INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
An application architecture for real time vector quantization

Kaul, Richard Tucker, Ph.D.
The Ohio State University, 1994
AN APPLICATION ARCHITECTURE FOR REAL TIME VECTOR QUANTIZATION

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Richard Tucker Kaul, B.S., M.S.

The Ohio State University

1994

Dissertation Committee:

Steven B. Biblyk
Stanley C. Ahalt
Mohammed Ismail

Approved by

Adviser
Department of Electrical Engineering
To my parents. Theodore and Mary.
ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to Dr. Steven Biblyk for his continued support and guidance during my career at OSU. His understanding and insights have proven invaluable. Thanks also go to the other members of my advisory committee, Dr. Stanley C. Ahalt and Dr. Mohammed Ismail, for their comments, time, and suggestions. The support of the NASA Lewis Research center via grant NAG3-1802, their kind payment for fabrication of the prototype and the help of Monty Andro and Joe Harrold made the research possible. The time and effort of Mark Lindsley, formerly of IBM, for running the chip tester is also appreciated.

I thank my parents for their understanding, patience and support during times that were both rewarding and trying. Without them, none of this would have happened.
VITA

June, 1983 ................. B.S. in Physics and Mathematics,
                         The Ohio State University,
                         Columbus, Ohio

1983–1985 .................. M.S. in Physics,
                         Department of Physics,
                         The University of Illinois at Urbana-Champaign

1986–1992 ................. M.S. in Electrical Engineering,
                         The Ohio State University,
                         Columbus, Ohio

1992–present ............... Staff Engineer,
                         International Business Machines,
                         Boca Raton, FL

PUBLICATIONS

Adaptive Filtering Using Neural Network Integrated Circuits, Richard Kaul, Steve
Bibyk, and Mohamed Ismail. Presented at the 1990 International Symposium on
Circuits and Systems, New Orleans, LA.

An All Digital Implementation of a Modified Hamming Net for Video Compression
with Prediction and Quantization Circuits, Richard Kaul, Ken Adkins, and Steven
Bibyk. Presented at the IEEE International Conference on Systems Engineering,
Dayton, OH.

FIELDS OF STUDY

Major Field: Electrical Engineering

Studies in VLSI Circuit Design, Communication Systems
# Table of Contents

**DEDICATION** .......................................................... ii

**ACKNOWLEDGEMENTS** ............................................. iii

**VITA** ........................................................................... iv

**LIST OF FIGURES** ....................................................... viii

**LIST OF TABLES** ........................................................ xiv

**CHAPTER**

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Introduction .................................................. 1</td>
</tr>
<tr>
<td></td>
<td>1.1 Structure Of This Dissertation ..................... 3</td>
</tr>
<tr>
<td>II</td>
<td>Compression Algorithms ...................................... 4</td>
</tr>
<tr>
<td></td>
<td>2.1 A Brief Overview of a DPCM Video System .......... 4</td>
</tr>
<tr>
<td></td>
<td>2.2 A Brief Overview of MPEG 2 ............................. 6</td>
</tr>
<tr>
<td></td>
<td>2.3 A Brief Overview of Vector Quantization ............ 19</td>
</tr>
<tr>
<td></td>
<td>2.4 Hardware Implementations of VQ ....................... 27</td>
</tr>
<tr>
<td></td>
<td>2.5 Summary ..................................................... 28</td>
</tr>
<tr>
<td>III</td>
<td>VQ Hardware System Architecture ........................ 29</td>
</tr>
<tr>
<td></td>
<td>3.1 A Simplified Model VQ System ........................ 29</td>
</tr>
<tr>
<td></td>
<td>3.2 The Computational Scale Of A Simple VQ System .... 34</td>
</tr>
<tr>
<td></td>
<td>3.2.1 Distance and Metrics ................................. 36</td>
</tr>
<tr>
<td></td>
<td>3.3 Implementation Choices: Analog vs. Digital .......... 41</td>
</tr>
<tr>
<td></td>
<td>3.4 Details of the Digital Design ......................... 44</td>
</tr>
<tr>
<td></td>
<td>3.4.1 Storage of Codebooks ................................. 46</td>
</tr>
</tbody>
</table>

v
# List of Figures

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A modified DPCM system with a nonlinear quantizer and predictor.</td>
</tr>
<tr>
<td>2</td>
<td>A series of frames and how they might be coded in an MPEG 2 data-stream.</td>
</tr>
<tr>
<td>3</td>
<td>The subsampling of the YCrCb image used in the MPEG 2 4:2:0 format. The chrominance components in both the horizontal and vertical dimensions are half the number of the luminance components.</td>
</tr>
<tr>
<td>4</td>
<td>The subsampling of the YCrCb image used in the MPEG 2 4:2:2 format. The chrominance samples in the horizontal dimension is half that of the luminance component.</td>
</tr>
<tr>
<td>5</td>
<td>A sample of the scalar quantizer that is often used in MPEG processes.</td>
</tr>
<tr>
<td>6</td>
<td>The zig-zag pattern of the spatial to frequency conversion. The spatial part is arranged in a standard pixel pattern, while the frequencies are arranged so that the lowest frequencies are in the upper left hand corner and the highest are in the lower right hand corner.</td>
</tr>
<tr>
<td>7</td>
<td>A typical VQ system.</td>
</tr>
<tr>
<td>8</td>
<td>The quantization of the input pattern space $\mathbb{R}^n$ into $j$ decision regions. In the usual case there is only one quantization vector associated with a region, and the region each vector occupies is representative of the probability of correctness of the estimate.</td>
</tr>
<tr>
<td>9</td>
<td>The multi-stage pipeline for multi-stage VQ. Each stage finds the best match for its input and concatenates its output codeword to the input codeword.</td>
</tr>
<tr>
<td>10</td>
<td>A simplified, prototype VQ system.</td>
</tr>
</tbody>
</table>
A sample differential vector quantization system block diagram. ... 32

The difference between the measures $d_1$ and $d_\infty$ and the Euclidean metric $d_2$ in two dimensions on the interval $[0,2]$. The values $d_1$ are denoted by the legend $\text{abs}(x)+\text{abs}(y)$, those for $d_2$ are denoted by the legend $\sqrt{x^2+y^2}$ and those of $d_\infty$ are denoted by $(x>y)?x:y$. ... 39

A block diagram of the overall architecture of the associative processor and the functions of the cellular processors that comprise the associative processor. ................................................................. 41

A winner-takes-all circuit based on current switching. .................. 43

An example of how the pixels are taken out of the image to form the vector to be quantized. ................................. 45

The block diagram of the selected CP function. ............................. 48

An alternate CP functional diagram. .............................................. 49

The overall internal architecture of the Associative Processor. There are two unidirectional busses, the data bus and the control bus, and one bidirectional bus, the match bus. ................................. 55

The flow chart showing the functions performed while storing a memory word. All cellular processors perform the same function synchronously in an SIMD machine structure. ................................. 56

The flow chart showing the functions performed while comparing an input vector with a stored vector. All processors perform the same comparison synchronously. Note that the idle state is used to reset the accumulator that stores the sum of differences. ................................. 57

The high level layout of the control mechanism of the Associative processor. The control machine is a Moore machine, depending only on the current state stored in the State Register so there are no glitches on the output lines. .............................................. 60
22 The finished state machine diagram showing the state assignments and names. Any variables not explicitly mentioned are set to zero by the synthesis program. Also not shown is the reset mechanism, which is activated by the RESET signal and sends the state machine to the IDLE state asynchronously from any state it was in. .................................................... 62

23 The dfr301 D flip-flop with an asynchronous reset and \( Q \) output from the MSU SCMOS library. ................................................................. 69

24 The dnf311 D flip-flop with both \( Q \) and \( \overline{Q} \) outputs........................................ 70

25 The two basic methods for clock distribution: a clock chain (left) where all the buffers are of increasing size and a clock tree (right) where the buffers are the same size, but placed in parallel to reduce the loading on each buffer. ......................................................... 71

26 The clock distribution network for the Associative Processor. ........... 72

27 The problems caused by an edge triggered register with clock skew. In the left case, the combinatorial logic has not settled on a new value by the time the clock edge arrives, so the value in the register doesn’t change. In the right case, the value in the A register is processed by the time the B register is ready, so the value that should have been there from the time period before is lost ............................................................... 76

28 The address decoder circuitry for the Associative Processor. The address lines are buffered in the address register. ........................................... 79

29 A functional diagram of an enabled 4-to-16 decoder. ......................... 82

30 The functional diagram of the Q3 and Q2 part of the Pri8 encoder. . 90

31 The transistor level diagram of the Q0 encoder of the Pri8 encoder. . 91

32 The transistor level diagram of the Q1 encoder of the Pri8 encoder. . 92

33 The circuit that stores a given bit of each of the four components of the stored vector. For example, this circuit might be used to store the MSBs of each of the four byte wide components of the stored word. . 94
34 The bit storage mechanism used. The input line is bi-directional. It serves as the input if load is high, otherwise the line has the output of the memory location and feeds into the arithmetic unit.

35 A bi-directional mux that connects the four bits of the memory storage to the arithmetic unit and/or the data bus. When load is low, the signals flow from the memcells to the arithmetic unit, when load is high, signals flow from the data bus to the memcell selected by the load signal.

36 The signal flow when loading the memory elements for a given cellular processor depending on the state of the control machine. Care must be taken here to control the load signal since it controls the directional flow of data into or out of the memory.

37 The signal flow when a CP is being used in a match operation. The signal flow from the memory is only valid for the first four clock cycles, after which the flow of data from the memory bank and bus lines is not important for the functioning of the CP.

38 The local decode structure for each CP. This takes the global load, store, sel0, sel1 and local address line and creates the local control signals for the CP.

39 The computational part of the cellular processor. There are two banks of eight bit ripple carry adders, four banks of eight bit wide memory, eight XNORs, some additional combinatorial logic for the second bank of adders, and a ten bit accumulator register.

40 The circuit that calculates the values for the A8 and A9 values of the accumulator register.

41 The gate level design for an asynchronously resettable, holdable register. When used in the CP the reset signal is a synchronous signal.

42 The circuit to detect the minimum difference. The circuits are meant to be cascaded, with V_{out} for bit N being used as V_{in} for bit N - 1.
An example of an accumulator register feeding a row of minimum difference subcircuit. The inputs to the minimum difference row are the values in the accumulator register, the valid data stored bit from the control subsection of the CP, and the values on the MATCH bus. The output is the $V_{out}$ signal for the LSB of the accumulator and is fed to the 32 bit priority encoder.

The circuit to split the I/O of the wired-OR internal line into an input and output part.

An example of connecting 4 associative processors together to form a single AP with a maximum size of 128 code words. The connections for the tenth and first bits are shown; the other connections are similar.

The values of the values on the signal lines of the four Cellular Processors involved in the sample operation just before the rising edge at $\phi_1 + 2$ (a) and $\phi_1 + 3$ (b).

The values of the values on the signal lines of the four Cellular Processors involved in the sample operation just before the rising edge at $\phi_1 + 4$ in (a) and $\phi_1 + 5$ in (b).

The values of the values on the signal lines of the four Cellular Processors involved in the sample operation at the end of the matching operation.

The floor plan of the prototype AP.

The layout of the prototype AP.

A high level logical view of two of the MATCH bus of two APs connected together.

A solution to the feedback problem of the wired-OR bus extension circuit.

A simple encoding system that uses the minimum difference and address generated by the AP to encode a new, incremental codebook in a real-time fashion.
54 An iterative solution using a modified AP architecture in which multiple codevectors are associated with each CP to find the minimum difference in a system. Each codevector will be made of multiple components, so the selector is more complicated in this case. ............................. 163

55 A simple parameterization of a partitioning curve. ............................. 168

56 A possible wired-OR circuit. Note the single pullup and many pulldowns. 183
# List of Tables

<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Some sample MPEG data streams and their relative compression.</td>
</tr>
<tr>
<td>2</td>
<td>Some projected bit rates for MPEG 2 compression of various image types. All are at frame rates of 30 Hz.</td>
</tr>
<tr>
<td>3</td>
<td>The quantity of calculations required for a simple VQ system with 256 codewords and 4 \times 4 input vectors.</td>
</tr>
<tr>
<td>4</td>
<td>The state transition table for the Control Finite State Machine. PS is the present state of the state register, and NS is the next state of the state register.</td>
</tr>
<tr>
<td>5</td>
<td>The optimal state assignments for a minimal PLA implementation as found by nova.</td>
</tr>
<tr>
<td>6</td>
<td>The behavior of an enabled 4:16 decoder. EN is the enable line value, C3-C0 the input lines, and DF--D0 the output lines.</td>
</tr>
<tr>
<td>7</td>
<td>The address encoder synthesized behavior. Note that the bit order is opposite that typically used.</td>
</tr>
<tr>
<td>8</td>
<td>The function table for an 8 bit priority encoder.</td>
</tr>
<tr>
<td>9</td>
<td>The sizes of the various elements that make up the cellular processor. These sizes are the ratios of the area each occupies on the floor-plan of the chip and provide clues as to where optimizations are possible.</td>
</tr>
<tr>
<td>10</td>
<td>The transition table for the minimum difference circuit for a given bit position. See the text for a description of what all the values are.</td>
</tr>
<tr>
<td>11</td>
<td>The accumulator values at the end of the first clock cycle of the comparison operation.</td>
</tr>
<tr>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>12</td>
<td>The accumulator values at the end of the second clock cycle of the comparison operation.</td>
</tr>
<tr>
<td>13</td>
<td>The accumulator values at the end of the third clock cycle of the comparison operation.</td>
</tr>
<tr>
<td>14</td>
<td>The accumulator values at the end of the fourth clock cycle of the comparison operation.</td>
</tr>
<tr>
<td>15</td>
<td>The number and type of the pins used in the Associative Processor.</td>
</tr>
<tr>
<td>16</td>
<td>The sizes of the elements making up the AP and their contributions to the overall size of the AP.</td>
</tr>
</tbody>
</table>
CHAPTER I

Introduction

With the inception of the modern Information Age, the need for efficient communication has exploded. Recent years have seen a massive increase in the amount and value of information flowing across the world. Initially the information was transmitted on analog lines for use by humans: telephones, telegraphs and the like. The introduction and tremendous increase in power of modern digital computers have made data manipulation by digital computers an extremely important facet of modern life as the world goes to valuing and producing more intellectual property.

