM blocks</td>
<td>40</td>
</tr>
<tr>
<td>5.1</td>
<td>The generated coefficients after the Matlab simulation</td>
<td>46</td>
</tr>
<tr>
<td>5.2</td>
<td>The generated coefficients after simulating the FPGA's architecture</td>
<td>47</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

During the past 20 years, there has been a considerable growth of interest in problems related pattern recognition knowledge extraction from data bases, intelligent signal processing, and their hardware implementation. This interest has created an increasing need for theoretical methods and experimental software and hardware for use in the design of pattern recognition and related areas [1]. In general, pattern recognition is concerned primarily with the description and analysis of measurements taken from physical or mental processes. This thesis focuses on hardware implementation of a wavelet-based algorithm that can be used, with the help of some external neural networks, for pattern recognition applications. This hardware implementation will use many advantages of the advanced technology of field programmable gate arrays. A brief introduction to wavelets, pattern recognition and field programmable gate arrays follows in this chapter.

1.1 Pattern Recognition and Wavelets

In the beginning, a simple definition of wavelets should be given. "Wavelets are mathematical functions that cut up data into different frequency components, and then study each component with a resolution matched to its scale. They have advantages over traditional Fourier methods in analyzing physical situations where the signal contains discontinuities and sharp spikes. Wavelets were developed independently in the fields of
mathematics, quantum physics, electrical engineering, and seismic geology. Interchanges between these fields during the last ten years have led to many new wavelet applications such as image compression, human vision, radar, target detection, speech recognition, and earthquake prediction" Graps, 1995. These are only some of the many reasons why pattern recognition is very related to wavelets [2].

In this thesis wavelets are used in a unique way to preprocess input data for pattern recognition. The method iterates on Haar wavelet and uses entropy based measure to define a unique transformation, optimum for a given pattern recognition problem. The theoretical foundation of this wavelet preprocessing is subject of another thesis [6].

1.2 Field Programmable Gate Arrays

A field programmable gate array (FPGA) is a general-purpose, multi-level programmable logic device that is customized in the package by the end users. FPGAs are composed of blocks of logic connected with programmable interconnect. The programmable interconnect between blocks allows users to implement multi-level logic, removing many of the size limitations of the PLD-derived two-level logic structure. This extensible architecture can currently support hundreds of thousands of logic gates at system speeds in tens of mega-hertz. The size, structure and number of blocks; and the amount and connectivity of the interconnect vary considerably among FPGA architectures. This difference in architectures is driven by different programming technologies and different target applications of the part [3].
FPGAs are useful in many different applications. Digital signal processing (DSP) applications can be implemented in FPGAs faster and more cheaply than with the current DSP chips. Also, micro-controllers can be implemented in FPGAs. Thus, the need to buy old micro-controller chips is gone with the availability of configuring the FPGA to do the job of the micro-controller. Image processing, compression and recognition are a few of the applications that can be implemented using FPGAs.

1.3 Overview

After this brief introduction, the SRAM FPGAs will be discussed in detail in chapter 2. Because a Xilinx XC4000 SRAM FPGA is used in the design, its basic architecture is presented in detail. Most of the discussion on Xilinx FPGA architecture is based on the Xilinx FPGA Data Sheet [4]. Not only the architecture, but even the way in which it is presented. Chapter 3 discusses a new technique in pattern recognition using wavelet-based algorithm. This algorithm depends on one-dimensional Haar wavelet transform. It will include two different architectures mapped to a Xilinx XC4000 FPGA. However, another architecture is developed in Chapter 4. This architecture is mapped into the FPGA with the use of the available user RAMs. Chapter 5 presents the discussion of the results and the design recommendations. The last chapter, chapter six, summarizes the work done in this thesis and presents the future work. The Matlab programs used in the thesis are shown in the Appendix A. Appendix B has all the ABEL Hardware Description Language (AHDL) codes used in the design. Finally, Appendix C shows the waveform of the simulation of the implemented hardware.
Chapter 2
SRAM Field Programmable Gate Arrays

2.1 Introduction

SRAM FPGAs are programmed by configuring memory cells from external sources. These memory cells, distributed within the FPGA, control the logic and interconnect that perform the function of the FPGA [5]. SRAM FPGAs can be reprogrammed to change, or modify, their functions while present in the systems. This capability, not available with any other type of logic, gives the system designer the ability implement intelligent designs. "An FPGA can even be reconfigured dynamically to perform different functions at different times", Xilinx, 1998 [4]. Reconfigurable FPGA, or reconfigurable processing units, can be used to design system that can be reconfigured for different operations, or to build multipurpose hardware. In other words, reprogrammability makes SRAM FPGAs ideal for rapid prototyping. To make use of SRAM FPGAs, they are reloaded as part of the normal operation of the system. This capability leads to significant improvement in effective density for those designs [3], [4].

2.2 Xilinx 4000-Series FPGAs

XC4000-series devices, also they are known as the Xilinx FPGAs, operate with high speed because of using advanced CMOS technology and enhanced architecture. XC4000-series SRAM FPGAs are built with a programmable architecture of configurable logic blocks (CLBs), interconnected by flexible routing resources, and surrounded by programmable input/output blocks (IOBs) (Figure 2.1) [4]. They have large routing resources to implement complex interconnect patterns. Loading configuration bits into
internal RAM cells programs the XC4000 FPGAs. The FPGA can either read its configuration bits from an external PROM, or the configuration bits can be loaded into the FPGA from an external device such as a personal computer. "Because these FPGAs can be reprogrammed an unlimited number of times, they can be used in intelligent designs where hardware is changed dynamically, or where hardware must be suited to different user applications", Xilinx, 1998 [4].

![General architecture of the Xilinx XC4000 FPGA](image)

Figure 2.1 General architecture of the Xilinx XC4000 FPGA [9].

XC4000 FPGAs include two major configurable circuits: configurable logic blocks (CLBs) and input/output blocks (IOBs). While CLBs provide the functional elements for building the user's logic, IOBs provide the interface between the package pins and internal signal lines [3]. Two other types of circuits are also available.
3-State buffers (TBUFs) driving horizontal long lines are associated with each CLB.

An on-chip oscillator is also available. Programmable interconnect resources provide routing paths to connect the inputs and outputs of these configurable circuits to the appropriate networks. Programming internal SRAM cells, as mentioned before, does the functionality of each circuit block. The values stored in these cells control the logic functions and interconnections implemented in the FPGA [3].

2.2.1 Configurable Logic Blocks (CLBs)

Configurable logic blocks implement the logic in an FPGA. The CLB architecture is shown in Figure (2.2). Two 4-input function generators (F and G) offer permitted flexibility. "Most combinatorial logic functions need four or fewer inputs", Xilinx, 1998 [4]. A third function generator (H) is also available. However, the H function generator has three inputs. The CLB can make special functions of up to nine variables. Each CLB contains two flip-flops that can be used to store the output of the function generators. However, the flip-flops and function generators can also be used independently. These flip-flops can be configured as latches [4].
Figure 2.2 Configurable logic block (CLB) [9].

Function generator outputs can also drive two outputs independent of the output of the flip-flops' outputs. Seventeen CLB inputs and outputs provide access to the function generators and flip-flops. These inputs and outputs connect to the programmable interconnect resources outside the block. The function generators, with outputs labeled F' and G', can each implement any function of four inputs. Each function generator is implemented as a memory lookup table (Figure 2.3). Therefore, the propagation delay is of the function implemented. A CLB can be used to implement any of the following functions. "It can implement any function of up to four variables plus any second function of up to four unrelated variables plus any third function of up to three unrelated
variables, any single function of five variables or some functions of up to nine variables", Xilinx, 1998 [4]. For instance the look-up table stored in ROM on Figure 2.3 can be used to realize $F = (A+D)$. The figure shows the selected output for $A=1$ and $D=0$.

![Memory lookup table used in F and G function generators.](image)

**2.2.2 Input/Output blocks (IOBs)**

Input/output blocks (IOBs) are responsible for the interface between the internal logic and the external package pins. Each IOB controls one pin and can be programmed for input, output or bi-directional signals [4]. Figure 2.4 shows a simplified block diagram of the XC4000 IOB. Signals to be output from the chip can be registered before output and enabled by a clock enable. Outputs can be optionally pulled up or down. Inputs can enter into the interior of the chip directly or registered [5].
2.2.3 Programmable Interconnect

The internal connections are made of metal parts with programmable interconnecting points and switching matrices to implement the preferred routing. A power matrix structure of routing resources is built to achieve efficient programmed routing (Figure 2.5). There are several types of interconnect. CLB routing controls and connects each row and column of the CLB array. IOB routing forms a Versa Ring around the outside of the CLB array. This routing connects the I/O with the internal configurable logic blocks (CLBs). Global routing consists of dedicated circuits primarily
designed to distribute clocks throughout the FPGA with very small delay. Also, global routing can be used for other high fanout signals [4]. Several interconnect types are distinguished by the relative length of their segments: single-length, double-length, quad, and long lines. In addition, direct connects allow fast data flow between adjacent CLBs, and between IOBs and CLBs. The single-length and double-length lines intersect at a programmable switch matrix (PSM). Each switch matrix consists of programmable pass transistors used to form connections between the lines (Figure 2.6). For instance, a single-length signal entering on the left side of the switch matrix can be routed to a single-length line on the top, right or bottom sides, or any combination thereof, if multiple branches are required. Also, a double-length signal can be connected to a double-length line on any of the other three edges of the programmable switch matrix [4].

2.3 Using Function Generators As RAM

"Optional modes for each CLB make the memory lookup tables in the F’ and G’ function generators usable as an array of Read/Write memory cells", Xilinx, 1998 [4]. The available modes are level-sensitive, edge-triggered, and dual-port edge-triggered RAMs. Each CLB can be programmed as a 16x2, 32x1 or 16x1 RAM. “XC4000-Series devices are the first programmable logic devices with edge-triggered (synchronous) and dual-port RAM accessible to the user”, Xilinx, 1998 [4]. Edge-triggered RAM eases system timing. Dual-port RAM improves the effective throughput of FIFO applications. These features can be individually programmed in any XC4000-series CLB [4].
Figure 2.5 High level routing diagram of the XC4000 FPGA [8].

Figure 2.6 Programmable switch matrix [8].
The function generators in any CLB can be configured as RAM arrays in the following sizes: two 16x1 RAMs, two data inputs and two data outputs with identical or different addressing for each RAM (Figure 2.7) [4].

![Diagram](image)

Figure 2.7 16×2 (or 16×1) edge-triggered single-port RAM [8].

It can also be configured as one 32x1 RAM, one data input and one data output (Figure 2.8). For example, the F function generator can be configured as a 16x1 RAM.
while the other function generators (G and H) are used to implement any function of up to five inputs [4].

![Diagram of 32x1 edge-triggered single-port RAM]

Figure 2.8 32x1 edge-triggered single-port RAM [8].

In addition, the XC4000-series RAM may have either of two timing modes. The first is edge-triggered (synchronous), where data is written by the edge of the CLB clock. In this mode, WE is used as a clock enable. The second mode is level-sensitive (Asynchronous), where an external WE signal is used as the write strobe. The selected timing mode applies to both function generators within a CLB when both are configured as RAM. The number of read ports is also programmable. One of them is single-port,
where each function generator has a common read/write port. The other one is the dual-port, where both function generators are configured together as a single 16x1 dual-port RAM with one write port and two read ports. Simultaneous read and write operations to the same or different addresses are possible [4].

2.3.1 Dual-Port Edge-Triggered Mode

In dual-port mode, both the two 4-input function generators (F and G) are used to produce a single 16x1 RAM with one write port and two read ports (Figure 2.9). The resulting RAM array can be read and written simultaneously at two independent addresses [4].

Figure 2.9 16x1 edge-triggered dual-port RAM [8].
Simultaneous read and write operations at the same address can be done. Dual-port mode always has edge-triggered write timing. Figure 2.10 shows a simple model of an XC4000-series CLB configured as dual-port RAM. One address port, labeled A[3:0], supplies both the read and write address for the F function generator. This function generator acts as a 16x1 single-port edge-triggered RAM array [4].

![Figure 2.10 Simplified mode of dual-port RAM](image)

The RAM output, single-port out (SPO), appears at the F function generator output. SPO, therefore, reflects the data at address A[3:0]. The other address port, labeled DPRA[3:0] for dual-port read address, supplies the read address for the G function generator. However, the write address for the G function generator comes from the...
address A[3:0]. The output from this 16x1 RAM, dual-port out (DPO), appears at the G function generator output. DPO, therefore, reflects the data at address DPRA[3:0]. Therefore, by using A[3:0] for the write address and DPRA[3:0] for the read address, and reading only the DPO output, a FIFO that can read and write simultaneously is implemented. Simultaneous access improves the effective throughput of the FIFO. The relationships between CLB pins and RAM inputs and outputs for dual-port, edge-triggered mode are shown in Table 2.1 [4].

Table 2.1 Dual-port edge-triggered RAM signals [8].

<table>
<thead>
<tr>
<th>RAM Signal</th>
<th>CLB Pin</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>D0</td>
<td>Data In</td>
</tr>
<tr>
<td>A[3:0]</td>
<td>F1-F4</td>
<td>Read Address for F, Write Address for F and G</td>
</tr>
<tr>
<td>DPRA[3:0]</td>
<td>G1-G4</td>
<td>Read Address for G</td>
</tr>
<tr>
<td>WE</td>
<td>WE</td>
<td>Write Enable</td>
</tr>
<tr>
<td>WCLK</td>
<td>K</td>
<td>Clock</td>
</tr>
<tr>
<td>SPO</td>
<td>F'</td>
<td>Single Port Out (addressed by A[3:0])</td>
</tr>
<tr>
<td>DPO</td>
<td>G'</td>
<td>Dual Port Out (addressed by DPRA[3:0])</td>
</tr>
</tbody>
</table>
Chapter 3

Wavelet-Based Algorithm for Pattern Recognition Applications

The purpose of pattern classification is to determine to which category or class a given sample belongs. The design of a classifier consists of two parts. One is to collect data samples from various classes and to find the boundaries that separate the classes. This process is called classifier design, or training. The other is to test the designed classifier by feeding the samples whose class identities are known [1].

3.1 The Problem

When a signal is received, it contains information from a source and some noise. By analyzing this signal one can get the correct information by applying some signal processing. However, to recognize the category of this signal, one must use a classifier circuit after the signal being processed. A training process was developed in a thesis written by Aman Sareen [6]. His training algorithm uses two signals and 100 noisy signals generated from the two signals. After applying simple Haar wavelet processes to the signals many times, the training algorithm selects some of the generated coefficients to be new input coefficients for the Haar wavelet again. Selected coefficients depend on the training algorithm, which uses the probability density function according to space distribution of the training samples. Figure 3.1 shows the block diagram of the training process. Samples of the signal are read and need to be processed using the Haar wavelet.
Then, the block labeled "Selecting Coefficients" selects and feeds back the selected coefficients. Section 3.2 presents details of the algorithm used.

![Block Diagram](image)

Figure 3.1 The block diagram of the problem

### 3.2 The Algorithm

The wavelet used is the Haar wavelet described in [7]. The following equations show the Haar transform.

\[
\begin{align*}
    r(0) &= \frac{x(0) + x(1)}{2} \\
    d(0) &= x(0) - x(1)
\end{align*}
\]  

(3.1)

When applying this formula to the coefficients of the original signal \(2^n\) coefficients), new coefficients are generated and a Haar matrix is formed. The total number of coefficients in the matrix (including the coefficients of the original signal) equals \((n+1) \times 2^n\) (Figure 3.2). This algorithm was implemented by the following code:
levls= round (log10(numCoeff)/log10(2));

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;

% subtract pairwise

B=Matrix(oddr,:)-Matrix(evnr,:);

C=[C;A;B];

bgn=bgn+rows;

end;

Matrix=C;

D=[D;C];

rows=round(rows/2);

end;
Once the Haar transform has been performed, the entropy measure is evaluated. For each Haar wavelet coefficient the probability density function (pdf) of the two classes which are subject of the classification task are computed using

\[
pdf = \frac{P \exp(-0.5 \frac{(x - m)^2}{\sigma})}{\sqrt{2\pi\sigma}}
\]

(3.2)

where \( m \) and \( \sigma \) are the mean and the standard deviation respectively.

Figure 3.2 Haar wavelet for \( m \) samples.

"note: \( m = 2^n \)" Total number of coefficients = \((n+1) \cdot m\)
Then, probabilities of correct classification are computed in the subspace of the input variable $x$ for which part of a given class is larger than the pdf of the other class. Probabilities of misclassification are computed as a difference of class probabilities and probabilities of correct classification. Then the entropy measure is computed. See the Matlab program 2 in Appendix A. For one-dimensional analysis, the coefficients that give the maximum information are selected. The whole process can be repeated several times (three is accepted). After that, the resultant information is used to tell what the noisy signal (input signal) means. This last step depends on the learning system to choose one of the learned signals [6].

3.3 FPGA Implementations For The Haar Wavelet Process

To implement the process of the Haar wavelet (Equation 3.1), we assumed that the following conditions are met. The number of samples per frame is eight. Each sample is represented by 8-bit number. Sampling the signals is taken care of before we receive the input data. In order to design the circuit, we thought of using the FPGA registers. Two architectures, parallel and serial, were developed.

3.3.1 Parallel Architecture

Because FPGAs support designs of parallel processing, the first choice for the architecture of this problem is parallel. Figure 3.3 shows the block diagram of the design. All the eight data samples are fed, stored, and processed at the same time. Each 8-bit register was built from four CLBs because each CLB has a 2-bit register available to the user. However, to reduce the number of CLBs used, the first eight registers were built from 1bit registers found in the IOBs. This modification reduced the design by 32 CLBs.
Then all the other registers were placed inside the CLBs of the adders and the subtractors. This change saved 96 CLBs so the total of 128 CLBs was reduced from the original architecture. Figures 3.4, 3.5, and 3.6 show the architectures of the registers, the adders, and the subtractors. Figure 3.7 shows the implementation of two one-bit adders (block marked as A in Figure 3.5) by a single CLB. This way an 8-bit adder uses 5 CLBs.