The ease and flexibility of modern computer networks and systems can allow a network to be built to interact with people in their homes, allowing such features as movies on demand, interactive multiplayer games on demand, and shopping and bill paying without leaving home. All of these interactive features depend on digital computers to have some communication between the base system in the home and some central server. To make the system as flexible as possible it is important that the data going both ways is digital, and to make the system as easy to use as possible, we need graphical interfaces to the user. This means that the service providers must provide a way to present video data to the end user over a digital network.

The need for the central server to deliver digital data means that video information
needs to be broadcast over a large domain. This means that an efficient means of encoding an analog image such as those produced by video tape, movies, and most cameras must be encoded in a digital stream, sent over a possibly noisy network, then decoded in many homes. A network such as the one currently envisioned has a central server providing video images over a network, controlled by a simple digital stream from a remote node. Since the central server will feed many nodes the same signal, and since the many destinations will generally be serving an entertainment role in the home, it is important that the node in the home be as inexpensive as possible. As such, it is important to find a communication architecture in which the processing at the remote target node is as simple as possible, while the processing in the server node can be more complex.

Several communications architectures have been proposed, and some are nearing implementation and testing [1]. Most systems proposed for interactive video have depended on MPEG 1 or MPEG 2. The MPEG algorithms are computationally complex and symmetrical. They are symmetrical in the sense that the encoding station uses algorithms based on such things as Discrete Cosine Transforms (DCTs) and other operations. As such, they are computationally intensive and tend to require expensive, high powered chips and large amounts of memory. Putting a myriad of expensive, hot, high speed chips into every home in America may not be attractive or even a viable market solution.

A better solution is a system based on Vector Quantization (VQ). In a VQ system the computations can be very complex at the transmitter, but at the receiver they are
very simple and can be done with very inexpensive hardware. Therefore, the problem we need to overcome is the ability to perform Vector Quantization algorithms in a real time system. We are not terribly constrained by cost in this setup, since the relative cost of the transmitter can be higher since there will be far fewer transmitters required than decoders.

The architecture that we develop will be motivated by the desire to do efficient, robust digital video, but will not be so limited in scope. The system presented here will be a very fast, custom architecture for use with many problems in neural networks, pattern recognition, and other very difficult problems.

1.1 Structure Of This Dissertation

The general reasoning on why digital broadcasting is important has already been presented, as has a brief mention on the relative merits of DCT based algorithms and VQ algorithms. These will be addressed in more detail in the next chapter, as will the overall communications architecture. The third chapter will deal with the architecture of an associative processor designed to enable VQ technology. The fourth chapter will deal with the implementation of the associative processor, and the fifth with its testing. The final chapter will discuss conclusions and provide some insights on future directions. The appendices will deal with various issues that come up during the implementation and provide a synthesizable VHDL description of a chip similar to the one presented.
CHAPTER II

Compression Algorithms

Many compression algorithms for visual data have been proposed, nearly all of which depend on features of the human visual system to achieve high compression. This chapter will briefly discuss a Differential Pulse Code Modulation (DPCM) system, MPEG 2, and Vector Quantization (VQ) and compare the features of each system. We will begin with the simplest of the algorithms, DPCM, and then discuss parts of the much more complex MPEG system. The final discussion will touch on VQ and how it can be implemented in a simpler system like that of DPCM or as a component in the more complicated MPEG system. The demonstration design presented will be for the simpler system.

2.1 A Brief Overview of a DPCM Video System

Much of this research was motivated by a desire to improve upon a DPCM proposed for implementation by NASA [2, 3]. The overall structure of the proposed DPCM system is shown in Figure 1. This particular algorithm varies from a standard DPCM system in that it uses a "non-adaptive" predictor (NAP). NASA hopes that such a system can be used for High Definition Television (HDTV) or other space satellite image transmission.
Figure 1: A modified DPCM system with a nonlinear quantizer and predictor.
In this system the standard television scan line is sampled at four times the color subcarrier frequency. The difference between the digitized sampled signals and the output of the NAP is quantized and the result is Huffman coded. The overall system yields an average compression ratio of approximately 4:1. Such a system was demonstrated in laboratory conditions to produce very high quality images.

The chief disadvantage to a system such as this is its reliance on the Huffman coder for much of its compression. In this system the Huffman coder is responsible for approximately half the compression of the system, but Huffman coders are notoriously sensitive to channel noise. As such, they require frequent synchronization signals to stop errors from propagating. Further, without additional coding there is no way to recover from the error. The losslessness of the Huffman coder is not as useful as it might be in a system such as this because the nonlinear quantizer used already makes the overall system lossy.

2.2 A Brief Overview of MPEG 2

MPEG 2 (ISO 11172) as specified in [4] is a standard video compression technique that is becoming widely accepted as the coming standard for digital video transmission in the US [1]. The MPEG 2 standard actually has an audio compression standard that will not be considered in this discussion, but the details can be found in the MPEG 2 standard. As this dissertation is being written all the major cable television companies have settled upon MPEG 2 as the standard digital format that they will use for digital in the home.

The MPEG 2 proposed standard was defined in a very short time through a
competitive convergence process. The process began with a technology request by the MPEG committee, which resulted in fourteen different techniques being proposed. In under a year the systems were evaluated and an architecture defined. While the standard is still undergoing revision at the time of this writing, the speed with which the process proceeded shows the strong perceived need for a standard digital compression method.

MPEG 2 is a more comprehensive method of compression than that of the DPCM method mentioned before. It uses a combination of interframe and intraframe compression to achieve a good quality image and very high compression through use of both spatial and temporal compression. The design goal for MPEG 2 was a for a "generic" solution that could be used for many applications. As such, it was designed for maximum flexibility. Among the applications for which the algorithms were designed were digital storage media, where the size of digitized video could easily overwhelm the size of even CD-ROM storage devices. Because it was to be generic, the final algorithm had to be symmetrical in that encoding and decoding were to be essentially equal in complexity.

The final algorithm has many features, such as random access, fast forward/reverse searches, reverse playback, audio-visual synchronization, editability, and format flexibility. Further, the coding delay of the system is kept to under 150 ms to maintain a smooth, conversational system useful for video conferencing, although higher image quality can increase the delay to the more noticeable 1 second interval.

The algorithm itself uses many of the features of the video signal to enhance
Figure 2: A series of frames and how they might be coded in an MPEG 2 datastream.

compression. It uses the interframe and intraframe redundancy of the image to its fullest to achieve compressions of approximately 100:1. As an example of the full features, consider what a series of frames to be transmitted might look like, such as the one in Figure 2. The MPEG 2 standard defines three kinds of pictures:

- Intrapictures, usually denoted as I, or I frames. These pictures do not depend on any other pictures.

- Predicted pictures, usually denoted as P, or P frames. These depend only on the last P or I frame.
Table 1: Some sample MPEG data streams and their relative compression.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bicycle: 352 x 240</td>
<td>10</td>
<td>24:1</td>
<td>40</td>
<td>36:1</td>
<td>100</td>
<td>75:1</td>
</tr>
<tr>
<td>Earth: 352 x 288</td>
<td>121</td>
<td>21:1</td>
<td>120</td>
<td>35:1</td>
<td>479</td>
<td>191:1</td>
</tr>
<tr>
<td>Flowers: 352 x 240</td>
<td>10</td>
<td>14:1</td>
<td>40</td>
<td>31:1</td>
<td>112</td>
<td>75:1</td>
</tr>
</tbody>
</table>

- Bidirectional pictures, usually denoted as B, or B frames. These are bidirectionally predicted pictures and depend on the previous picture and the next picture.

The I frames provide random access points into the MPEG 2 stream and are thus used as fast-forward and rewind marker images, and as resynchronization frames when the decoder and transmitter lose synchronization through noise, buffer overrun, or other mechanisms. These I frames are essentially JPEG images and provide the least compression. The P frames depend on the previous image, whether it be an I frame or a decoded P frame. They are used in the motion compensation algorithm and provide moderate compression. The B frames depend on the previously decoded frame, from either an I or P frame, and the next decoded frame, also either an I or P frame. Since it depends on both a past and a future frame, it can provide very good motion-compensated interpolation. B frames have the highest compression ratios of any of the frames. As an example, consider Table 1, where some examples are given of the compression of some MPEG streams.

As can been seen from Table 1, much of the compression in the data stream
comes from the B frames. This compression does not come freely, however, and adds significant complexity to the overall data stream. For example, the series of frames in Figure 2 have to be viewed in the order:

1 2 3 4 5 6 7 8 9

but would have to be transmitted as

1 5 2 3 4 9 6 7 8.

This means that a minimum of two complete frames must be kept in local storage for the MPEG 2 decoder. While the MPEG 2 standard does allow data streams without B frames, these data streams take more bandwidth and do not fall inside the main profile/main level definition of MPEG 2 and are thus not recommended. Even at a minimum one complete frame must be held in memory, since P frame support is always required. For this reason, all MPEG 2 chips supporting main level/main profile at CCIR 601 resolution require at least 2 Mbytes of RAM; systems not supporting B frames require 1 Mbyte of RAM.

The compression ratio of the overall MPEG 2 datastream can vary quite a bit depending on application and subject matter, but for most video the 100:1 ratio results in image quality approximately equal to that generated by VCRs. Some projected bit rates are shown in Table 2. The interframe dependency is required to achieve the very high compression ratios required, but the random-access nature of the standard favors pure intraframe encoding. The combination of requirements is satisfied by a delicate balance of block based motion compensation for temporal
Table 2: Some projected bit rates for MPEG 2 compression of various image types. All are at frame rates of 30 Hz.

<table>
<thead>
<tr>
<th>Format</th>
<th>Video Parameters</th>
<th>Compressed Bit Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIF</td>
<td>352 × 210</td>
<td>1.2-3 Mbps</td>
</tr>
<tr>
<td>CCHR 601</td>
<td>720 × 186</td>
<td>5-10 Mbps</td>
</tr>
<tr>
<td>EDTV</td>
<td>960 × 486</td>
<td>7-15 Mbps</td>
</tr>
<tr>
<td>HDTV</td>
<td>1920 × 1080</td>
<td>20-40 Mbps</td>
</tr>
</tbody>
</table>
Figure 3: The subsampling of the YCrCb image used in the MPEG 2 4:2:0 format. The chrominance components in both the horizontal and vertical dimensions are half the number of the luminance components.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table: 

- X represents luminance pels
- O represents chrominance pels

For the spatial quantization MPEG 2 chose to do two separate techniques. First, the standard uses color subsampling to achieve some compression. MPEG 2 uses the YCrCb system, which is often mistakenly called the YUV system. The luminance, or Y, component is fully sampled in a 4 × 4 block, but the other two chrominance components are typically subsampled so that there is only one or two samples taken from the 4 × 4 block, as shown in Figure 3 and Figure 4. In the 4:2:0 format shown in Figure 3 the Cb and Cr matrices are half the size of the Y-matrix in both the horizontal and vertical dimensions. The 4:2:2 format shown in Figure 4, where...
Figure 4: The subsampling of the YCrCb image used in the MPEG 2 4:2:2 format. The chrominance samples in the horizontal dimension is half that of the luminance component.
the both the Cb and Cr matrices are one half the size of the Y-matrix. The 4:2:0 and 4:2:2 formats are most commonly used because the human eye is very sensitive to changes in light intensity, but less sensitive to colors. There is also a 4:4:4 format where the Y, Cb, and Cr matrices are all the same size.

The other method of intraframe spatial compression used is the DCT. The MPEG standard specifies using the DCT on 8 x 8 blocks of data over vector quantization techniques. While several authors have pointed out that VQ techniques could easily have been inserted into the standard for spatial compression [5], the committee felt more comfortable using the same mechanism as that used by its sister JPEG committee. In fact, the default DCT quantization table was chosen to be that developed by the JPEG committee. The decision to use the DCT algorithm is like most engineering tradeoffs: there are good points and drawbacks. Using the DCT meant that silicon designs already available for JPEG chips could be used for MPEG chips, but using a DCT means that the receiver must perform a similarly intensive and silicon-area intensive operation.

The DCT algorithm in the forward (encoding) direction for 8 pixels in one dimension can be written as

\[
S(u) = \frac{C(u)}{2} \sum_{x=0}^{7} s(x) \cos \left( (2x + 1) \frac{u\pi}{16} \right) \tag{2.1}
\]

where

\[
C(u) = \begin{cases} 
\frac{1}{\sqrt{2}} & \text{for } u = 0 \\
1 & \text{for } u > 0 
\end{cases} \tag{2.2}
\]

\[
s(x) = \text{1-D sample value} \tag{2.3}
\]

\[
S(u) = \text{1-D DCT coefficient} \tag{2.4}
\]
In two dimensions the DCT for an $8 \times 8$ block is defined to be

$$ S(u, v) = \frac{C'(v)C'(u)}{2} \sum_{y=0}^{7} \sum_{x=0}^{7} s(x, y) \cos \left[(2x + 1)\frac{u\pi}{16}\right] \cos \left[(2y + 1)\frac{v\pi}{16}\right] $$  \hspace{1cm} (2.5)

where

$$ C'(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } u = 0 \\ 1 & \text{for } u > 0 \end{cases} $$ \hspace{1cm} (2.6)$$

$$ C'(v) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } v = 0 \\ 1 & \text{for } v > 0 \end{cases} $$ \hspace{1cm} (2.7)

$$ s(y, x) = \text{2-D sample value} $$ \hspace{1cm} (2.8)

$$ S(u, v) = \text{2-D DCT coefficient} $$ \hspace{1cm} (2.9)

These equations are very computationally intensive, so much so that in their present form they are rarely used. A good review of the literature as in [6] will show many innovative fast DCTs. For one dimensional encoding a good algorithm [7] can get by with as few as 5 multiplications, although additional divisions are required for the scalar quantization that comes after the DCT is applied. For the two dimensional DCT, the best algorithm [8] requires 54 multiplies, 462 additions and 6 shifts, not including the quantization calculations.

At this stage no information has been lost and no compression has been done. All that has been accomplished is that spatial information has been transferred to the frequency domain. At this point, the MPEG standard does a scalar (one-dimensional) quantization step. This is done to provide compression and it depends on the fact that the human eye is less sensitive to the higher frequencies than the lower ones, often perceiving the high frequencies as "noise." An example of a scalar quantization that
Figure 5: A sample of the scalar quantizer that is often used in MPEG processes.
is used is shown in Figure 5. After the values of the components are quantized, the results are scanned in a zig-zag pattern like that in Figure 6 and fed into a Huffman encoder. The zig-zag pattern is used because typically on the lower frequencies have much data and the Huffman coder works more efficiently if long strings of repeated data are available.

For an I frame at this stage the computations are complete. This discussion will not concern itself with the motion compensation and prediction elements since they are beyond the scope and interests of this discussion, which will center on the complexity of the spatial compression used.

Obviously, these are very serious computational requirements. The prospects for using MPEG 2 are improved somewhat by the fact that many vendors were starting from bases that included JPEG chips, which perform these calculations in much the same way as MPEG 2 does. This has meant that many vendors are rushing to market with MPEG 2 chips in a bid to set a standard for digital image transmission.
While MPEG 2 has the advantage of being most ready for implementation, there are many drawbacks for MPEG 2 based systems. The generation of chips being readied for use in MPEG 2 home units are still expensive, large, hot chips that will make initial systems rather expensive. A representative sample of chips ranges from the SGS-Thompson STi3500, to the C-Cube SD4, to the IIT VCP. Both the SGS and C-Cube parts require a fast processor to separate the MPEG 2 stream into its data and control components, while the IIT chip can do all the data stream parsing, but at a higher cost. The complexity of these chips is very high, and all use relatively fast clock cycles of approximately 40-80MHz, and all are fabricated on fast, submicron silicon lines. The number of I/O lines in these chips requires packaging to be in 160 (or more) pins in plastic quad flat packages (PQFP), and the chips generate between one and two watts of heat. The typical fabrication process is done in a process that is between 0.6 μm and 0.8 μm. The prices of these chips indicate the complexity and technology involved in their design: in 1995 dollars in quantities of greater than 250,000, these chips are expected to cost between $35 (for an MPEG 2 video only decoder) and $100 (for the more exotic MPEG 2 video and audio decoder) US dollars. Not only is the cost of the MPEG 2 chip high, these chips also require a significant amount of dedicated storage for their use. As mentioned before, for CCIR 601 resolution MPEG 2 main level/main profile compliant chips a minimum of 2 Megabytes of DRAM are required (more is required to support the equivalent PAL resolution). In 1994 dollars, the price of such a memory subsystem is expected to be approximately $40 US.
As we can see, there is a high price to be paid for using MPEG 2 in systems. Not only is there a chip price exceeding $75 for the raw parts, there may well need to be a fast microcontroller in the system to parse the data stream, and there needs to be a significant board area allocated to the MPEG 2 system for the video/audio decoder chips and their associated DRAMs. As such, the expected price of video decoder boxes in 1995 is expected to be much more than $500. Whether the market accepts systems with such a high price tag remains to be seen.

2.3 A Brief Overview of Vector Quantization

Vector quantization (VQ) can be introduced in many ways, but perhaps the most interesting is to view it as a learning process. DPCM and MPEG 2 have direct roots in communications theory, but VQ can be more easily visualized as an intelligent, possibly adaptive selection of representative elements of a decision space. While not a new field by any means, interest in VQ has exploded in recent years as new research and algorithms have brought innovative implementations into the realm of possibility with modern VLSI techniques. A summary of some of the work can be found in [9, 10].

A typical VQ system is shown in Figure 7. In such a system there are c vectors which are chosen to be representative of the input image. The input pattern space $\mathbf{R}^n$ will be partitioned by the memory elements $\mathbf{m}_i$ into at most c separate memories, which will in turn split the pattern space into at most c regions as shown in Figure 8. In its full adaptive form, the system can be allowed to move the representative vectors $\mathbf{m}_i$ to find optimal code books. The movement of the memory elements $\mathbf{m}_i$ through
Figure 7: A typical VQ system.
Figure 8: The quantization of the input pattern space $\mathbb{R}^n$ into $j$ decision regions. In the usual case there is only one quantization vector associated with a region, and the region each vector occupies is representative of the probability of correctness of the estimate.
the pattern space $\mathbb{R}^n$ represent system learning and is one of the most interesting and promising parts of the system.

The number of memory elements, $c$, clearly has a large role in how good the image quality is after compression. The more elements, assuming that they are well trained, the better the reconstructed image. The dimension of the vector to be quantized also plays a fairly significant role in the image quality. Smaller vectors like $2 \times 2$ yield the least "blockiness", while larger vectors like $8 \times 8$ yield higher compression. Various schemes have been tried to mix vector sizes to produce higher image quality.

Vector quantization is the natural result of the process of self-organization [11] if the system is adaptive in appropriate ways. In an adaptive VQ (AVQ) system the system can settle into an optimal arrangement only if the training set and adaptation rules are an exact match to the subject data. The exact nature of the arrangement will depend on the performance criteria used in the measurement, such as minimizing the mean-square error between the training set and the prototype vectors. In the ideal case, the distribution of the learned vectors should statistically resemble the (possibly unknown) distribution of patterns in the training set.

It is possible to choose a completely optimal memory arrangement. This upper bound is achieved by a full search of the input space with all the input elements using the Linde–Buzo–Gray (LBG) algorithm [12]. While this is an optimal codebook, the algorithm is quite complex and generally must be done off-line in batch mode as it is very time consuming. The complexity of this algorithm and the fact that the human visual system does not need a completely optimal codebook have given rise to many
sub-optimal systems that are more easy to implement and yield acceptable results.

There are many ways of generating sub-optimal VQ codebooks and their methods and results give varying degrees of success depending on the image to be compressed. For example, the standard VQ system utilizes the spatial redundancy of an image for compression. That same spatial coherency can be further exploited by using a cache structure in the system as shown by [13]. In these cache codebook (CCB) systems a number of the most recently accessed codewords are stored. When a new input vector is applied to the system, the system checks its cache of recently accessed codewords and finds the best match. If this match is below some threshold value that codeword is used, but if the match is not accurate enough the rest of the codebook is searched.

Other VQ algorithms use the familiar tree search method from computer science on the input space. In such a search a series of hyperplanes are applied as tests to the input vector. The search continues down a branch of the code tree based on some testing criteria. This method, first proposed by [14], does demand that the codebook be designed such that a test can be applied at each node, which can be very difficult in practice. The simpler binary tree search can be further generalized to trellis and lattice searches as done by [15, 16].

Other methods to produce VQ codebooks by breaking the quantization into distinct steps. In these applications the input vector is applied to a small codebook. The best match is subtracted from the input vector and the residue is passed on to the next quantizer in the stream. At the next processing element the best match is again selected and the remainder passed to the third processor, and so on. This setup.
called multi-stage VQ (MSVQ) is shown in Figure 9. The codewords generated at each stage of the process are concatenated to form a final codeword [17]. The chief advantage of methods such as this is that the number of codewords is small at each stage, so comparisons can be done quickly.

There are many other similar methods, such as Separating-Mean VQ [18] and Gain/Shape VQ (GSVQ) [14]. GSVQ is the more interesting of these two, since it uses two independent codebooks to quantize the shape and gain of the waveform.

It is important to note that that methods like MSVQ and GSVQ only simplify the calculation by ordering it, not by exploiting the underlying structure of the data. For the case of video and speech data there is a large amount of underlying redundancy in an image that must be accounted for to accomplish more optimal compression. For more optimal compression of data with high spatial or temporal correlation methods such as preprocessors and predictors can significantly improve performance [19, 20, 21, 22].
Finally, it is important to notice that not only does VQ have an important role in spatial compression, it can also be used effectively in interframe/interfield coding for even better compression. Rao et al [23] have shown a tree-based simplified VQ system in simulations that reported very low bit rate transmissions.

It is important to note that VQ is very asymmetric process. The decoding process is very simple since it is merely a table lookup function that matches the address of the incoming codeword to the value of the codeword itself. The encoding process is the one that is very complicated and can take a long time. This asymmetry is very desirable for many applications, however. For broadcast operations such as cable TV, direct satellite broadcast, and the like, the fact that the large number of decoder boxes can have a very cheap decoder subsystem easily offsets the requirement for a few more expensive encoders. This fact has not escaped the attention of the brutally price conscious personal computer market. It is well known that consumers will not pay much for added features unless they are deemed necessities. In the case of multimedia video, this means that adding hardware for decoding video images is not practical. Realizing this, several companies have introduced VQ based algorithms for video playback. Both Intel's Indeo and Supermac's Cinepak use VQ technology to implement video decoders than can play 15-20 frames per second on dumb framebuffer systems under Microsoft's Windows 3.1 on Intel 80486-class machines. The encoding process for the two systems either requires very expensive hardware, or very long runs on standard microprocessor architectures.

An additional benefit to VQ based systems is that VQ produces fixed length
codebooks. For algorithms that can produce nearly maximal-entropy codebooks, the compression achieved is close to optimal without resorting to additional methods such as variable length coding [24]. If the codewords are arranged such that their relative distance is similar to the Hamming distance between their indices, the coder performance will not degrade as quickly in the presence of channel noise as would a variable-length coder [25]. This is because when a single channel bit error occurs, the decoder selects a codeword that is selected is most similar to the original. In comparison, a variable-length code will be corrupted if even a single bit is corrupt, which means that digital transmission of variable-lengths codes requires very high signal to noise ratios (QAM–64 requires approximately 27 dB, for example) or the addition of error detection/error recovery codes, which will decrease the effective compression of the digital data stream.

As was mentioned earlier, there is no reason that a VQ–based MPEG–type standard could not be proposed, substituting VQ techniques for the DCT and other operations. For example, an adaptive VQ system could achieve very good spatial compression without depending on the noise susceptible DCT algorithm. Further, VQ techniques map very well into the motion compensation algorithms where blocks of pixels in two images must be compared to find maximally likely matches. This motion compensation is done on 16 × 16 pixels blocks in the MPEG 2 specification, since in sequences where there is movement the image tends not to change too much. This means that any VQ application to this mapping must be done with an architecture that scales well to larger dimension.
2.4 Hardware Implementations of VQ

VQ suffers in comparison to DCT-based compression techniques in the wealth of hardware available. This is because DCT algorithms are relatively easy to understand and implement in a standard digital domain. MPEG and JPEG algorithms can be done in specialized chips at full speed, or in standard DSPs at reduced performance. In comparison, VQ is not as well suited to a standard serial hardware design. Other circuit organizations must be implemented to efficiently construct a VQ system.

Panchanathan and Goldberg [26] proposed an interesting CAM-based architecture for VQ encoding. They use CAMs with mask registers to serially narrow the search for the nearest matching codeword to the input vector. They accomplish this by storing the input vector in the CAMs and applying the codewords serially to the hardware. While an innovative approach, it is unwieldy in practice since it requires a very large number of chips and relatively complex control.

Ramamoorthy et al proposed a bit-serial approach using a systolic array implementing the MSVQ algorithm mentioned earlier. Using an image block of $4 \times 4$ they used a 2 stage design where the first stage used 128 codewords and 64 in the second stage. The bit-serial design allowed the system to efficiently calculate the Euclidean distance between the input vector and the codevector. Their hardware, implemented in $3\mu m$ CMOS, used an inner product processor occupying $62 \text{ mm}^2$ and took 25 clock cycles to complete its calculation. For the full implementation of this architecture 256 inner product processors would be required, which is effectively too large for implementation.
Other researchers have proposed using analog elements as part of the encoder to increase speed and decrease area. Sheu et al [27, 28] have a prototype with 64 $5 \times 5$ codewords in an area of 4.6mm $\times$ 6.8mm in 2 $\mu$m CMOS. The prototype can quantize on pattern every 500ns. Tuttle et al [29] show a design with 256 $4 \times 4$ codewords on a die of 6.8mm $\times$ 6.9mm that operates at 5MHz. Both use the Euclidean distance between the input vector, but the nearest matches were found differently. Sheu used a multi-input analog WTA circuit similar to those used by Mead [30], where Tuttle used a binary tree comparison structure.