Figure 3.3 Parallel architecture for Haar wavelet (eight samples).
Figure 3.4 8-bit register mapped into eight IOBs.

Figure 3.5 Computing the average with registered output (five CLBs).
Figure 3.6 8-bit subtractor with registered output (5 CLBs).
3.3.2 Serial Architecture

With the assumption of getting the eight samples in serial, with the least significant coefficient being fed first, another architecture was developed. Figure 3.8 shows this serial architecture. After each two clock cycles, two new coefficients are received. Then finding the average and the difference between the coefficients can be
done in one more clock cycle. This leads to a reduction of the number of registers needed.
The complexity in this design is in controlling when to process and when to store the coefficients and the resultant values. The architectures of the adders, subtractors, and the registers are similar to what was used in the parallel architecture.

Figure 3.8 Serial architecture for Haar wavelet (eight samples).

3.4 The Results of the two register-based architectures

Table 3.1 shows the architecture requirements and processing speed for the parallel architecture, and Table 3.2 shows the corresponding results for the serial architecture. The parallel architecture has many advantages over the serial architecture.
Although it works at a lower frequency, the parallel design processed data faster. This is because it finishes the whole process in four clock cycles, while the serial design takes 12 clock cycles to do the same job. The other advantage of the parallel architecture was the ability to have all the coefficients stored at a time, while two coefficients of each level were available in the other architecture. No controllers for the processors or the registers were needed in the parallel design, but in the serial one, there was a need of a controller for all the clock enable. However, the serial architecture used smaller number of IOBs and a little bit smaller number of CLBs. For designs of larger number of samples, these two architectures are not efficient. For this reason, the use of the FPGA’s user RAMs is the best choice available. Using RAMs in the architecture for the Haar wavelet processing can reduce the number of CLBs needed for store data bits by 87%. Chapter 4 shows this architecture and the details about the Dual-Port RAM in the XC4000 FPGAs.

Table 3.1 Hardware requirements and the performance of the parallel architecture.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of IOBs used</td>
<td>128</td>
</tr>
<tr>
<td>Number of CLBs used</td>
<td>120</td>
</tr>
<tr>
<td>CLB Flops/Latches</td>
<td>192</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>192</td>
</tr>
<tr>
<td>3 input LUTs</td>
<td>0</td>
</tr>
<tr>
<td>IO Flops/Latches</td>
<td>64</td>
</tr>
<tr>
<td>Maximum Frequency</td>
<td>68.743 MHz</td>
</tr>
<tr>
<td>Maximum net delay</td>
<td>7.164 ns</td>
</tr>
</tbody>
</table>
Table 3.2 Hardware requirements and the performance of the serial architecture.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of IOBs used</td>
<td>72</td>
</tr>
<tr>
<td>Number of CLBs used</td>
<td>98</td>
</tr>
<tr>
<td>Flops/Latches</td>
<td>168</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>112</td>
</tr>
<tr>
<td>3 input LUTs</td>
<td>28</td>
</tr>
<tr>
<td>Flops/Latches</td>
<td>8</td>
</tr>
<tr>
<td>Maximum Frequency</td>
<td>74.521 MHz</td>
</tr>
<tr>
<td>Maximum net delay</td>
<td>6.819 ns</td>
</tr>
</tbody>
</table>
Chapter 4

The Reconfigurable Architecture Using FPGA’s Dual-Port RAM

4.1 Using Dual-Port RAM

When we used the registers of the field programmable gate array, we found that the number of the configurable logic blocks (CLBs) used was large. Therefore, the registers seem to be a wrong choice. When comparing a CLB’s RAM with registers, we found that we can reduce the number of CLBs used by approximately 87% if we used RAM instead of the registers. Since Xilinx field programmable gate arrays have different RAM styles, we have to choose what helps improving the design in speed and logic cost. 32x1 and 16x1 single-port RAMs do not allow the read and write to be done in the same clock cycle. On the other hand, 16x1 dual-port RAM allows the read and write operations from and to different addresses at the same time along with the possibility of write and read to and from the same address simultaneously. Therefore, dual-port RAM was the best available choice for this type of design.

By using 16x1 dual-port RAM, we can make the number of samples per frame 16 instead of 8. Another level in the Haar wavelet block diagram is a result of this change. Figure 4.1(a) shows the block diagram of the Haar wavelet for eight samples while Figure 4.1(b) shows the Haar wavelet for 16 samples. From Figure 4.1, it is clear that one more level is the difference between the 16 samples and the 8 samples. The total number of coefficients is 32 and 80 for the eight and the 16 samples respectively. The whole design is discussed in detail in the next section.
Total number of coefficients $= 4 \times 8 = 32$

(a) Block diagram of the Haar wavelet for eight samples.

Total number of coefficients $= 5 \times 16 = 80$

(b) Block diagram of the Haar wavelet for 16 Samples.

Figure 4.1. Block diagrams of the Haar wavelet for the eight and the 16 samples.
4.2 The RAM Based Haar Wavelet Architecture

Figure 4.2 shows the simplified block diagram of the Haar wavelet architecture using the FPGA’s dual-port RAMs. The 16 samples are received in serial, with the least significant sample first, and are stored in the first RAM. After storing the data in this RAM, a processing element (PE) receives the two coefficients needed to compute the average and the difference. Then it writes the results in the next RAM. This process is done for all the RAM blocks and the processing elements. The controllers shown in Figure 4.2 control the read and write RAM addresses and the results of the processing elements.

Figure 4.2 Simplified block diagram of the Haar wavelet using dual-port RAM.

4.2.1 The Processing Elements

The processing elements compute the average and the difference of two signed 8bit numbers. These processors are designed to read one data per clock cycle. To make the computations of two numbers, a processor must have a register to hold the first
number. Then it computes the average and the difference at the same time. Another register is needed for storing the difference between these numbers. Therefore, the average is sent to the next RAM first. Then the difference is shifted out in the next clock cycle. These two steps are controlled by a multiplexer. Figure 4.3 shows the block diagram of a processing element (PE).

![Diagram of processing element](image)

Figure 4.3 Block diagram of processing element.

Figure 4.4 shows the block that computes the average. This block is implemented in the FPGA in the same way shown in Figure 3.5 without using the flip-flops. The average is computed by using an 8-bit adder and then by shifting the summation one bit to the right. The most significant bit of the result depends on the sign of the two numbers. Therefore, a simple two-level logic is added to this architecture to generate the sign of the result. The architecture for the block of the subtractor, and the register that follows, is
Figure 4.4 Block diagram shows the architecture that computes the average.
shown in Figure 3.6. One of the advantages of using XC4000 FPGAs can be shown here. The registers can be built in the same CLBs used for the subtractor. In other words, the registers added to the subtractors cost no additional CLBs.

### 4.2.2 The Controllers

The architecture shown in Figure 4.2 shows four controllers. These controllers play the main role in this design. They control the read addresses for previous RAMs and the write addresses for the next RAMs. Also, they control what result should be written to the next RAM by controlling the multiplexers. Therefore, these controllers are made carefully to insure the arrival of the control signals at the correct time. A simple state machine for one of the controllers is shown in Figure 4.5.

![State machine representation of CONT1](image)

Figure 4.5 State machine representation of the controller block "CONT1".
This controller is designed using a hardware description language called ABEL-HDL (or AHDL). The AHDL code used to synthesize the controllers is shown in Figure 4.6. Designing state machines using hardware description language is simpler than building them from the gate level. After the code is written, a synthesizer generates the logic implementation of the state machine. Therefore, designers save time by using hardware description languages.

```vhdl
module CONT1
Title 'CONT1' 
Declarations
C pin; " clock ";
START pin;
RESET pin;
M pin istype 'com'; " select to write result of addition or subtraction"
WE pin istype 'com'; " write enable for next RAM ";
RA..RA0 pin istype 'reg'; " read address of prev. RAM ";
WA..WA0 pin istype 'reg'; " write address of next RAM ";
RA=[RA3,RA2,RA1,RA0];
WA=[WA3,WA2,WA1,WA0];
Q3..Q0 node istype 'reg'; SM=[Q3,Q2,Q1,Q0];
S0=0; S1=3; S2=5; S3=9;
Equations
WA.clk=C;
RA.clk=C;
SM.clk=C;
WA.AR=RESET;
RA.AR=RESET;
SM.AR=RESET;
State_Diagram SM
State S0:
RA:=0; WA:=15;
WE=0; M=0;
IF START THEN S1 ELSE S0;
State S1:
RA:=RA+1; WA:=WA+1;
WE=0; M=0;
GOTO S2;
State S2:
RA:=RA+1; WA:=WA+1;
WE=1; M=0;
GOTO S3;
State S3:RA:=
RA+1; WA:=WA+1;
WE=1; M=1;
IF(WA==15) THEN S0 ELSE S2;
end CONT1
```

Figure 4.6 The AHDL code of the controller block " CONT1".
4.2.3 Comparison between the RAM-based and the parallel architectures

To compare this RAM-based design, I modified the parallel architecture to process 16 samples instead of 8. Table 4.1 shows the architecture requirements and processing speed for the parallel architecture and the corresponding results for the RAM-based architecture. The RAM-based architecture uses smaller area of the FPGA while the parallel architecture shows the opposite. The most important point is that the parallel architecture uses 256 IOBs, maximum number of IOBs in larger chips is 560. This restricts the implementation of the parallel architecture using FPGAs for larger number of samples. However, the RAM-based architecture shows an efficient implementation and then can be used to build the reconfigurable architecture, which, in addition to implementing the Haar transform, selects the best coefficients and then reprocess the data.