2.5 Summary

There has been a good deal of research into VQ codebook design and construction, particularly into algorithms for efficient, suboptimal generation. The research into implementable algorithms has been lacking, however. It can be easily shown that standard digital computer architecture does not adapt well to VQ’s associative searches, so other more novel organizations are needed. As hinted in the previous section we shall see that the single instruction, multiple data (SIMD) machine maps well into solving the problem of effective implementations of real time vector quantization. The rest of this work will describe the design of an efficient quantizer for real time video applications.
CHAPTER III

VQ Hardware System Architecture

The design goals of the overall system have been presented in the previous chapter: an efficient scheme for broadcasting digital video that is also as cost effective as possible. As mentioned in the previous chapter, a vector quantization algorithm is more economical than other algorithms. This due to the fact that the receivers have simple, easy to implement and cheap decoders. This comes at the expense of more complicated encoding in a vector quantization system, which is an acceptable cost since there are far more receivers than transmitters in real broadcast systems. Therefore, this chapter will introduce the considerations that drove the architecture and resulted in the final design of a real-time vector quantizer for use in the encoder of a video signal in a vector quantization system. We will touch briefly on the scale of the calculation for a vector quantization system, the architectural constraints and the overall system implementation.

3.1 A Simplified Model VQ System

Let us first begin by considering a generic and straightforward vector quantization (VQ) system. A simplified, conceptual model of a VQ system is shown in Figure 10. This system consists of an input vector, a quantizer, a transmitter and channel, a
Figure 10: A simplified, prototype VQ system.
receiver, an inverse quantizer (done by table lookup), and an output channel. The system is intentionally kept simple for conceptualization: we will discuss some of the additional complications of a realistic system later.

In this basic system, the job of the quantizer is to take input vectors and find which of the codebook vectors most nearly matches the input vector. In this system, the input vector $d$ is presented to the quantizer, which then finds the codeword vector $x_s$ nearest to the input vector $d$. Once the codeword $x_s$ is found, the system outputs the corresponding symbol $s$. This symbol $s$ is then transmitted over the channel to the receiving system. The receiving system takes the incoming symbol $s$ and translates it into the recreation vector $\hat{x}_s$ stored in its codebook.

This system is conceptually very simple, but more difficult in practice. The receiver merely has to do a table lookup to get the correct recreation vector based on the input symbol, which can easily be done using most any embedded controller. Post-processing or other image improvement techniques may require additional hardware which will not be discussed here. The transmitter has the more difficult task of calculating which recreation vector is closest to the input vector based on a given distortion measure. This means generating the distortion measure, then comparing all the distortion measures for each codeword with each other and finding the minimum, and finally encoding the output symbol based on the minimum distortion measure. For large codebooks with $N$ entries merely finding the minimum difference in a serial computer can take $N - 1$ operations, irrespective of the number of other operations to be done. This is important since the larger the number of codewords, the smaller
the distortion measures and the better the image looks, providing the codebook was well constructed. Clearly, for a real time system some parallelism must be employed and appropriate distortion measures chosen.

An improved system over the simplest system to be discussed here and in the rest of this document would be the Differential Vector Quantization (DVQ) system shown in Figure 11. The DVQ system borrows several architectural similarities from the Differential Pulse Code Modulation (DPCM) system. In DPCM there is a CODEC (COder-DECoder) structure, and a typical system will employ a multi-
level Huffman compression system to achieve good results. An example of a DPCM system with a nonuniform quantizer and multi-level Huffman compression scheme is given in [2]. The DPCM structure results in very good compression, but relying on Huffman coding has several drawbacks. A Huffman code is of variable length since the compression is non-uniform, so any system using a Huffman code must have buffers sufficiently large to handle the worst case scenarios. Further, Huffman codes depend on the previous code words being correctly decoded, so recovery from channel noise is very difficult and many additional synchronization symbols are required.

A DVQ system differs from a VQ system in that the vectors of differences between vectors are vector quantized, and the recreation symbol for this difference is transmitted to the receiver. The receiver takes the symbol and creates the recreation difference vector and adds it to the predicted difference vector. In particular, the system works by taking a predicted value, \( p \), generated from the previous vector(s) and subtracting it from the input vector, \( x \).

In the receiver, this difference vector, \( y \), is then vector quantized and the recreation symbol generated. This recreation symbol is transmitted across the channel and feedback into an inverse quantizer (table lookup) to regenerate \( \hat{y} \). This difference \( \hat{y} \) is then added to the predicted value, \( p \), to create the reconstructed vector, \( \hat{x} \).

The transmitter has a complete receiver circuit in it which it uses to create the next predicted pixel based on the previous vectors, so that both the receiver and the transmitter agree on the predicted vector, \( p \).

In a DVQ system using codebooks generated by the FSCL algorithm [31], the
codebooks are made such that the probability of any codeword appearing is nearly equal. This means that Huffman coding is not required to achieve high compression. Further, by arranging codewords such that codewords which are Hamming distance close are also distortion measure close, single bit errors in the transmission channel will have an even lower effect. For a Hamming compressed data stream, a bit in error will cause all codewords after that bit to be incorrectly decoded, and an error bit in the synchronizing codeword will cause the entire line to be incorrectly decoded. For a FSCL DVQ system, a single bit error will cause only an error in the vector being quantized, and that additional distortion can be minimized by arranging the codewords appropriately as described before. For some detailed analysis of a DVQ system see [32].

3.2 The Computational Scale Of A Simple VQ System

The calculations for a VQ system on the face of it seem rather simple. One vector must be compared to the vectors in a codebook and the index of the best match sent. The problem comes when the comparisons must be done in real time on data as fast as that coming from a video stream.

Consider the case for which our prototype hardware was designed: the input is a real time monochrome video stream, the input is assumed to be a block of $2 \times 2$ pixels, and there are 256 codewords in the codebook. For a completely serial approach as might be done with a standard microprocessor architecture, this means that many calculations must be done. For each codeword there have to be 4 subtractions (one for each of the vector components), 4 multiplications (to square each of the component
Table 3: The quantity of calculations required for a simple VQ system with 256 codewords and 4 x 4 input vectors.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Times Performed/Vector</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector component subtraction</td>
<td>4</td>
<td>1024</td>
</tr>
<tr>
<td>Vector component difference multiplication</td>
<td>4</td>
<td>1024</td>
</tr>
<tr>
<td>Vector difference addition</td>
<td>3</td>
<td>768</td>
</tr>
<tr>
<td>Vector distance square root</td>
<td>1</td>
<td>256</td>
</tr>
<tr>
<td>Vector distance comparison</td>
<td>1</td>
<td>256</td>
</tr>
</tbody>
</table>

differences). 4 additions (to add the squares of all component differences) and a square root taken of the result. Since this must be done for each codeword, this must be repeated 256 times. Finally, all the differences must be compared to find the minimum difference. So overall, we must perform 768 additions, 1024 subtractions, 1024 multiplies, 256 square roots and 256 comparisons in a period of approximately 30μsec. This is clearly beyond the capabilities of modern microprocessors. The results are shown in Table 3.2.

Given the shear quantity of calculations that must be done, it is clear that some simplification must be done to this system to make it viable for real-time operation. To begin, we can note that the calculations are extremely regular in that all codewords have the same value subtracted from each of their components, and that all other mathematical operations performed on the vector elements are the same. This means that a single instruction, multiple data (SIMD) machine is an ideal target architecture. In SIMD machines there are multiple subprocessors operating in lockstep on data. All subprocessors perform the same function at the same time on possibly different
data. There is a common instruction control unit that controls the operation of the system. This architecture maps well to implementation in silicon, since it is easy to replicate structures, and since the behavior of SIMD machines is well understood [33].

Having settled on a SIMD architecture further simplification must be done to make the size of the architecture palatable. We can reasonably restrict the incoming data to integers if we assume the data is digitized by standard methods. This is only somewhat helpful since it means that only the subtraction, multiplication and addition operations can be performed by integer units. The square root and comparison must still be floating point operations.

### 3.2.1 Distance and Metrics

For the quantizer in our sample system, we need to enable to find the “closest” match in our codebook to a given input vector. Since the system uses input vectors we can begin by using the standard Euclidean definition of distance:

\[ d = \sqrt{\sum_{i=0}^{N-1} (a_i - b_i)^2} \]  

(3.1)

where \( N \) is the dimension of the vector, \( a_i \) and \( b_i \) are the components of the two vectors, and \( d \) is the distance between the two vectors. This definition of distance is not ideal for mapping into modern VLSI, however. The multiplications and square root are not area efficient calculations and using them in an SIMD structure would quickly limit the size of the SIMD machine.

If the Euclidean distance is not well suited to the implementation of the problem, there are other metrics whose distances could be used. Einstein showed that even
the Euclidean metric above is only an approximation of the metric governing the
real world as we know it, and in so doing radically changed physics at a high energy
level, but which had little effect on daily life. The success of Einstein’s search for a
new metric shows that it is reasonable to consider that other metrics can be suited
for various descriptions of the natural world. Mathematically, a metric space can be
defined as a pair \((X, d)\) consisting of a set \(X\) and a distance, \(d\), \(i.e.\) a single valued,
nonnegative, real function \(d(x, y)\) defined for all \(x, y \in X\) which has the following
three properties:

1. \(d(x, y) = 0\) if and only if \(x = y\):
2. Symmetry: \(d(x, y) = d(y, x)\):
3. Triangle inequality: \(d(x, z) \leq d(x, y) + d(y, z)\):

These spaces are usually denoted by a single letter, like

\[ R = (X, d). \]  \(3.2\)

It is important to note that the underlying set can be many things. For example, the
set \(C[a, b]\) of all continuous functions defined on the closed interval \([a, b]\), with distance

\[ d(f, g) = \max_{a \leq t \leq b} |f(t) - g(t)| \]  \(3.3\)

is a metric space of great importance in mathematical analysis.

If we return to the set of \(n\)-tuples of real numbers \(x = (x_1, \ldots, x_n)\), we can define
a generalized distance measure as

\[ d_p(u, v) = \left( \sum_{i=1}^{n} |u_i - v_i|^p \right)^{1/p} \]  \(3.4\)
where $p$ is a fixed number and $p \geq 1$. The proof of conditions (1) and (2) of the metric definition are obvious. The triangle inequality is more difficult and is given in Appendix A. $d_2$ is the Euclidean distance typically used for distance measurements. $d_1$ and $d_\infty$ are also popular metrics in their own right [21, 26] due to their computational simplicity. Both $d_1$ and $d_\infty$ avoid the relatively high overhead of exponentiation. The lack of multiplication is obvious for $p = 1$; for $p = \infty$ there is also no multiplication because the distance measure becomes just the largest of the component differences.

The generalized metric 3.1 has several interesting properties. It can be shown that

$$d_1 \geq d_2 \geq \ldots \geq d_p \geq d_\infty. \quad (3.5)$$

$d_1$ is a very popular measure that is often called the Manhattan distance, from the rectilinear layout of much of that city, or the Minkowski metric after the mathematician who provided much of the basis of metric space theory in mathematical analysis.

As an example of what different values of $p$ in 3.4 can yield consider the cases of $p = 1$, the Minkowski metric, and $p = 2$, the Euclidean metric in two dimensions. By straightforward analysis it can easily be shown that the maximum difference between the two metrics occurs when $x = y$ and that in that case $d_1 = d_2/\sqrt{2}$. The difference between the distance measures $d_1$, $d_2$ and $d_\infty$ on the interval $[0, 2]$ are shown in Figure 12. Note that for values along the $x$ and $y$ axes the distances all measure the same values. Only positive values need to be plotted because of the first two properties of the metric definition.

The disadvantage using an alternate metric is that it tends to exaggerate the image degradation produced by the vector quantization process when judged by the
Figure 12: The difference between the measures $d_1$ and $d_\infty$ and the Euclidean metric $d_2$ in two dimensions on the interval $[0, 2]$. The values $d_1$ are denoted by the legend $\text{abs}(x) + \text{abs}(y)$, those for $d_2$ are denoted by the legend $\sqrt{x^2+y^2}$ and those of $d_\infty$ are denoted by $(x>y)?x:y$. 
standard measurement of image quality. For most image processing applications the key metric quoted by researchers is the normalized mean square error (NMSE), which is given by

$$\text{NMSE} = \frac{\sum_{i=1}^{n} (x_i - \hat{x}_i)^2}{\sum_{i=1}^{n} x_i^2}$$

(3.6)

where $x$ is the input image pixel and $\hat{x}$ is the reconstructed image pixel. As should be obvious this measure of image quality is strongly biased towards the $d_2$ measure due to its structure.

It is quite impossible, however, to reduce the very complex human visual system to a representation by one number [30]. Most researchers take the NMSE numbers as only a very rough guide to the image quality produced by a process [34]. The human visual system is very sensitive to small variations in smooth backgrounds, and is very good at picking up errors along edges. Errors in random patterns or very complicated images tend to be lost in the visual "noise" of such images. However, edge performance of images using the $d_1$ metric can be as good as that of the $d_2$ [32].

Given this lack of understanding of the underlying dynamics of the human visual system, about all that can be done to test a proposed system is to test it and have a group of humans look at the results for a set of images. This was done and a large sample of images showed no visible degradation between the $d_1$ and $d_2$ measures. The relative simplicity of the $d_1$ metric made it the choice of fabrication.

Having settled on the $d_1$ metric, the overall architecture of the associative processor that would drive the vector quantizer was clear. As shown in Figure 13, the basic structure is an SIMD machine with as many subprocessors (which will also be called
cellular processors or CPs) as codewords in the codebook. An input vector is broadcast to all the CPs, which then take the absolute value of the difference between the input vector component and the stored codeword vector component, and add those values together to form the total distance between the input and the codeword. Finally, all the CPs must do a cooperative calculation to find the minimum distance.

3.3 Implementation Choices: Analog vs. Digital

Analog signal processing has long been the baseline system for high bandwidth data transmission and manipulation. In the earliest stages nearly all computers were analog and large systems were developed in the hopes of doing complicated calculations.
Analog computers offer the highest possible number of calculations per second, but they suffer from the drawbacks of being highly specialized, very difficult to change and design, and very difficult to program.

Digital signal processing (DSP) has grown steadily more powerful over the years as the density of circuit elements on a chip has exploded exponentially. The sheer number of transistors available in practical chips and their increase in speed have meant that a shift towards less efficient but more flexible designs is possible. Digital circuitry offers flexibility, ease of programming, logarithmic growth in area as accuracy increases, fixed and predictable error conditions, performance, and yield, reliable storage, and analytic properties.

Many of the issues involved in digital vs. analog designs were evaluated early in the design cycle. Several chips were built using analog winner-takes-all (WTA) circuits to evaluate design possibilities for the comparison operations that would need to be done in the final chip. While the WTA elements like those shown in Figure 14 were successful, it was determined that an analog implementation of the rest of the chip would be wasteful in terms of area and design effort since it would not be an easily scalable chip. Further, there is a problem with memory stability and accuracy in an analog system. Kerns [35] showed much the same results in 2μm CMOS technology. His results showed that for typical processes the area used by an analog comparator equaled that of a digital comparator at approximately 9 bits of accuracy, but the analog comparator was significantly more difficult to design and analyze. Further, as the technology scales to both smaller geometries and larger chips the digital design
Figure 14: A winner-takes-all circuit based on current switching.
scales very easily, whereas the analog design will not scale as well due to increased short channel effects below 2\(\mu\)m gate lengths and due to the fact that the transistors would become much more difficult to match over a large chip for an analog design. The final decision was that to come up with a flexible, extensible design a digital architecture was better than an analog system.

3.4 Details of the Digital Design

One of the advantages of digital design is that it is very easy to design a machine such that specific actions are taken at a specific time and in given special purpose subsections. The block diagram for the prototype digital system is given in Figure 7. There is a front-end section to the box which takes the input signal, digitizes it, and buffers it for use by the associative processor. This buffering is required because the quantizer will grab 2 \(\times\) 2 blocks of pixels to quantize. For this design each pixel is assumed to be quantized to a single byte of data in the range \([0,255]\). The pixels are selected as shown in Figure 15. The transmitter consists of one or more Associative Processors (APs), some memory buffering for the line recall, some analog to digital converters, and a transmission unit. The vector quantizing part of the system is the Associative Processor unit. The APs can combine to yield a larger maximum codebook than is possible with one chip, and are the subject of the design discussion for the rest of this dissertation. The back-end section is responsible for transmitting the index into the codebook to the remote site in whatever manner is appropriate.

The front-end and back-end are also responsible for synchronizing the codebook if adaptive codebooks are desired. In such a system the front-end would have to
Figure 15: An example of how the pixels are taken out of the image to form the vector to be quantized.
continuously monitor the encoded data stream to monitor the errors being generated by the system. When those errors exceeded some criteria, the front-end would have to interrupt the data stream and send a new codebook to the receiver, and load the new codebook into the AP.

The remaining discussion in this chapter will cover the major functions and requirements of the functional blocks of the AP: the control logic, the data bus I/O interface, the cellular processors, the multichip extension circuitry, the address encoding circuitry, and the minimum distance detection circuitry.

### 3.4.1 Storage of Codebooks

As may be obvious, some facility must be made available to store the codewords in the AP. The requirement of stability and repeatability in the memory made a strong argument for a digital system. Since each of the subprocessors in this SIMD structure needed to access only one memory unit, the subprocessors (called cellular processors or CPs in future discussion) were equipped with a local memory bank. For this SIMD architecture the adaptability of the codebook was viewed as a major requirement, so the codewords had to be dynamically loadable into each of the cellular processors (CPs) in the AP. Since the size of the codebook was to be variable, a bit was added to the local memory store to indicate if the value in the CP's store was valid. If no valid data was in the CP the CP had to be able to disable itself during operation. This meant that 33 bits of data had to be stored in each CP: 32 for the codeword and 1 for a valid tag bit.

Each CP had to have its own address so that the codevector could be stored in
it. This address was very important to the system, since the address of the nearest matching codevector was the output of the AP.

3.4.2 The Operations of the CPs

The cellular processors are each responsible for calculating the distance between the input vector and the stored codeword, and for participating in the global minimum determination. To conserve space and insure maximum packing density, the CPs are designed such that they read in a single component (byte) of the input codevector at the time, then perform the subtraction between the input byte and the corresponding codeword byte. The absolute value of the difference is fed into an accumulator. After each of the components has been subtracted from the input code vector the result in the accumulator register is the sum of the absolute value of differences. There is a special circuit attached to the accumulator register which does a cooperative calculation with all the other CPs in the associative processor to find the minimum difference. If a given CP has the minimum difference, it drives its output high. The block diagram of the operation of the CPs is shown in Figure 16.

Other designs of the CP are possible. For example, a full word width subtractor followed by a tree of adders could have been fabricated, as shown in Figure 17. This CP architecture has the advantage of possibly being faster than the synchronous design above. Further, there is no state machine that has to be designed for the above system. The entire CP in this case (and the entire AP, if designed properly) can be made asynchronous. The benefits of asynchronous design are well known, as are the drawbacks. Asynchronous designs are notoriously difficult to integrate into
Figure 16: The block diagram of the selected CP function.
Figure 17: An alternate CP functional diagram.
systems unless they designed in a fully static, very careful fashion.

The chief disadvantage of the design in Figure 17 is that it does not scale well as the dimension of the codevector changes. For this demonstration system with a $2 \times 2$ vector of dimension 4, such a design could have been done. For use in higher dimensions this design rapidly consumes more area. The codeword storage area must grow linearly with increasing codeword dimension, but the area of the subtractor/adder combination will grow more than linearly as the dimension increases.

In contrast, the synchronous design has a memory region that must grow linearly as the dimension of the codevector increases. The second adder stage of the accumulator must grow logarithmically as the dimension of the codevector increases. The higher speed requirements for the synchronous design are not a major factor, even in the prototype done in $2 \mu m$. Since most commercial silicon for digital systems tends to be done in submicron fabrication facilities today, it would be even less of a factor in those faster technologies.

### 3.4.3 Minimum Difference Determination

Each of the CPs arrives at its distance measure independently, then cooperates in the determination of the minimum distance. The exact nature of the calculation will be given in Section 4.3.6, but it is based on a global wired-or calculation in which all CPs with valid data in them participate. If a given CP has the minimum difference it sets its output bit high, and these output bits are sent to an output encoder.

Since more than one CP can have the identical distance, some arbitration method must be used to select which CP had the minimum distance. The method chosen for
the prototype was to use the address of the first CP having the minimum difference. Many other methods could have been used to handle identical distances. For example, the AMD CAM chip uses a masking sequence to select each of the identical memories in sequence, starting with the lowest address and continuing up a chain until all identical memories have been identified. This was not deemed a necessary feature for this demonstration vehicle, but remains an intriguing feature to implement in future derivatives if an adaptive encoder requires the knowledge of all identical matches.

3.4.4 Control Functions

The overall function of the system depends on the coordination of the data bus broadcast with the function of each CP. There is a control block that must synchronize all the functions of the data bus and CPs: it must respond to reset signal, initiate load and match operations, and load the output registers of the chip. The state diagram for the control system is shown in Figure 22 in the next chapter, as are the detailed transitions.

3.5 Summary

This section has detailed the architecture for an associative processor to do vector quantization efficiently. The major points have been:

- The reasons for picking another distance measure, some of its mathematical properties and its behavior have been highlighted. For practical, area efficient implementation in a CMOS process a Minkowski metric with its relatively simple absolute value measurement was selected.
A SIMD architecture employing simple subprocessors was described. This selection was necessary to make the system fast enough to respond in real-time to a video stream. The calculations for the Minkowski metric are simple enough to make the subprocessor small, quick and efficient, but the shear volume of calculations meant that some parallelism must be employed.

The reasons why a digital design was done have been explained. For a design in which the comparisons involve more than about 9 digits in CMOS, digital designs are more efficient than analog ones, are far easier to design and more extensible.

A synchronous design was be done to provide the basis of a more extensible architecture. Synchronous designs such as this tend to require more interaction with the input section of the system design, but such interactions are simple and controllable by common, inexpensive microcontrollers.

The design has a loadable codebook and features that assist the codebook designers where such assists are feasible. The ability of the associative processor to load any codebook means that many different algorithms can be used to generate codebooks. We will not concern ourselves with the design of the codebooks other than to note that many different algorithms exist, and the ones that can be used for real time systems tend to use good, but suboptimal codebooks.
CHAPTER IV

Implementation of the Associative processor.

This chapter will deal with the implementation details of the Associative Processor (AP) described in the previous chapter. The full details of operation and function for all the component parts of the quantizer will be discussed, as will the overall function of the chip, and an example of the internal operations as the principle functions of the chip as sample operations are executed.

4.1 Detailed Architectural Overview of the AP

The associative processor (AP) is designed to be a highly parallel machine of the single instruction, multiple data (SIMD) type. Each associative processor is implemented as a single chip and consists of a control unit, I/O circuitry and several cellular processors (CPs). Each CP acts independently of the other CPs to do its distance calculations and only communicates with the other processors when the CPs communicate to find the minimum distance. SIMD machines are very common in the computer literature [33] and are the most common and best understood of the parallel computers available today.

The associative process was designed to have 32 cellular processors per chip. This was the maximum size that the limited fabrication budget could sustain. As such, it
was designed so that extensions to multiple chips could be easily accomplished so that a larger codebook could be efficiently used. The associative processor was designed with two distinct data busses and one control bus, as shown in Figure 18. There is an eight bit bus which serves as an input data bus. Its purpose is to broadcast a byte-wide segment of the input vector to all the cellular processors on a chip. The control bus coordinates the actions of the cellular processors and the data bus to do the distance calculation and forces a wait while the comparison to find the minimum distance is done. The match bus is wired in a wired-or manner, so all processors communicate on the match bus simultaneously. The CPs both read and write this bus, and this dual function makes a unique and very fast comparison operation.

The cellular processors take the individual components of the input vector and compare them with their locally stored components, which generates the sum of the absolute value of the differences (SAVD) between the components of the stored vector and the input vector. This SAVD difference is stored in a register and compared to the differences that all the other CPs have found. If a CP has the nearest match, it must broadcast this information to an address encoder, which will serve to decode the winning address. It is the CP that is the heart of the computational process. All other circuitry is just support circuitry to feed data and control signals to the CPs so that they can process the data in the correct order, to address a particular processor so that it can store data, and to encode the winning cellular processor's address.

The CPs have three basic modes: store, compare, and a reset function to initialize and clear all the cells. The reset function returns the CP to a state in which it does
Figure 18: The overall internal architecture of the Associative Processor. There are two unidirectional busses, the data bus and the control bus, and one bidirectional bus, the match bus.
Figure 19: The flow chart showing the functions performed while storing a memory word. All cellular processors perform the same function synchronously in an SIMD machine structure.

not participate in the finding of the minimum difference.

In the store mode, the associative processor stores a vector to an addressed memory location. The cellular processor chosen determined by the value loaded in the address register. This is the way in which a single CP is chosen to store the data on the data bus into that CP's local storage. Since only one processor can be addressed at a time, all the other CPs will ignore the value on the data bus.

Store mode is entered synchronously based on the status of the store pin. In the clock cycle beginning immediately after the store line is asserted, the value stored in the input register is written to the first byte in the addressed word. At the beginning of the next cycle, the second byte is stored, then the third and fourth. After all 4 bytes have been stored, the AP returns to its idle state, as shown in the Figure 19.

Compare mode starts with the assertion of the MATCH signal and begins the series of operations shown in Figure 20. After the MATCH signal is asserted, the value in the input register is subtracted from the first byte of the stored vector in all the CPs.
Figure 20: The flow chart showing the functions performed while comparing an input vector with a stored vector. All processors perform the same comparison synchronously. Note that the idle state is used to reset the accumulator that stores the sum of differences.
The absolute value of the result placed in the accumulator register. In the second clock period, the next value latched into the input register is subtracted from the second byte, and the absolute value is added to the accumulator. The process then repeats for the third and fourth bytes of the stored vector. Once the entire sum of absolute value of differences (SAVD) is generated, the minimum difference is found via a combinational circuit.

The circuit that does the minimum difference comparison is an analog combinatorial circuit, and as such tradeoffs have to be made between speed and power dissipation. Because these circuits read and drive the match bus with its associated high capacitance, correct sizing can be an art. The settling time of the combinatorial circuit cannot be determined \textit{à priori} because extension across chips is desired and there needs to be a method of extending the time period allowed for comparison. The exact implementation of this circuit is discussed in Section 4.3.6.