Table 4.1 Comparison between the parallel and the RAM-based architectures.

<table>
<thead>
<tr>
<th></th>
<th>Parallel Architecture</th>
<th>RAM-based Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of IOBs used</td>
<td>256</td>
<td>16</td>
</tr>
<tr>
<td>Number of CLBs used</td>
<td>320</td>
<td>136</td>
</tr>
<tr>
<td>Adders</td>
<td>32</td>
<td>4</td>
</tr>
<tr>
<td>Subtractors</td>
<td>32</td>
<td>4</td>
</tr>
<tr>
<td>Registers</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>Clock cycles needed</td>
<td>5</td>
<td>35</td>
</tr>
<tr>
<td>Maximum Frequency</td>
<td>21.45 MHz</td>
<td>33.57 MHz</td>
</tr>
</tbody>
</table>
4.3 Selecting coefficients

Using the algorithm described in Section 3.2, the coefficients that give the maximum information need to be selected. These selected coefficients are then fed back to the Haar architecture to be reprocessed. Repeating this step three times, as indicated in Section 3.2, requires storing the addresses of these coefficients in ROMs. Figure 4.7 shows a block diagram of the top-level design of the reconfigurable architecture of the algorithm. Once the Haar wavelet process is done, the first ROM gives the addresses for selecting the coefficients. After reprocessing these coefficients, new coefficients are selected using the addresses in the second ROM. This step is repeated one more time, but this time with the addresses stored in the last ROM.

Figure 4.7 Simplified block diagram of the system.
From Figure 4.7, we need to have some multiplexers and controllers to control all the addresses and the flow of data through the design. First, a multiplexer is needed to choose a direct input or the fed-back data. Another multiplexer is needed to choose one of the five RAMs when we read a selected coefficient. Four other multiplexers are used to control the read addresses of the first four RAMs. Two multiplexers are added to choose between the ROMs. A complete architecture, with all the multiplexers, is shown in Figure 4.8. Tables 4.2 and 4.3 show the addresses of the selected coefficients in each step and the data stored in the three ROM blocks. These addresses are divided into two groups. First three bits are for choosing which one of the five RAM blocks is selected. The other four bits are for choosing which one of the 16 memory cells in the selected RAM is selected. The RAM address is obtained as the integer value that results from division of the coefficient number by 16 and the reminder represents the memory cell address. For instance, if the entropy based-algorithm selected coefficient number 24, then 24 by 16 equals 1 and the reminder equals 8. This means that the address of this coefficient has the RAM address S=1 (001)_b (RAM number 2) and the cell address is A=8 (1000)_b (S is any number from 0 to 4 and A is any number from 0 to 15).

One more RAM block is needed for temporarily storing the selected coefficients. After writing all the selected data in this temporary RAM, a new iteration of the Haar wavelet can be started. The complex problem in this design is to design the controllers and to make sure that each data written or read are the correct ones and that they are read (or written) from (or to) the correct memory location at the correct time. Dividing the controller to small controllers simplifies the design. Thus, Part1, Part2, ROM-Cont, and
Main are the controllers needed to control this reconfigurable architecture. All these controllers are written in AHDL to simplify the design. Figures 4.9 shows the code for the Main block. Refer to Appendix B for the codes of the rest of the controllers.

Figure 4.8 The whole system (some control signals are not shown).
Table 4.2 The selected coefficients and their locations.

<table>
<thead>
<tr>
<th>The Selected Coefficients</th>
<th>After Iteration #1</th>
<th>After Iteration #2</th>
<th>After Iteration #3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>68,52,51,36,35,24, 22,21,20,15,10,9,8, 55,11,18</td>
<td>80,76,75,74,72,67, 66,64,63,59,58,57, 56,55,53,51</td>
<td>80,79,78,77,76,75, 73,72,69,68,67,65, 63,62,59,58</td>
</tr>
<tr>
<td>Addresses for the selected coefficients are stored In</td>
<td>ROM#1</td>
<td>ROM#2</td>
<td>ROM#3</td>
</tr>
</tbody>
</table>

Table 4.3 The data stored in the three ROM blocks.

<table>
<thead>
<tr>
<th>Location</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM #1</td>
<td>S</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>14</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>ROM #2</td>
<td>S</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>15</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>14</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>ROM #3</td>
<td>S</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>14</td>
<td>13</td>
<td>10</td>
</tr>
</tbody>
</table>
module Main
Title 'Main'

Declarations

CLK PIN;
RESET PIN;
START PIN;

SL1..SL0 PIN ISTYPE 'COM';
SL = [SL1..SL0];
Q3..Q0 NODE ISTYPE 'REG';
SM=[Q3,Q2,Q1,Q0];
S0=0; S1=3; S2=5;S3=9;

Equations

SM.CLK=CLK;
SM.CLK=CLK;

STATE_DIAGRAM SM
STATE S0:          SL=0;
                   IF START THEN S1 ELSE S0;
STATE S1:          SL=0;
                   IF START THEN S2 ELSE S1;
STATE S2:          SL=1;
                   IF START THEN S3 ELSE S2;
STATE S3:          SL=2;
                   IF START THEN S0 ELSE S3;

end Main

Figure 4.9 The AHDL code for the controller "Main".

The FPGA layout of the implemented reconfigurable RAM-based architecture is shown in Figure 4.10. It is clear, by looking to Figure 4.10, that the design occupied around 50% of the XC4010 chip. Thus this chip can be programmed to perform larger
circuit (i.e. 32 samples). 128-samples reconfigurable architecture can be implemented using larger FPGA such as the XC4052.

Figure 4.10 The layout of the implemented reconfigurable RAM-based architecture.
Chapter 5
Simulation Results, Discussion and Design Recommendations

This chapter presents the simulation results using the Matlab and the FPGA design tools. Architecture requirements and performance will also be presented along with a discussion of the results. The last section, design recommendations, includes comparisons between the three developed architectures.

5.1 Simulation

For the input signal shown below and after running the Matlab programs the generated matrix of coefficients and the coefficients that need to be selected are shown in Table 5.1.

\[
\text{input signal = \{1, 12, 22, 23, 48, 52, 36, 29, 22, 13, 1, -2, -8, -16, -21, -9\}}
\]

For the same input signal, part of the functional simulation of the RAM-based architecture, using Xilinx Project Manager, is shown in Figure 5.1. Simulation results that start with the input signal being loaded to the first RAM identified as DATA_IN is shown in Appendix C for the consecutive 22.4us. Using the whole waveform results, Table 5.2 is generated. For instance, the results after completion of the first iteration are shown in Figure 5.1 with RAM1DATA which correspond to the first level values from Table 5.2 through RAM4DATA which correspond to the fourth level values. The values shown in the waveform (Figure 5.1) are represented in hexadecimal while these values are translated to decimal to create Table 5.2. For example, the value of RAM1DATA, at
time=1.15 us, is \( (0C)h \), 0C in hexadecimal equals 12 in decimal which is the second value shown in Table 5.1. Addresses of the selected coefficients (shown in boldface) translated to S and A values are represented by RAM_SEL and LUT_RA signals. The selected coefficient numbers are shown in the last column of Table 5.1. For instance, the first selected coefficient C67 is obtained from the fifth level transform and its value in Table 5.1 is -49. Comparing these simulation results, it seems that the results in both simulations are almost the same.

![Figure 5.1 Part of the waveform results of the functional simulation.](image-url)
The differences come from the fact that dividing binary numbers by using the right shift operation ignores the fraction of numbers. For example, 13/2 in binary is rounded to 6 in the Haar wavelet hardware, while the result is kept as 6.5 in Matlab. After reprocessing numbers with small errors many times, the error can increase and then be considered as a large error. However, the results were approximately similar to each other. In order to avoid the numerical discrepancy between Matlab and the developed architecture, Matlab code was modified by using the floor function. The corrected results exactly match the results obtained by simulation of the designed architecture.

5.2 The architecture's FPGA requirements and the performance

The RAM-based architecture developed in Chapter 4 has been mapped to a Xilinx XC4010XL FPGA. This reconfigurable architecture used 20 input and output blocks (IOBs). This is because the design receives an 8bit number, a reset, a start and a clock. Also, it gives an 8bit number and a DONE signal. The total number of configurable logic blocks (CLBs) used was 212 out of 400. These CLBs were used to implement the Haar wavelet and the process of selecting coefficients. The Maximum frequency with which the architecture can process data was 17.70 MHz. Also, the whole process takes about 220 clock cycles. This means that we can process new samples approximately every 11 us. In other words, the architecture can process the real time data if they are received in no more than 88 KHz.
Table 5.1 The generated coefficients after the matlab simulation.