4.2 Associative Processor Interface Implementation

There are many control and buffering elements of the Associative Processor. There is a control unit that generates the global control signals that coordinate the actions of the CPs, an address decoder to select cellular processors in storage mode, an address encoder to encode the address of a processor having the minimum distance from the input vector in the match mode, a driver circuit to allow the expansion of the comparison operation to additional chips, and data bus drivers to broadcast values on the data bus. This section will deal with the implementation details, timing considerations and other factors of the global interface and control parts of the Asso-
ative Processor. The next section will deal with the implementation of the cellular processors.

### 4.2.1 The AP Global Control Subcircuit

We will first consider the control bus and its affects on the overall control scheme of the AP. Detailed control lines are not broadcast to all processors. Instead, each processor has some logic in it to decode the control lines. This was done to provide some redundancy, increase fault tolerance and to reduce the number of global signal lines to be routed. This section will describe the global signals that are broadcast to all the processors. The interpretation of the signals and the detailed decoding each cellular processor does will be described in Section 4.3.2.

The global control subcircuit contains a finite state machine that is the coordinator all of all the CPs. Its job is to signal the CPs what function is to be performed. As shown in Figure 21, the control circuit consists of a State Register consisting of 4 bits, a PLA to determine the next state and some buffer/driver circuitry. The control machine a Moore machine; the outputs only depend on the current state of the machine and are independent of the inputs. The only input for which this is not strictly true is the asynchronous \texttt{RESET} signal, which is connected only to the State Register and each of the cellular processors. The State Register is made of D flip-flops with asynchronous resets, to which the \texttt{RESET} signal is attached. The \texttt{RESET} line is not wired into the PLA since that is not necessary because this is a Moore machine. The \texttt{IDLE} state was chosen to be the state into which the control logic was reset, and the value of the state was chosen to be 0000. It is important that the
Figure 21: The high level layout of the control mechanism of the Associative processor. The control machine is a Moore machine, depending only on the current state stored in the State Register so there are no glitches on the output lines.
RESET signal be asserted for at least three clock cycles any time it is asserted because the CPs need the RESET signal stable for a minimum of two clock cycles, and the PLA state machine will need at least one to two to settle correctly, depending on the exact state of the clock when RESET is asserted. One output of the PLA generates a signal which resets the accumulator register, which means that a complete reset of the system requires a minimum of three clock cycles. The state diagram of the control machinery is shown in Figure 22.

The global signals that form the control bus of the AP are: store, accum, regclr, sel0 and sel1. These five signals are enough to control and regulate the CPs. For the load phase, only the store, sel0 and sel1 lines are used. The store signal is a global signal, so it must be AND-ed with an address line to uniquely identify a CP in which the vector is to be stored. The sel0 and sel1 lines mux the data bus into one of four eight-bit memory locations in the processor.

For the comparison operation, more control lines are used. The sel0 and sel1 lines are used to mux the memory cells into the computational element. The accum line is used to signal when the accumulator is active. The regclr line is used to reset the accumulator so that it starts at zero.

Synchronously on the rising clock edge after the SYSMATCH line goes high, regclr goes high, allowing the now cleared accumulator register in the cellular processor to begin accumulating the differences between the stored value in the accumulator. At the same time, both sel0 and sel1 are set to zero, to mux in the first byte of the

1 Asynchronous reset signals are notoriously difficult to deal with in system design, since register setup timings have to be satisfied and that is difficult with an asynchronous signal. In situations where a choice can be easily made, it is often better to design with synchronous resets.
Figure 22: The finished state machine diagram showing the state assignments and names. Any variables not explicitly mentioned are set to zero by the synthesis program. Also not shown is the reset mechanism, which is activated by the RESET signal and sends the state machine to the IDLE state asynchronously from any state it was in.
stored vector is subtracted from the value on the data bus. The absolute value taken and the value stored in the accumulator register. At the next rising clock edge, \texttt{sel0} goes high and the second byte is subtracted from the data bus. The absolute value taken and the result added to the value in the accumulator. At the next rising clock edge, \texttt{sel1} goes high, the third stored byte is subtracted from the value on the data bus and the result is added to the value in the accumulator register. On the next rising clock edge, the same operation is performed on the fourth byte, with \texttt{sel0} going low. After the sum of absolute value of differences (SAVD) is found, \texttt{accum} goes high and the accumulator register is not allowed to change.

After the SAVD is found, the comparison phase begins. At this stage, all the CPs begin communicating on the match bus. The details will be covered in another section, but the circuit is combinatorial and may take several clock cycles to complete depending on the system configuration. To accommodate this, there are two pins \texttt{wait0} and \texttt{wait1} which specify the number of “wait” states before the system returns to the \texttt{IDLE} state. During these states, \texttt{accum} is held high so that the accumulator is stable. Since \texttt{accum} signal overrides all other signals except \texttt{RESET}, the values of the \texttt{sel0} and \texttt{sel1} as well as the state of the data bus are not important. The state machine implemented is shown in Figure 22 and Table 4.

The state assignments are not random; they were in fact chosen using the \texttt{nova} state assignment program to minimize the logic in the PLA. This is important since the PLA is implemented in a pseudo-NMOS logic with pullups. Minimizing the transitions and the number of pulldown transistors used will lower the power require-
Table 4: The state transition table for the Control Finite State Machine. PS is the present state of the state register, and NS is the next state of the state register.

<table>
<thead>
<tr>
<th>Reset</th>
<th>Store</th>
<th>Find</th>
<th>Wait</th>
<th>PS</th>
<th>NS</th>
<th>Accum</th>
<th>RegClr</th>
<th>Mux</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>XX</td>
<td>XXX</td>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>XX</td>
<td>0000</td>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>XX</td>
<td>0000</td>
<td>0111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>XX</td>
<td>0111</td>
<td>0110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>XX</td>
<td>0110</td>
<td>0101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>XX</td>
<td>0101</td>
<td>0100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>XX</td>
<td>0100</td>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>XX</td>
<td>0000</td>
<td>1011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>XX</td>
<td>1011</td>
<td>1010</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>XX</td>
<td>1010</td>
<td>1001</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>XX</td>
<td>1001</td>
<td>1000</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>XX</td>
<td>1000</td>
<td>1111</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>00</td>
<td>1111</td>
<td>0000</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>01</td>
<td>1111</td>
<td>1110</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>01</td>
<td>1111</td>
<td>1110</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>00</td>
<td>1110</td>
<td>1101</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0X</td>
<td>1110</td>
<td>1101</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>10</td>
<td>1101</td>
<td>0000</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>1X</td>
<td>1101</td>
<td>1100</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>XX</td>
<td>1101</td>
<td>1100</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
ments of the chip. The input to nova to generate this state machine is shown in the Appendix, Section C.1.

This problem is small enough that an exhaustive search of optimal state assignments is possible. This means that the -e i e -r options can be given to nova. These options will cause nova to do an exhaustive search of all possible options. One constraint had to be applied to the solution, however. Since the PLA was to be wired to a register to form the finite state machine, and since the state register was to be reset by an asynchronous line reset operation that forces each of the state register elements to 0, the IDLE state had to be 0000. This was done to minimize the size of the PLA, since it was far more compact to reset the state register than to add the additional reset logic in the PLA. If the chosen assignment of the IDLE state not been met other options to nova would have had to be used to constrain the state assignment search. As it was, the minimum logic state that nova found was the one given in Table 5. Note the highly structured output of the design. The upper two bits of the state are always 01 during the time when a vector is to be stored in the AP. The upper two bits are 10 during the computational phase of the matching operation and 11 during the comparison phase of the matching operation. Such a structure suggests a good implementation for many of the control signals.

Once the optimal state assignments for this Moore machine was determined, the PLA had to be generated. This was done using the input deck shown in Appendix C, Section C.2. This deck is input to the OCTTOOLS program bdsyn to produce the logic equations for the circuit. The bdsyn program is invoked using bdsyn -t -Z -a.
Table 5: The optimal state assignments for a minimal PLA implementation as found by nova.

<table>
<thead>
<tr>
<th>State Name</th>
<th>State Value (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
<td>0000</td>
</tr>
<tr>
<td>LPATH0</td>
<td>0111</td>
</tr>
<tr>
<td>LPATH1</td>
<td>0110</td>
</tr>
<tr>
<td>LPATH2</td>
<td>0101</td>
</tr>
<tr>
<td>LPATH3</td>
<td>0100</td>
</tr>
<tr>
<td>MPATH0</td>
<td>1011</td>
</tr>
<tr>
<td>MPATH1</td>
<td>1010</td>
</tr>
<tr>
<td>MPATH2</td>
<td>1001</td>
</tr>
<tr>
<td>MPATH3</td>
<td>1000</td>
</tr>
<tr>
<td>MPATH4</td>
<td>1111</td>
</tr>
<tr>
<td>MPATH5</td>
<td>1110</td>
</tr>
<tr>
<td>MPATH6</td>
<td>1101</td>
</tr>
<tr>
<td>MPATH7</td>
<td>1100</td>
</tr>
</tbody>
</table>
The -Z option is used to set all unassigned options to zero. This is important, since the design implicitly assumes that variables not explicitly assigned are set to zero. For example, allowing the load signal to rise and become high during the compare cycle would corrupt the vector stored in the memory of the CP.

Once the logic equations are found, the octpla program is used to generate the PLA. In particular, by repeating the various options it was found that for a small PLA like this the option -nofold was found to generate a minimum sized PLA. The -nofold option prevents folding in the PLA. Folding often generates smaller PLAs for larger circuits, but in this small design it generates a larger PLA.

After the PLA was designed, it was converted from OCTTOOLS format to CIF, then from CIF to MAGIC format. During the conversion, various design rules were found to be violated and the resulting PLA was edited by hand to correct the errors. Since octpla also tends to put too few well plugs into the circuit, many were added by hand. Finally, the resulting PLA was extracted from MAGIC and simulated using irsim. Since octpla will often generate incorrect layouts, it is imperative to choose good test vectors. It is not possible to exhaustively test all inputs to the PLA in a reasonable amount of time; instead, all the expected operational states and paths are input to be tested. Once the correct function is assured for the design states, randomly chosen test vectors were chosen to determine that the correct operation was assured. The tests and their inputs decks are too large to be given here.

Once the PLA was designed and generated, the four D flip-flops that form the state register were connected. These D flip-flops were chosen to be the dfrf301
cells from the MSU standard cell library of the LagerIV toolset. These flop-flops feature asynchronous reset signals, a positive low reset signal, a \( D \) input and a \( Q \) output. The behavior and design of these cells is discussed in Section 4.2.2.

While the state register and the PLA form the functional heart of the control circuit, there is more circuitry in the control subcircuit to drive the outputs of the control subcircuit over the highly capacitive control lines. The way the circuit was designed, triple strength inverters were used to buffer the output of the PLA. Second stage drivers not included in the control block were 9x inverters (triple the triple strength inverters). The chain of inverters forms a minimum delay tree to drive the capacitance of the signal lines. This is important because of the relatively large capacitance of the signal lines and because it is important that the control lines be nearly synchronous with the data lines, or the chip will be needlessly slowed. The details of sizing are given in Section 1.2.3.

An exception to the buffering scheme of the PLA is the store signal. Because of the distribution pattern of the store signal, it was necessary for the store signal to be double buffered in the control block, and double buffered again on the distribution network. This is necessary because the capacitive load on the store signal is so high and because the signal routing required it.

### 4.2.2 The D flip-flops

There are two types of standard cell D flip-flops used in the design of the AP. Both are from the MSU SCMOS Standard Cell Library version 2.2. There is the dfrrf301 cell, which is an asynchronously resettable flip-flop with just \( Q \) for an output and \( D \)
Figure 23: The dfrf301 D flip-flop with an asynchronous reset and \( Q \) output from the MSU SC(MOS) library.

for an input. The diagram for the cell is shown in Figure 23. This cell is only used in the PLA where the state register must be resettable.

The dfnf311 cell is also from the MSU library. It is similar to the dfrf301 cell, but has no reset ability. It has both \( Q \) and \( \overline{Q} \) outputs, however, and is smaller than the dfrf301 cell. The smaller size means that it is used in more places where a resettable register isn’t required. The diagram for this cell is shown in Figure 24.
Figure 24: The dfm311 D flip-flop with both $Q$ and $\bar{Q}$ outputs.
Figure 25: The two basic methods for clock distribution: a clock chain (left) where all the buffers are of increasing size and a clock tree (right) where the buffers are the same size, but placed in parallel to reduce the loading on each buffer.

4.2.3 Clocking and Clock Drivers

Clock powering trees and clocking decisions are among the more difficult decisions for any chip. It is not uncommon on larger industrial designs to see teams devoted to designing clock trees for a given chip and a given technology. These teams must be very conscious of the technology for which they are designing and the designers are more aware of analog design considerations than typical digital designers.

There are two basic schemes for a clock distribution: clock trees or clock chains. These are shown in Figure 25. Both schemes work, but the clock tree is the more commonly chosen method of distributing a clock. It has the advantage that there are many more identical elements. These identical elements can be designed quite carefully. There are also advantages when automated placement is used, since better

\[\text{This also applies to any signal driving a net with a large fanout. For this chip, the data bus and control bus are also highly capacitive and need similar size and fanout considerations.}\]
optimization can be obtained with many small elements which can be allowed to float about on the chip. The disadvantage is that the clock tree must be carefully designed. If the clock tree is not carefully balanced there can be clock skew between logic blocks, and this can slow the maximum clock speed. Neither method is inherently superior; while the vast majority of chips are designed with clock powering trees, there are several examples of clock chains, also. Digital Equipment Corporation's Alpha microprocessor uses a clock chain where the final transistors have gate lengths more than 15 cm long in a 0.65 \( \mu \text{m} \) process [36].