<table>
<thead>
<tr>
<th>First Level Values of (C0-C15)</th>
<th>Second Level Values of (C16-C31)</th>
<th>Third Level Values of (C32-C47)</th>
<th>Fourth Level Values of (C48-C63)</th>
<th>Fifth Level Values of (C64-C79)</th>
<th>The Selected Coefficients For Next Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Iteration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>15</td>
<td>28</td>
<td>13</td>
<td>C67</td>
</tr>
<tr>
<td>12</td>
<td>23</td>
<td>41</td>
<td>2</td>
<td>31</td>
<td>C51</td>
</tr>
<tr>
<td>22</td>
<td>50</td>
<td>8</td>
<td>-27</td>
<td>-3</td>
<td>C50</td>
</tr>
<tr>
<td>23</td>
<td>33</td>
<td>-13</td>
<td>22</td>
<td>-49</td>
<td>C35</td>
</tr>
<tr>
<td>48</td>
<td>17</td>
<td>-16</td>
<td>1</td>
<td>6</td>
<td>C34</td>
</tr>
<tr>
<td>52</td>
<td>-1</td>
<td>18</td>
<td>11</td>
<td>-10</td>
<td>C23</td>
</tr>
<tr>
<td>36</td>
<td>-12</td>
<td>18</td>
<td>-34</td>
<td>-9</td>
<td>C21</td>
</tr>
<tr>
<td>29</td>
<td>-15</td>
<td>3</td>
<td>15</td>
<td>-48</td>
<td>C20</td>
</tr>
<tr>
<td>22</td>
<td>-12</td>
<td>6</td>
<td>-2</td>
<td>0</td>
<td>C19</td>
</tr>
<tr>
<td>13</td>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>-4</td>
<td>C14</td>
</tr>
<tr>
<td>1</td>
<td>-4</td>
<td>6</td>
<td>-7</td>
<td>0</td>
<td>C9</td>
</tr>
<tr>
<td>-2</td>
<td>7</td>
<td>-2</td>
<td>7</td>
<td>-15</td>
<td>C8</td>
</tr>
<tr>
<td>-8</td>
<td>9</td>
<td>-11</td>
<td>-11</td>
<td>1</td>
<td>C7</td>
</tr>
<tr>
<td>-16</td>
<td>3</td>
<td>-10</td>
<td>13</td>
<td>-23</td>
<td>C54</td>
</tr>
<tr>
<td>-21</td>
<td>8</td>
<td>6</td>
<td>-1</td>
<td>-7</td>
<td>C10</td>
</tr>
<tr>
<td>-9</td>
<td>-12</td>
<td>20</td>
<td>-14</td>
<td>14</td>
<td>C17</td>
</tr>
<tr>
<td>Second Iteration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-49</td>
<td>-13</td>
<td>-17</td>
<td>-7</td>
<td>1</td>
<td>C79</td>
</tr>
<tr>
<td>22</td>
<td>-20</td>
<td>3</td>
<td>8</td>
<td>-15</td>
<td>C75</td>
</tr>
<tr>
<td>-27</td>
<td>-3</td>
<td>12</td>
<td>-19</td>
<td>-6</td>
<td>C74</td>
</tr>
<tr>
<td>-13</td>
<td>8</td>
<td>5</td>
<td>7</td>
<td>-26</td>
<td>C73</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>7</td>
<td>-3</td>
<td>-8</td>
<td>C71</td>
</tr>
<tr>
<td>-15</td>
<td>17</td>
<td>-12</td>
<td>-13</td>
<td>10</td>
<td>C66</td>
</tr>
<tr>
<td>-1</td>
<td>-2</td>
<td>-11</td>
<td>18</td>
<td>10</td>
<td>C65</td>
</tr>
<tr>
<td>17</td>
<td>12</td>
<td>-14</td>
<td>2</td>
<td>16</td>
<td>C63</td>
</tr>
<tr>
<td>33</td>
<td>-70</td>
<td>-42</td>
<td>-20</td>
<td>1</td>
<td>C62</td>
</tr>
<tr>
<td>-21</td>
<td>-13</td>
<td>3</td>
<td>21</td>
<td>-41</td>
<td>C58</td>
</tr>
<tr>
<td>13</td>
<td>23</td>
<td>22</td>
<td>-45</td>
<td>-21</td>
<td>C57</td>
</tr>
<tr>
<td>22</td>
<td>-18</td>
<td>21</td>
<td>2</td>
<td>-46</td>
<td>C56</td>
</tr>
<tr>
<td>29</td>
<td>53</td>
<td>-57</td>
<td>-8</td>
<td>33</td>
<td>C55</td>
</tr>
<tr>
<td>-34</td>
<td>-9</td>
<td>41</td>
<td>73</td>
<td>-81</td>
<td>C54</td>
</tr>
<tr>
<td>1</td>
<td>63</td>
<td>62</td>
<td>-98</td>
<td>-60</td>
<td>C52</td>
</tr>
<tr>
<td>23</td>
<td>-22</td>
<td>85</td>
<td>-23</td>
<td>-76</td>
<td>C50</td>
</tr>
<tr>
<td>Third Iteration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-76</td>
<td>-61</td>
<td>-46</td>
<td>-27</td>
<td>-22</td>
<td>C79</td>
</tr>
<tr>
<td>-46</td>
<td>-31</td>
<td>-7</td>
<td>-18</td>
<td>-9</td>
<td>C78</td>
</tr>
<tr>
<td>-21</td>
<td>5</td>
<td>-35</td>
<td>-39</td>
<td>-37</td>
<td>C77</td>
</tr>
<tr>
<td>-41</td>
<td>-19</td>
<td>0</td>
<td>-35</td>
<td>-4</td>
<td>C76</td>
</tr>
<tr>
<td>16</td>
<td>-71</td>
<td>-30</td>
<td>-3</td>
<td>-14</td>
<td>C75</td>
</tr>
<tr>
<td>-6</td>
<td>1</td>
<td>24</td>
<td>-26</td>
<td>23</td>
<td>C74</td>
</tr>
<tr>
<td>-15</td>
<td>10</td>
<td>-72</td>
<td>-54</td>
<td>-74</td>
<td>C72</td>
</tr>
<tr>
<td>-23</td>
<td>-11</td>
<td>21</td>
<td>-94</td>
<td>40</td>
<td>C70</td>
</tr>
<tr>
<td>-98</td>
<td>-29</td>
<td>-5</td>
<td>5</td>
<td>1</td>
<td>C68</td>
</tr>
<tr>
<td>-45</td>
<td>20</td>
<td>15</td>
<td>-3</td>
<td>8</td>
<td>C67</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>-6</td>
<td>-20</td>
<td>-13</td>
<td>C66</td>
</tr>
<tr>
<td>-20</td>
<td>7</td>
<td>0</td>
<td>-7</td>
<td>-13</td>
<td>C64</td>
</tr>
<tr>
<td>2</td>
<td>-54</td>
<td>-49</td>
<td>-17</td>
<td>-40</td>
<td>C62</td>
</tr>
<tr>
<td>18</td>
<td>41</td>
<td>15</td>
<td>-64</td>
<td>47</td>
<td>C61</td>
</tr>
<tr>
<td>-3</td>
<td>-16</td>
<td>-95</td>
<td>-64</td>
<td>-63</td>
<td>C58</td>
</tr>
<tr>
<td>-19</td>
<td>17</td>
<td>-33</td>
<td>-62</td>
<td>-2</td>
<td>C57</td>
</tr>
</tbody>
</table>
Table 5.2 The generated coefficients after simulating the FPGA's architecture.