For this particular chip, a mixture of the two methods was used. The tree structure was chosen, with five "leaf" nodes delivering the clock signal to a given row as shown in Figure 26. Relatively large buffer drivers were used to distribute the clock signal.
to a given row of cellular processors (CPs) on the chip. The capacitive load of a
given CP is quite high; there are 10 register clock inputs per CP, and the inputs tend
to have a large capacitance. This means that each row driver has to drive 50 input
gates, so the driver for a given row has to be large.

Since the row drivers were themselves relatively large, it is not efficient to drive
them directly with the signal from the clock input pin. Instead, it is more efficient
to buffer the signal in an inverter chain, sizing the transistors appropriately. The
number of stages and their sizes can be determined as follows: given a change of $N$
buffers, each slightly bigger than the preceding, a source resistance $R_0$ and a load
capacitance $C_L$ we get that

$$T = \sum_{i=0}^{N} R_i C_{i+1} = \tau \sum_{i=0}^{N} \frac{C_{i+1}}{C_i}$$ (4.1)

As you might guess, the minimum delay is obtained by setting the delay times for all
the stages equal. This means that

$$C_{i+1} = aC_i$$ (4.2)

so

$$T = \tau(N + 1)a$$ (4.3)

which means that

$$C_L = a^{N+1}C_0$$ (4.4)

where $C_0 = \tau / R_0$. We can write that

$$T = \tau(N + 1) \left( \frac{C_L}{C_0} \right)^{\frac{1}{N+1}}$$ (4.5)
To find the optimum number of stages, differentiate with respect to $N$. Set the result to zero and we find that

$$a = \epsilon$$

$$N + 1 = \ln \frac{C_L}{C_0}$$

and the total minimum delay is

$$T = \epsilon r \ln \frac{C_L}{C_0}.$$  \hfill (4.6)

For the estimated loads on this circuit, a four inverter buffering scheme was found to be optimal. The input pin was selected to drive a unit inverter, which fed to a 3$x$ inverter, which then fed 3 3$x$ (for a total of 9$x$ drive) inverters, which powered the row drivers and which were themselves 3$x$ the 3 3$x$ drivers (for a total of 27$x$). This clock powering tree is shown in Figure 26. The capacitive load for the row drivers were the accumulator registers of the CPs. There are five CPs in a row, and each CP has 10 NAND gates that gate the clocks in the modified D flip-flops that form the accumulator register in the CP. Since the gate capacitance of the clock gate is roughly half the “unit” inverter mentioned above, this is very nearly an optimal chain.

For the buffering chain/tree designed, the clock skew in simulations was less than 1 ns. Power dissipation was also moderate. With a 2 pF load for the row driver (which is higher than expected) the clock buffering subcircuit dissipated approximately 0.1 mW peak power. Even with loads as high as this, the additional delay for the clock signal is less than 2 ns. This 2 ns. additional time must be considered when the chip is wired into the system, since the additional skew will affect the last comparison
operation and must be included in any time that the comparison circuitry needs to achieve a stable solution.

It is important to note that clock skew introduced by the clock powering tree can both help and hinder the chip designer, but the chip designer must be aware of where the skew will manifest itself in the system/chip interface. For the chip designer, the clock powering time, $T_{cp}$, helps where the input to the chip registers is concerned since it means there is an additional $T_{cp}$ for the setup time on the input registers or pins. However, the same $T_{cp}$ is removed from the output registers, since the results of the calculations on the chip will be delayed by $T_{cp}$ because of the lag between the system clock and the chip clock.

Finally, clock skew is quite important internally on the chip and care must be taken in a single phase clock design such as the one employed on this chip. For example, consider the situation where register A is connected to register B via some combinatorial logic. If the logic is very fast, then clock skew between A and B may cause A to change and the logic may propagate that change to B before it is supposed to do so. This will result in the value for the next state being entered in register B, and some errors could occur. If the logic is slow, then clock skew can cause errors by violating setup times. If register B gets the clock signal before A, then the sum of the times for the combinatorial logic plus the register setup time may be longer than the clock period between register A and register B because the clock skew has shortened the path. The two cases are shown in Figure 27.

The clocking for the AP is single phase, edge triggered logic. All clocking is
Figure 27: The problems caused by an edge triggered register with clock skew. In the left case, the combinatorial logic has not settled on a new value by the time the clock edge arrives, so the value in the register doesn't change. In the right case, the value in the A register is processed by the time the B register is ready, so the value that should have been there from the time period before is lost.
triggered on the rising edge of the clock signal. The skew between CPs on the AP is not important, since the clock is not a factor in any of the inter-CP communication. The clock is just a counter to say when the inter-CP communication is finished.

There are two distinct places where clock skew can be a problem in this particular implementation of the AP architecture, and in particular in the CPs used in this design. The first is in the accumulator registers. The accumulator register feeds back to itself via the combinational circuit that performs the SAVD. There can be problems if there is sufficient skew between register elements that the SAVD of the wrong vector components is loaded into the accumulator. The likelihood of this is reduced by the relatively large gate area that each modified D flip-flop presents to the clock. Since there is less statistical variation among the larger area transistor gate capacitances, the likelihood of excessive clock skew is reduced.

The other area where clock skew can be a problem for the CP is in the bus driver circuits and the system clock. If the input register can broadcast its value on the data bus before the clock signal arrives, then there can be erroneous data in the CP registers. Care must be taken that the data bus drivers charge more slowly than the clock distribution.

4.2.4 The Address Decoder

To provide the addressability required to store the vectors in a cellular processor, some mechanism is required to address the individual CPs. This addressing is much the same as in a conventional memory, but the decoder is designed so that no processor is addressed when the chip is not having a vector stored. This is an important safety
feature since it insures that a processor is not randomly addressed during a glitch.

The decoder is built from two enabled 4-to-16 decoders, a five bit address register and some NAND gates. The enabled 4-to-16 decoders are the DEC5 circuits from the MOSIS CMOSN Standard Cell Library. These decoders are more fully described in Section 4.2.5. The decoders take four inputs and an enable line, producing the value on the output line that is the decoded value of the address on the input lines, provided the enable line is held low. When the enable line is high, the output of the decoder is zero.

To make a 32 bit decoder, two of the DEC5 decoders are used. The MSB of the address register is NANDed with the store signal from the AP state machine to form the enable signal for the DEC5 cells. For the lower 16 lines, the enable signal is formed by taking $Q$ from the MSB of the address register and NAND-ing it with the store signal. The result is a signal that is low when the MSB is low and the store signal is low. Similarly, to form the enable signal for the upper 16 lines, the $\bar{Q}$ from the MSB of the address register is NANDed with the store signal. The address decoder is shown in Figure 28.

The address value is stored in a five bit register made of dfn311 D flip-flops. These flip-flops, like all those used on this chip, are trailing edge triggered, so the data valid on the rising edge of the clock pulse is locked and stable during the entire cycle of the memory address. There are an odd number of inverters in the clock powering chain, so while the individual flip flops are trailing edge triggered, the chip as a whole seems to be leading edge triggered because of the odd number of inverters. See Section 4.2.2
Figure 28: The address decoder circuitry for the Associative Processor. The address lines are buffered in the address register.
for more details on the register elements. Unlike the dfrf301 flip-flops, these are not resettable. This is not important for the operation of the circuit since the decoder requires both the SYSSTORE pin, the PLA and the value stored in the address input latch to do operations. Because of the trailing edge triggered nature of the latch, the address must be stable the period before the operation starts. The address must also be held steady throughout the storage operation unless partial vector addressing is to be used.

4.2.5 An Enabled 4:16 Decoder

The DEC5 cell is a 4-to-16 line decoder from the MOSIS CMOSN Scalable CMOS Library version 1.1. The cell has an enable signal which is active low. When the enable signal is high, none of the outputs of the DEC5 circuit will be high. When the enable signal is low, the output line corresponding to the decoded value of the input lines will be high. This result is summarized in Table 6.

The functional diagram of the decoder is given in Figure 29. The circuit is a collection of 16 four input ANDs, two ANDs and eight inverters. The ANDs are implemented as NANDs followed by inverters. The speed of this particular circuit is not terribly important since it is a part of the storage operation, which is much faster than the matching operation. The output lines of this decoder are AND-ed with the STORE signal to load a value into a CP. The loads for this particular circuit have no fanout, and tend to be dominated by the capacitive value of the lines and vias leading to the target CP.
Table 6: The behavior of an enabled 4:16 decoder. EN is the enable line value, C3-C0 the input lines, and DF-D0 the output lines.

<table>
<thead>
<tr>
<th>EN</th>
<th>C3-C0</th>
<th>DF-D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XXXX</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>0</td>
<td>0001</td>
<td>0000000000000010</td>
</tr>
<tr>
<td>0</td>
<td>0010</td>
<td>0000000000000010</td>
</tr>
<tr>
<td>0</td>
<td>0011</td>
<td>0000000000000100</td>
</tr>
<tr>
<td>0</td>
<td>0100</td>
<td>0000000000001000</td>
</tr>
<tr>
<td>0</td>
<td>0101</td>
<td>0000000000010000</td>
</tr>
<tr>
<td>0</td>
<td>0110</td>
<td>0000000000100000</td>
</tr>
<tr>
<td>0</td>
<td>0111</td>
<td>0000000001000000</td>
</tr>
<tr>
<td>0</td>
<td>1000</td>
<td>0000000100000000</td>
</tr>
<tr>
<td>0</td>
<td>1001</td>
<td>0000000100000000</td>
</tr>
<tr>
<td>0</td>
<td>1010</td>
<td>0000001000000000</td>
</tr>
<tr>
<td>0</td>
<td>1011</td>
<td>0000001000000000</td>
</tr>
<tr>
<td>0</td>
<td>1100</td>
<td>0001000000000000</td>
</tr>
<tr>
<td>0</td>
<td>1101</td>
<td>0001000000000000</td>
</tr>
<tr>
<td>0</td>
<td>1110</td>
<td>0100000000000000</td>
</tr>
<tr>
<td>0</td>
<td>1111</td>
<td>1000000000000000</td>
</tr>
</tbody>
</table>
Figure 29: A functional diagram of an enabled 4-to-16 decoder.
4.2.6 The Winning Address Encoder

There are 32 CPs that comprise a single AP and the output of the AP is the address of the CP with the lowest sum of absolute value of differences (SAVD). The values stored in the CPs are independent of one another, so there is a chance that two or more CPs could come up with the same SAVD. In this case, the CPs with the lowest SAVD would all signal that they have the lowest SAVD. A conventional 1-of-N encoder would incorrectly encode the winning value(s); in fact, the exact value of the output would depend on the value of the input lines and the design of the encoder.

The address encoder for the AP was designed with an arbitration mechanism for multiple matches. While multiple matches are unlikely, they can occur, so the mechanism that encodes the winning signal must provide some method to single out one of the possible winners. In the case of identical differences the training mechanism can be designed so that it does not favor one codebook vector over another (an example is the codebook generated by the FSCL algorithm), so picking any of the identical vectors is acceptable.

With these conditions, the simplest possible arbitration method was chosen: the first processor with the minimum difference has its address encoded in the output buffer. This means that a 32 line priority encoder is required. A 32 line priority encoder can be designed and is a simple extension of an eight line encoder, but such a circuit is slow and large. Instead, the simpler and faster method of combining four eight bit priority encoders and some glue logic was used to produce a 32 line encoder. Speed is relatively important for this element, since the delay introduced by encoding
the lowest SAVD address must be added to the comparison time that the CPs need to settle. This value must be added to the setup time required for the output register, so a small delay in the encoder is important.

The eight bit priority encoder used was the Pri8 eight bit priority encoder from the MOSIS CMOSN Standard Cell Library. The cell is quite large, but performs the desired operation quickly. The behavior of the Pri8 cell is discussed in Section 4.2.7. It is important to note that an eight bit priority encoder has four outputs: a line indicating the validity of the operation (i.e. that at least one of the eight input lines was high) and three lines encoding the address of the line (if any) that went high. The encoding scheme used is that \( Q_2, Q_1, Q_0 \) are the encoded address line of the first input line which is high, running from the most significant bit to least significant bit. \( Q_3 \) indicates that at least one of the input lines were high, and that the output value in \( Q_2-Q_0 \) is valid; if \( Q_3 \) is low then none of the input lines were high.

To connect the four priority encoders, a modified priority encoder must be designed. The modified priority encoder must take the output of all four Pri8 encoders and select which address is to be encoded.

Before discussing the design of the modified priority encoder some notation must be developed. \( Q_{12} \) is the \( Q_1 \) line of the third priority encoder; the numbering scheme will begin at zero to be consistent with the numbering scheme of the priority encoder lines. Since not all lines have the same meaning, and since the priority encoders have to be arranged in a priority encoding scheme themselves, the following equations must
be satisfied:

\[
E_0 = Q_0 \cdot Q_3 + Q_0 \cdot \overline{Q_3} \cdot Q_1 + Q_2 \cdot \overline{Q_3} \cdot \overline{Q_1} \cdot Q_3 + \\
Q_0 \cdot \overline{Q_3} \cdot \overline{Q_1} \cdot \overline{Q_2} \cdot Q_3
\]

\[(4.9)\]

\[
E_1 = Q_1 \cdot Q_3 + Q_1 \cdot \overline{Q_3} \cdot Q_1 + Q_2 \cdot \overline{Q_3} \cdot \overline{Q_1} \cdot Q_3 + \\
Q_2 \cdot \overline{Q_3} \cdot \overline{Q_1} \cdot \overline{Q_2} \cdot Q_3
\]

\[(4.10)\]

\[
E_2 = Q_2 \cdot Q_3 + Q_2 \cdot \overline{Q_3} \cdot Q_3 + Q_2 \cdot \overline{Q_3} \cdot \overline{Q_1} \cdot Q_3 + \\
Q_2 \cdot \overline{Q_3} \cdot \overline{Q_1} \cdot \overline{Q_2} \cdot Q_3
\]

\[(4.11)\]

\[
E_3 = Q_3 + Q_3
\]

\[(4.12)\]

\[
E_4 = Q_3 + Q_3
\]

\[(4.13)\]

\[
E_5 = Q_3 + Q_3 + Q_3 + Q_3
\]

\[(4.14)\]

These equations describe the function required. \(E_0\) to \(E_4\) are the address of the first line that has a high logic value. \(E_5\) signals that there was at least one input line that was high; if \(E_5\) is low, then the value on the \(E_0\)–\(E_4\) lines is not valid. These equations are not a minimal implementation. There is a tremendous amount of redundancy in the equations above, so use of an optimizer is strongly suggested.