<table>
<thead>
<tr>
<th></th>
<th>First Level Values of (C0-C15)</th>
<th>Second Level Values of (C16-C31)</th>
<th>Third Level Values of (C32-C47)</th>
<th>Fourth Level Values of (C48-C63)</th>
<th>Fifth Level Values of (C64-C79)</th>
<th>The Selected Coefficients For Next Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First Iteration</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
<td>14</td>
<td>27</td>
<td>12</td>
<td>C67</td>
</tr>
<tr>
<td>12</td>
<td>22</td>
<td>41</td>
<td>-3</td>
<td>-27</td>
<td>30</td>
<td>C51</td>
</tr>
<tr>
<td>22</td>
<td>50</td>
<td>8</td>
<td>-14</td>
<td>22</td>
<td>-3</td>
<td>C50</td>
</tr>
<tr>
<td>23</td>
<td>32</td>
<td>-18</td>
<td>10</td>
<td>-34</td>
<td>-9</td>
<td>C35</td>
</tr>
<tr>
<td>48</td>
<td>17</td>
<td>-16</td>
<td>15</td>
<td>-49</td>
<td>-10</td>
<td>C34</td>
</tr>
<tr>
<td>52</td>
<td>-1</td>
<td>18</td>
<td>8</td>
<td>-27</td>
<td>5</td>
<td>C23</td>
</tr>
<tr>
<td>36</td>
<td>-12</td>
<td>18</td>
<td>-34</td>
<td>15</td>
<td>-49</td>
<td>C21</td>
</tr>
<tr>
<td>29</td>
<td>-15</td>
<td>3</td>
<td>-3</td>
<td>-1</td>
<td>-5</td>
<td>C20</td>
</tr>
<tr>
<td>22</td>
<td>-11</td>
<td>1</td>
<td>-7</td>
<td>-1</td>
<td>1</td>
<td>C19</td>
</tr>
<tr>
<td>13</td>
<td>-1</td>
<td>2</td>
<td>-7</td>
<td>-1</td>
<td>-5</td>
<td>C14</td>
</tr>
<tr>
<td>1</td>
<td>-4</td>
<td>6</td>
<td>8</td>
<td>-15</td>
<td>C8</td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>7</td>
<td>-2</td>
<td>10</td>
<td>C9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-8</td>
<td>9</td>
<td>10</td>
<td>12</td>
<td>C7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-16</td>
<td>3</td>
<td>-11</td>
<td>24</td>
<td>C65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-21</td>
<td>8</td>
<td>6</td>
<td>-7</td>
<td>C10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-9</td>
<td>-12</td>
<td>20</td>
<td>15</td>
<td>C17</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Second Iteration</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-49</td>
<td>-14</td>
<td>-18</td>
<td>-8</td>
<td>-1</td>
<td>C79</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>-21</td>
<td>2</td>
<td>7</td>
<td>-15</td>
<td>C75</td>
<td></td>
</tr>
<tr>
<td>-27</td>
<td>-4</td>
<td>11</td>
<td>-20</td>
<td>-7</td>
<td>C74</td>
<td></td>
</tr>
<tr>
<td>-14</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>-27</td>
<td>C73</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>7</td>
<td>-3</td>
<td>-8</td>
<td>C71</td>
<td></td>
</tr>
<tr>
<td>-15</td>
<td>17</td>
<td>-12</td>
<td>-13</td>
<td>10</td>
<td>C66</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>-3</td>
<td>-12</td>
<td>19</td>
<td>10</td>
<td>C65</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>-14</td>
<td>2</td>
<td>17</td>
<td>C63</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>-71</td>
<td>-42</td>
<td>-20</td>
<td>0</td>
<td>C62</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>-12</td>
<td>2</td>
<td>21</td>
<td>-41</td>
<td>C58</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>23</td>
<td>22</td>
<td>-44</td>
<td>-22</td>
<td>C57</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>53</td>
<td>-58</td>
<td>-9</td>
<td>32</td>
<td>C55</td>
<td></td>
</tr>
<tr>
<td>-34</td>
<td>-9</td>
<td>41</td>
<td>73</td>
<td>-82</td>
<td>C54</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>63</td>
<td>62</td>
<td>-99</td>
<td>-61</td>
<td>C52</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>-21</td>
<td>84</td>
<td>-22</td>
<td>-77</td>
<td>C50</td>
<td></td>
</tr>
<tr>
<td><strong>Third Iteration</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-77</td>
<td>-61</td>
<td>-47</td>
<td>-27</td>
<td>-23</td>
<td>C79</td>
<td></td>
</tr>
<tr>
<td>-45</td>
<td>-32</td>
<td>-7</td>
<td>-19</td>
<td>-8</td>
<td>C78</td>
<td></td>
</tr>
<tr>
<td>-22</td>
<td>5</td>
<td>-36</td>
<td>-40</td>
<td>-38</td>
<td>C77</td>
<td></td>
</tr>
<tr>
<td>-41</td>
<td>-19</td>
<td>-1</td>
<td>-35</td>
<td>-5</td>
<td>C76</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>-72</td>
<td>-29</td>
<td>-3</td>
<td>-14</td>
<td>C75</td>
<td></td>
</tr>
<tr>
<td>-7</td>
<td>0</td>
<td>24</td>
<td>-25</td>
<td>22</td>
<td>C74</td>
<td></td>
</tr>
<tr>
<td>-15</td>
<td>10</td>
<td>-72</td>
<td>-53</td>
<td>-74</td>
<td>C72</td>
<td></td>
</tr>
<tr>
<td>-22</td>
<td>-12</td>
<td>22</td>
<td>-94</td>
<td>41</td>
<td>C70</td>
<td></td>
</tr>
<tr>
<td>-99</td>
<td>-32</td>
<td>-7</td>
<td>4</td>
<td>0</td>
<td>C68</td>
<td></td>
</tr>
<tr>
<td>-44</td>
<td>19</td>
<td>15</td>
<td>4</td>
<td>8</td>
<td>C67</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>24</td>
<td>-7</td>
<td>22</td>
<td>-15</td>
<td>C66</td>
<td></td>
</tr>
<tr>
<td>-20</td>
<td>7</td>
<td>0</td>
<td>-7</td>
<td>-15</td>
<td>C64</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-55</td>
<td>-51</td>
<td>-17</td>
<td>-41</td>
<td>C62</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>41</td>
<td>17</td>
<td>-65</td>
<td>48</td>
<td>C61</td>
<td></td>
</tr>
<tr>
<td>-3</td>
<td>-17</td>
<td>-96</td>
<td>-68</td>
<td>-65</td>
<td>C58</td>
<td></td>
</tr>
<tr>
<td>-20</td>
<td>17</td>
<td>-34</td>
<td>-62</td>
<td>-6</td>
<td>C57</td>
<td></td>
</tr>
</tbody>
</table>
Each RAM and ROM that was used in the design was constructed from eight CLBs. One CLB is configured as 16x1 RAM or ROM; therefore, eight CLBs implement 16x8 RAM or ROM. The simplified architecture of these memory devices is shown in Figure 5.2. Each Processing element contains an adder with a divider by two, a subtractor, and a register. To compute the average of two 8bit numbers, the adder adds the two numbers and then divides the result by two (shift the results one bit to the right). Five CLBs implementing this operation are shown in Figure 5.3. Also, the architecture of the subtractor was built of five CLBs as shown in Chapter 3 in Figure 3.6. However, the 8bit register was built using only four CLBs.

Figure 5.2 Architecture of 16x8 RAM (eight CLBs are needed).
Figure 5.3 Architecture of the block that computes the average.
The last parts of the design are the controllers and the multiplexers. First, the multiplexers are implemented as shown in Figure 5.4. Each 2x1 2-bit Multiplexer can be mapped into one CLB. Thus, four CLBs can build a 2x1 8-bit multiplexer.

Figure 5.4 Architecture of a 2x1 MUX (2-bit MUX per CLB).

On the other hand, the controllers of this design used different numbers of CLBs. First, in all the state machines of the controllers, a one-hot encoding is used. Because the number of allowed inputs per function generator is limited, the one-hot encoding was chosen. Therefore, it can be said that each CLB implements two states of a state machine because
it contains two flip-flops. Some of these controllers have counters for the addresses of the RAMs or the ROMs. Each 2-bit counter can be mapped in one CLB. Therefore, a 4-bit counter can be mapped into two CLBs. Finally, designers can estimate the number of configurable logic blocks (CLBs) needed. This estimation is not accurate because it does not include the CLBs used to ease the routings. However, this estimation can be multiplied by 130% to be on the safe side.

5.3 Design Recommendations

This thesis is using one-dimensional Haar wavelet algorithm for pattern recognition applications. Applying the algorithm to a Xilinx XC4000 FPGA was done in three different architectures: parallel, serial and RAM-based. Each architecture has some advantages over the others. First, the parallel architecture is the only architecture that can process a continuous flow of data. Therefore, it can be applied to different pattern recognition applications. A very small number of cycles needed to finish a process seems to indicate that the parallel architecture is the best architecture. However, the parallel architecture needs a large number of adders and subtractors. Thus, the number of CLBs needed is very high. This is clearly seen in Table 3.1. The worst disadvantage of using the parallel design is the very large number of input and output blocks (IOBs). 128 IOBs were used in eight data samples (each is 8-bit). For this reason it is too costly to use the parallel architecture because it requires a large number of different FPGA resources. Second, the serial design was developed to reduce the required resources. It requires a smaller number of CLBs compared to the parallel architecture. The number of the CLBs
was reduced to 98 instead of 120 CLBs in the parallel design (see Tables 3.1 and 3.2). This reduction was not enough to make the serial architecture a winner. However, the number of IOBs used in the serial architecture was reduced to 51% of what the parallel architecture used. The frequency is almost the same, while the number of cycles needed to finish the process is doubled. Third, the RAM-based design was developed. This architecture used 212 configurable logic blocks, but these CLBs are used to develop the whole process (Haar wavelet and selecting the coefficients). Also, 16 samples are fed in this design instead of eight. With these modifications, the number of CLBs used was acceptable. The number of input and output blocks was also small. Although the design processes 16 data samples in almost 200 cycles with a low frequency, the rate of processing data is still acceptable. This architecture explores most of the features offered by FPGAs. Using registers, arithmetic operations, RAM, ROM, multiplexers, buffers and state machines in one chip, support the one-chip solution. That is the most important advantage of using FPGAs. However, this architecture depends on the results generated by the Matlab program. Thus, the locations of the selected coefficients are known before the architecture is built. In particular, the addresses information of the selected coefficients in each iteration is pre-stored in ROMs during the configuration stage. This architecture may be improved if, instead of using the XC4000 FPGAs, a reconfigurable FPGA, also called reconfigurable processing units (RPU), is used (i.e. XC6200). Finally, it can be concluded that the three architectures can be used, but with some assumptions and limitations.
6.1 Summary

This thesis developed different architectures that implement a one-dimensional Haar wavelet using FPGAs. These FPGA-based Haar wavelet architectures were developed using different styles and FPGA resources. For instance, parallel and serial architectures were used with FPGA registers. FPGA RAM and ROM resources were the essential part of the other architecture. With the existence of a neural network (i.e. classifier) that reads the information out of this architecture, many pattern recognition problems can be addressed. In summary, the thesis started by showing the basic architecture of a SRAM field programmable gate array. A brief description of the main architecture of the Xilinx XC4000 FPGA is presented in Chapter 2. The building blocks of an FPGA are shown also. Then a process based on a Haar wavelet is described in Chapter 3. Three different architectures using FPGAs are presented in detail in Chapters 3 and 4. The requirements and performances of these architectures are given and discussed in detail in Chapter 5. All of these architectures can be considered from the point of view of efficient utilization of FPGA resources, size of the mapped architecture and the speed, and the optimum architecture for a specific application selected.
6.2 Future Work

There are many modifications that can be made in future. Some of these future works are discussed in this section. First, 16 data samples can be increased to some applicable number such as 128. Another modification under the same point is to increase the number of bits per sample from eight to 12 or 16. These increases make the design applicable to many current pattern recognition problems. Second, an FPGA board that can communicate to a personal computer through the PCI bus can be a good topic for future work. This helps to process data coming from outside (i.e. ADC) or from inside the PC and then send the results through the PCI bus to the system. Third, to make a real time reconfigurable architecture the XC6200 can be the best choice. These XC6200 partially reconfigurable FPGAs, also called reconfigurable processing units (RPU), can do the whole process and do the calculations for selecting the coefficients. Writing Matlab codes is not required because the chip does it all. Also, designing a board for these chips can be a next step. Fourth, architecture for 2-dimensional (or higher) Haar wavelet can be a good step after this thesis. Developing a neural network that takes the processed data from the current architecture and completes the classifier or the recognition part of the design can be interesting and makes the design applicable. Finally, there are many future works that can be thought of after this thesis. These can be modifications of the architecture shown or pattern recognition applications that can use this algorithm.
References


Appendix A

The Matlab Codes

A.1 The Main Matlab Program:

clear
clc

iterations=1;

absinfoincr=.005;

fidout=fopen('proc16ou.ddd','w');

fidin =fopen('proc16in.ddd','w');

% define signals here.

signal1=[0, 20, 33, 40, 57, 38, 12, 1, -13, -20, -24, -19, -10, 0, 15, 7];

signal2=[0, 9, 17, 22, 43, 50, 32, 28, 20, 11, 0, -4, -10, -21, -23, -12];

fprintf(fidin,'Input signals are:
');

fprintf(fidin,' %3.0f
', signal1);

fprintf(fidin,' %3.0f
', signal2);

numCoeff=length(signal1);

sig1noiselevel=6;

sig2noiselevel=6;

currinf=1;

previnf=0;

%stringmat=['r-';'b-';'g-';'r-';'m-';'y-';'b-';'g-';'r-';'m-'];
for i=1:50
    Matrix(:,i)=signal1+sig1noiselevel*rand(numCoeff,1);
end;
for i=51:100
    Matrix(:,i)=signal2+sig2noiselevel*rand(numCoeff,1);
end;
fprintf(fidin,'The First Noisy signal is:
');
fprintf(fidin,' %3.0f
',Matrix(:,1));
fprintf(fidin,'The Second Noisy signal is:
');
fprintf(fidin,' %3.0f
',Matrix(:,51));
levls=round(log10(numCoeff)/log10(2))
x=1:numCoeff
% while abs(currinf-previnf)>absinfoincr
while iterations<=3
fprintf(fidout,'Iteration Number = %i\n',iterations);

rows=numCoeff;

D=Matrix;

previnf=currinf;

for i=1:levls

C=[];

begn=1;

while begin<numCoeff

oddr=(begn:2:begn+rows-1);
evnr=(begn+1:2:begn+rows-1);

A=(Matrix(oddr,:)+Matrix(evnr,:))/2;

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;

for i=1:(numCoeff*(levls+1))
if Matrix(i,51)>127 \& Matrix(i,51)<(-127)
    fprintf(1,' Not a good data\n');
end; %if
end; %for

% fprintf(fidout,'For The First Noisy signal, the Matrix becomes:\n');
% fprintf(fidout,' %3.0f\n',Matrix(:,1));
fprintf(fidout,'For The Second Noisy signal, the Matrix becomes:\n');
fprintf(fidout,' %3.0f\n',Matrix(:,51));
[rowsMatrix colsMatrix] = size(Matrix);
% make other matrices to ease 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);
currinf = orderedinf(1);
index=fliplr(index);

% reset the Matrix before going into the other loop.
rownum = [index(1:numCoeff)];
Matrix=Matrix(rownum,:);
iterations=iterations+1;

fprintf(fidout,' The chosen addresses(indices) are:\n');
fprintf(fidout,'%2.0f \n',index(1:numCoeff));
end; %while iterations
fclose all;
A.2 The Matlab Code for infopdfV Function:

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;
pplotPdf2=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;
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(casel)*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;
% evaluate weighted probabilities
P1w=sum(pdf1x').*step;
P2w=sum(pdf2x').*step;

% find the entropy measure
entr = -(P1.*log(P1)+P2.*log(P2)) + P1w.*log(P1w)+P2w.*log(P2w);

% find information
P11=P1.^2.//(P1+P2);
P22=P2.^2.//(P1+P2);
maxentr=-(P1.*log(P1)+P2.*log(P2))+P11.*log(P11)+P22.*log(P22);
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;
Appendix B

The AHDL Codes For the Controllers

B.1 The AHDL Code for Main Block:

module Main
Title 'Main'

Declarations

CLK PIN;
RESET PIN;
START PIN;
DONE PIN ISTYPE 'COM'
SL1..SL0 PIN ISTYPE 'COM';
SL = [SL1..SL0];
Q3..Q0 NODE ISTYPE 'REG';
SM=[Q3,Q2,Q1,Q0];
S0=0; S1=3; S2=5;S3=9;

Equations

SM.CLK=CLK;
SM.CLK=CLK;

STATE_DIAGRAM SM

STATE S0: SL=0;

IF START THEN S1 ELSE S0;
STATE S1: SL=0;
    IF START THEN S2 ELSE S1;
STATE S2: SL=1;
    IF START THEN S3 ELSE S2;
STATE S3: SL=2;
    IF START THEN
        DONE=1; GOTO S0 ELSE S3; END;
end Main

B.2 The AHDL Code for the Part1 Block:

module PART1
Title 'PART1'
Declarations
    CLK PIN; "CLOCK"
    START PIN;
    RESET PIN;
    SX PIN ISTYPE 'COM';
    RA3..RA0 PIN ISTYPE 'REG';
    RA=[RA3,RA2,RA1,RA0];
    Q2..Q0NODE ISTYPE 'REG';
    SM=[Q2,Q1,Q0];
S0=0; S1=3; S2=5;

Equations
RA.CLK=CLK;
RA.AR=RESET;
SM.CLK=CLK;
SM.AR=RESET;

STATE DIAGRAM SM

STATE S0: RA:=0;
SX=0;
IF START THEN S1 ELSE S0;

STATE S1: RA:=RA+1;
SX=1;
GOTO S2;

STATE S2: RA:=RA+1;
SX=1;
IF (RA==15) THEN S0 ELSE S1;

end PART1

B.3 The AHDL Code for the Part2 Block:

module PART2
Title 'PART2'

Declarations
CLK pin; "clock"

START pin;

RESET pin;

WE pin istype 'com'; "write enable for FIRST RAM"

WA3..WA0 pin istype 'reg'; "write address of FIRST RAM"

WA=[WA3,WA2,WA1,WA0];

Q2..Q0 node istype 'reg';

SM=[Q2,Q1,Q0];

S0=0; S1=3; S2=5;

Equations

WA.clk=CLK;

WA.AR=RESET;

SM.clk=CLK;

SM.AR=RESET;

State Diagram SM

State S0: WA:=0;

WE=0;

IF START then S1 else S0;

State S1: WA:=WA+1;

WE=1;

GOTO S2;

State S2: WA:=WA+1;
WE=1;
IF(WA==15) THEN S0 ELSE S1;
end PART2

B.4 The AHDL Code for the ROM_CONT Block:

module ROM_CONT
Title 'ROM_CONT'
Declarations
START PIN;
RESET PIN;
CLK PIN;
SR PIN ISTYPE 'COM';
RA3..RA0 PIN istype 'reg';
RA=[RA3,RA2,RA1,RA0];
Q1..Q0 NODE ISTYPE 'REG';
SM=[Q1,Q0];
S0=0;
S1=3;
Equations
   SM.CLK=CLK;
   SM.AR=RESET;
   RA.CLK=CLK;
RA.AR=RESET;

STATE_DIAGRAM SM

STATE S0: RA:=0;
       SR=0;
       IF START THEN S1 ELSE S0;

STATE S1: RA:=RA+1;
       SR=1;
       IF (RA==15) THEN S0 ELSE S1;

end ROM_CONT

B.5 The AHDL Code for the CONT1 Block:

module CONT1
Title 'CONT1'
Declarations
    C pin; " clock "
    START pin;
RESET pin;

M pin istype 'com'; " select to write result of addition or subtraction"

WE pin istype 'com'; " write enable for next RAM "

RA3..RA0    pin istype 'reg'; " read address of prev. RAM "

WA3..WA0    pin istype 'reg'; " write address of next RAM "

RA=[RA3,RA2,RA1,RA0];

WA=[WA3,WA2,WA1,WA0];

Q3..Q0node istype 'reg';

SM=[Q3,Q2,Q1,Q0];

S0=0; S1=3; S2=5; S3=9;

Equations

WA.clk=C;

RA.clk=C;

SM.clk=C;

WA.AR=RESET;

RA.AR=RESET;

SM.AR=RESET;

State_Diagram SM

State S0: RA:=0;

WA:=15;

WE=0;
M = 0;

IF START THEN S1 ELSE S0;

State S1:  RA := RA + 1;
          WA := WA + 1;
          WE = 0;
          M = 0;
          GOTO S2;

State S2:  RA := RA + 1;
          WA := WA + 1;
          WE = 1;
          M = 0;
          GOTO S3;

State S3:  RA := RA + 1;
          WA := WA + 1;
          WE = 1;
          M = 1;
          IF(WA == 15) THEN S0 ELSE S2;

end CONT1
B.6 The AHDL Code for the CONT2 Block:

module CONT2
Title 'CONT2'

Declarations

C pin; " clock "
START pin;
RESET pin;
M pin istype 'com'; " select to write result of addition or subtraction"
WE pin istype 'com'; " write enable for next RAM "
RA3..RA0 pin istype 'reg'; " read address of prev. RAM "
WA3..WA0 pin istype 'reg'; " write address of next RAM "
RA=[RA3,RA2,RA1,RA0];
WA=[WA3,WA2,WA1,WA0];
Q3..Q0node istype 'reg';
SM=[Q3,Q2,Q1,Q0];
S0=0; S1=3; S2=5; S3=9;

Equations

WA.clk=C;
RA.clk=C;
SM.clk=C;
WA.AR=RESET;
RA.AR=RESET;
SM.AR=RESET;

State Diagram SM

State S0:  RA:=0;
           WA:=15;
           WE=0;
           M=0;
           IF START then S1 else S0;

State S1:  RA:=RA+1;
           WA:=WA+1;
           WE=0;
           M=0;
           GOTO S2;

State S2:  RA:=RA+1;
           WA:=WA+1;
           WE=1;
           M=0;
           GOTO S3;

State S3:  RA:=RA+1;
           WA:=WA+1;
WE=1;
M=1;
IF(WA==15) THEN S0 ELSE S2;
end CONT2

B.7 The AHDL Code for the CONT3 Block:

module CONT3
Title 'CONT3'

Declarations
  C pin; "clock"
  START pin;
  RESET PIN;
  M pin istype 'com'; "select to write result of addition or subtraction"
  WE pin istype 'com'; "write enable for next RAM"
  RA3..RA0 pin istype 'reg'; "read address of prev. RAM"
  WA3..WA0 pin istype 'reg'; "write address of next RAM"
  RA=[RA3,RA2,RA1,RA0];
  WA=[WA3,WA2,WA1,WA0];
  Q3..Q0node istype 'reg';
\[ \text{SM} = \{Q3, Q2, Q1, Q0\}; \]
\[ S0 = 0; S1 = 3; S2 = 5; S3 = 9; \]

Equations

\[ \text{WA.clk} = C; \]
\[ \text{RA.clk} = C; \]
\[ \text{SM.clk} = C; \]
\[ \text{WA.AR} = \text{RESET}; \]
\[ \text{RA.AR} = \text{RESET}; \]
\[ \text{SM.AR} = \text{RESET}; \]

State Diagram SM

State S0:
\[ \text{RA} := 0; \]
\[ \text{WA} := 15; \]
\[ \text{WE} = 0; \]
\[ \text{M} = 0; \]
\[ \text{IF START then S1 else S0}; \]

State S1:
\[ \text{RA} := \text{RA} + 1; \]
\[ \text{WA} := \text{WA} + 1; \]
\[ \text{WE} = 0; \]
\[ \text{M} = 0; \]
\[ \text{GOTO S2}; \]

State S2:
\[ \text{RA} := \text{RA} + 1; \]
\[ \text{WA} := \text{WA} + 1; \]
WE=1;
M=0;
GOTO S3;

State S3: RA:=RA+1;
WA:=WA+1;
WE=1;
M=1;

IF(WA==15) THEN S0 ELSE S2;

end CONT3

B.8 The AHDL Code for the CONT4 Block:

module CONT4
Title 'CONT4'
Declarations
   C    pin; " clock "
   START    pin;
   RESET    pin;
   M    pin istype 'com'; " select to write result of addition or subtraction"
   WE    pin istype 'com'; " write enable for next RAM "

RA3..RA0 pin istype 'reg'; "read address of last RAM"
WA3..WA0 pin istype 'reg'; "write address of next RAM"
RA=[RA3,RA2,RA1,RA0];
WA=[WA3,WA2,WA1,WA0];

Q3..Q0node istype 'reg';
SM=[Q3,Q2,Q1,Q0];
S0=0; S1=3; S2=5; S3=9;

Equations
WA.clk=C;
RA.clk=C;
SM.clk=C;
WA.AR=RESET;
RA.AR=RESET;
SM.AR=RESET;

State_Diagram SM
State S0:
RA:=0;
WA:=0;
WE=0;
M=0;
IF START then S1 else S0;
State S1:
RA:=RA+1;
WA:=0;
WE=0;
M=0;
GOTO S2;

State S2:  RA:=RA+1;
WA:=WA+1;
WE=1;
M=0;
GOTO S3;

State S3:  RA:=RA+1;
WA:=WA+1;
WE=1;
M=1;

IF(WA==15) THEN S0 ELSE S2;

end CONT4
Appendix C

The Architecture's Simulation Waveform Result
<p>| B | DATA_IN7. (hex) | 1 | START | 1 | CLOCK | 1 | RESET | 1 |
| B | RA_ONE3. (hex) | 1 | RAM1DATA7. (hex) | 1 | RA_ZERO3. (hex) | 1 | RAM2DATA7. (hex) | 1 |
| B | RAM3DATA7. (hex) | 1 | WA_TWO3. (hex) | 1 | RAM3DATA7. (hex) | 1 | WA_THREE3. (hex) | 1 |
| B | RA_THREE3. (hex) | 1 | RAM4DATA7. (hex) | 1 | WA_FOUR3. (hex) | 1 | LUT_RA3. (hex) | 1 |
| B | RAM_SEL2. (hex) | 1 | DATA_SEL7. (hex) | 1 | W_ADRES3. (hex) | 1 | R_ADRES3. (hex) | 1 |
| O | U32.DONE | .... | .... | .... | .... | .... | .... | .... |</p>
<table>
<thead>
<tr>
<th>B DATA_IN7. (hex)</th>
<th>1 START...........</th>
</tr>
</thead>
<tbody>
<tr>
<td>B CLOCK............</td>
<td>1 RESETALL.......</td>
</tr>
<tr>
<td>B WA_ZERO3. (hex)</td>
<td></td>
</tr>
<tr>
<td>B DATA_INP7. (hex)</td>
<td></td>
</tr>
<tr>
<td>B RA_ONE3. (hex)</td>
<td>4</td>
</tr>
<tr>
<td>B RAM1DATA7. (hex)</td>
<td></td>
</tr>
<tr>
<td>B WA_ONE3. (hex)</td>
<td>4</td>
</tr>
<tr>
<td>B RA_TWO3. (hex)</td>
<td>4</td>
</tr>
<tr>
<td>B RAM2DATA7. (hex)</td>
<td></td>
</tr>
<tr>
<td>B WA_TWO3. (hex)</td>
<td>4</td>
</tr>
<tr>
<td>B RA_THREE3. (hex)</td>
<td></td>
</tr>
<tr>
<td>B RAM3DATA7. (hex)</td>
<td></td>
</tr>
<tr>
<td>B WA_THREE3. (hex)</td>
<td></td>
</tr>
<tr>
<td>B RA_FORTH3. (hex)</td>
<td></td>
</tr>
<tr>
<td>B RAM4DATA7. (hex)</td>
<td></td>
</tr>
<tr>
<td>B WA_FOUR3. (hex)</td>
<td></td>
</tr>
<tr>
<td>B LUT_RA3. (hex)</td>
<td>4</td>
</tr>
<tr>
<td>B RAM_SEL2. (hex)</td>
<td></td>
</tr>
<tr>
<td>B RAM5DATA7. (hex)</td>
<td></td>
</tr>
<tr>
<td>B DATA_SEL7. (hex)</td>
<td></td>
</tr>
<tr>
<td>B W_ADRES3. (hex)</td>
<td></td>
</tr>
<tr>
<td>B R_ADRES3. (hex)</td>
<td></td>
</tr>
<tr>
<td>B FEEDATA7. (hex)</td>
<td></td>
</tr>
<tr>
<td>o U32_DONE -.....</td>
<td></td>
</tr>
</tbody>
</table>