Various design styles and techniques were compared in this implementation. Since Eqs 4.14 are just combinatorial expressions, the \texttt{bdsyn} synthesis program from the OCTTOOLS toolset can be used to automate circuit generation. Once \texttt{bdsyn} had generated the BLIF data, the \texttt{misII} synthesis/optimization program from the OCTTOOLS toolset was used to produce the standard cell layout. A PLA could have been used to decode the \(Q_{ij}\), but a standard cell layout turned out to be smaller and
faster in this case. Eqs 4.14 were encoded into a standard bdsyn compatible format, as shown in Appendix C, Section C.3. In this configuration, the misII program can synthesize this circuit easily, although the result will have a large amount of redundancy in the resulting circuit, making it larger than it needs to be. To reduce the cell count and optimize the circuit the misII program needs some additional hints. To provide those additional hints, the program requires something that does more than the standard msu.script file distributed with the OCTTOOLS toolset does. The script in Figure C.3.2 in Appendix C was used to perform algebraic simplification of the circuit. This is the most powerful form of decomposition and yields the smallest circuits, but it is very time consuming. Since this circuit is small enough, an exhaustive method like this is computationally acceptable.

It is imperative to run a simulator on the output of any optimized, automatically synthesized circuit. misII will frequently drop states or insert invalid states during the optimization phases, so generation of good test vector coverage is also very important. The verification phase of the misII program does a cursory examination of the true (valid high) parts of the logic; it does not check the behavior of the circuit when the output is false. This means that there may be additional high states induced in the output of the circuit even if all went well in the circuit synthesis. This is the case in this instance. As is shown in Table 7, irsim indicates that the circuit functions as expected, but there are additional high states induced in the circuit. These additional high states are not important in this instance because they occur when E5 is low. Since E5 is the signal that shows that a valid match has found, if it is low no match
has occurred and the result of the other 5 bits can be ignored.

The output of the misII program was further constrained to produce a two row design, rather than to produce a square design. This design used more area, but because of other floor-planning considerations it fit far better into the design of the AP.

The final part of the encoder is a register consisting of dfn311 D flip-flops triggered off the reghold signal of the PLA. When the reghold signal goes low, the value of the output encoder is latched into the output register until the next next match is found. Section 4.2.2 provides more details on the register cells, but it important to note here that the output register is a trailing clock edge triggered register. This means that the time to complete the comparison phase of the calculation is

\[ T_{\text{out}} = T_{\text{compare}} + T_{32 \text{ priority decode}} + T_{\text{register setup}}. \]  

where \( T_{\text{out}} \) is the time from the last SAVD operation, \( T_{\text{compare}} \) is the time to compare all the accumulator registers, \( T_{32 \text{ priority decode}} \) is the time to decode the 32 input lines from the CPs, and \( T_{\text{register setup}} \) is the setup time required for the output register.

### 4.2.7 The Eight Input Priority Encoder

The eight input priority encoder used in the address encoder came from the CMOSN standard cell library available from MOSIS. The function of the encoder is to take eight input lines and encode them into four output lines. The inputs are assigned priority such that the lowest numbered line that is high is the value encoded on the output. If none of the lines is high, then the highest order output line is not asserted.
Table 7: The address encoder synthesized behavior. Note that bit order is opposite that typically used.

<table>
<thead>
<tr>
<th>Input Vector (v00 to v1f)</th>
<th>Output Vector (en0 to en5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000000000000000000000000000000110</td>
<td>000110</td>
</tr>
<tr>
<td>1X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>00001</td>
</tr>
<tr>
<td>01X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>10001</td>
</tr>
<tr>
<td>001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>01001</td>
</tr>
<tr>
<td>0001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>11001</td>
</tr>
<tr>
<td>00001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>00101</td>
</tr>
<tr>
<td>000001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>10101</td>
</tr>
<tr>
<td>00000001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>01101</td>
</tr>
<tr>
<td>0000000001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>11101</td>
</tr>
<tr>
<td>000000000001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>00101</td>
</tr>
<tr>
<td>00000000000001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>10011</td>
</tr>
<tr>
<td>0000000000000001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>01011</td>
</tr>
<tr>
<td>000000000000000001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>11011</td>
</tr>
<tr>
<td>00000000000000000001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>00111</td>
</tr>
<tr>
<td>0000000000000000000001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>10111</td>
</tr>
<tr>
<td>000000000000000000000001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>01111</td>
</tr>
<tr>
<td>00000000000000000000000001X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X</td>
<td>11111</td>
</tr>
</tbody>
</table>

Note: The bit order is opposite that typically used.
Table 8: The function table for an 8 bit priority encoder.

<table>
<thead>
<tr>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
<th>D6</th>
<th>D7</th>
<th>Q3</th>
<th>Q2</th>
<th>Q1</th>
<th>Q0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The function table for the circuit is given in Table 4.2.7. Unlike the priority encoder designed above, all outputs are zero when all inputs are zero. Q3 is high when any input is high, and Q2-Q0 is the binary coded output of the input lines.

The functional diagram of the Pri8 encoder is given in Figures 30-32.

4.3 Implementation of the Cellular Processor

The cellular processor (CP) is designed so that it responds to a minimum of global control signals and to perform most functions independently of all the other CPs. The CP has several tasks. First, it must store the vector in a memory location. It must store and read this memory in a byte-wide manner. Second, the processor must take the absolute value of the differences between the stored bytes and the broadcast byte on the data bus, as controlled by the control bus lines. Thirdly, these absolute value of differences must be accumulated until the vector summation is finished. Finally,
Figure 30: The functional diagram of the $Q_3$ and $Q_2$ part of the Pri8 encoder.
Figure 31: The transistor level diagram of the Q0 encoder of the Pri8 encoder.
Figure 32: The transistor level diagram of the Q1 encoder of the Pri8 encoder.
the all the differences must be compared to and broadcast on the match bus lines.

The overall scheme is that the CPs work as elements in a SIMD machine with several distinct options. There is a state in which the processor is addressed and a memory element stored. In this case only one of the processors is active at a time. The other state is when all the CPs are comparing their stored vectors to the input vector. This second state has two distinct phases: the computation of the sum of absolute value of differences (SAVD) between the input vector components and the stored vector components, and the comparison of all the input vectors to the other input vectors.

We begin the discussion by considering the memory storage design for each CP. The second item is the exact decoding of the global control bus is described so that the local control signals are generated. The third item is the circuit that finds the SAVD and its accompanying accumulator register. Finally, the circuit that finds the minimum distance between the elements is discussed.

4.3.1 Memory Implementation

To store the vector in the processor, the circuit used in Figure 33 is used. The memory block subcircuit shown in Figure 33 is replicated eight times, one block connecting to each of the data bus lines. This means that each block shown will contain the same $n^{th}$ bit of the input vector. For example, all of the 5$^{th}$ bits of a stored vector would be stored in just one of the circuits shown in Figure 33. The memory block I/O is connected to the data bus when the load signal is asserted, turning on an NMOS pass transistor that connects the data bus line and the memory
Figure 33: The circuit that stores a given bit of each of the four components of the stored vector. For example, this circuit might be used to store the MSBs of each of the four byte-wide components of the stored word.
block. The memory cells are also sensitive to the `load` signal, knowing that they are to read the value on their I/O line if `load` is asserted, and to write to the I/O line if it is not.

Each memory block subcircuit has four pairs of coupled inverters to store the same bit in each of the bytes that make up the stored vector. No matter whether the I/O line to the memory block subcircuit is to be either an input or an output, there signal must travel through a bidirectional multiplexor circuit. At this stage, there is only one data path. Whether the value on the data path is the output of the memory cell or the input to the memory cell depends on whether or not `load` is asserted.

To store values, each coupled inverter must have a signal that breaks its feedback path and allows the value on the data bus line to load the data bus line value into the memory location. While the `load` signal determines the direction of the data flow into or out of the memory, the particular bit of the nibble to be written must be selected by a combination of the global `sel0`, `sel1`, and `load` signals and the local `address` signal. As shown in Section 4.3.2, the local load signals `load0`, `load1`, `load2` and `load3` are generated to select which bit in the storage nibble is written. The `loadx` signals are generated for each processor. This was done to avoid having an extremely large capacitive load on the signals, to reduce the number of global lines and to provide some fault tolerance, since it allowed for some processors to work even if some were damaged by poor fabrication, etc. This cell is shown in Figure 34.

Some care was also taken in designing the memory nibbles to make them relatively space efficient, since they take up a sizable fraction of the area of the CP, as shown
Figure 34: The bit storage mechanism used. The input line is bi-directional. It serves as the input if \texttt{load} is high, otherwise the line has the output of the memory location and feeds into the arithmetic unit.
Table 9: The sizes of the various elements that make up the cellular processor. These sizes are the ratios of the area each occupies on the floor-plan of the chip and provide clues as to where optimizations are possible.

<table>
<thead>
<tr>
<th>Cellular Processor subblock function</th>
<th>% of floor-plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 bit vector memory</td>
<td>16</td>
</tr>
<tr>
<td>Arithmetic computation (SAVD)</td>
<td>32</td>
</tr>
<tr>
<td>Accumulator register</td>
<td>18</td>
</tr>
<tr>
<td>Control bus decoder</td>
<td>4</td>
</tr>
<tr>
<td>Comparison circuit</td>
<td>18</td>
</tr>
<tr>
<td>Signal routing and dead space</td>
<td>12</td>
</tr>
</tbody>
</table>

in Table 4.3.1.

Full CMOS switches were used on the output data path to provide full voltage swing, drive, and speed for the SAVD operations. For input, PMOS transistors were used as switches to gate in the value on the data bus. This is a reasonable choice since the writing data path is not in the critical timing path of the processor; there is more than ample time for the input of the memory element to change. In fact, a slowly changing input will reduce the chance of error from clock skew between the input register, its drivers and the control signals. The data path from the memory to the arithmetic unit is much more critical, however. Because the setup time required to pass the value from the memory to the unit is a factor in the critical timing path, it is very important to have good drive in the output of the memory elements.

As mentioned before, the muxes that connect the memory cells to the data bus and the arithmetic unit must be bidirectional. This is a requirement since the data on the bus must be an input to the memory cell at times and the memory cell must
Figure 35: A bi-directional mux that connects the four bits of the memory storage to the arithmetic unit and/or the data bus. When load is low, the signals flow from the memcells to the arithmetic unit, when load is high, signals flow from the data bus to the memcell selected by the load signal.

Output to the arithmetic unit at other times. The mux in this implementation was a set of six full CMOS switches, as shown in Figure 35. The direction of the signal flow is dependent on the mode the system is in. If data is being stored, the memory control subsection generates the load signal if the the particular CP is addressed and the the AP is in store mode (see Section 4.3.2). If load is high, the selr lines are decoded to determine which memory cell is to be loaded. See Figure 36 for the signal flow when load is asserted. For use in matching operations, the load signal is not asserted to the CP and the signal flow is as shown in Figure 37.
Figure 36: The signal flow when loading the memory elements for a given cellular processor depending on the state of the control machine. Care must be taken here to control the load signal since it controls the directional flow of data into or out of the memory.
Figure 37: The signal flow when a CP is being used in a match operation. The signal flow from the memory is only valid for the first four clock cycles, after which the flow of data from the memory bank and bus lines is not important for the functioning of the CP.
For proper operation, it is important that the user store the one’s complement of the vector into the memory locations. The reasons for this are discussed in Section 4.3.3 on the arithmetic unit.

4.3.2 Memory Control Subsection

The memory control subsection does local signal decoding for memory control from the global memory control signals and maintains the processor valid signal storage. These functions were performed locally for several reasons: they give some fault tolerance to the system, simplify the design of the global signal drivers and lower the number of global lines.

To control the memory each CP has a control bus decoder in which the global load signal is combined with an address line that runs to only one cellular processor from the address decoder. When both load and address are high, the processor is selected to store the values that will be presented on the data bus. The processor generates a local signal, store, using the equation

$$\text{store} = \text{load} \cdot \text{address}.$$  \hspace{1cm} (4.16)

The store signal is fed into the S (store) input of an SR flip-flop. This flip-flop is used to signal that data has been stored in this particular processor. The Q output of this SR flip-flop forms the valid signal that is fed into the most significant bit of the minimum difference circuit. This is critical to the memory comparison phase of the associative processor’s function, since only those processors with valid vectors stored in them should participate in the selection of the minimum distance number.
The CPs without valid data still perform all the processing just as the ones with valid data, but they cannot participate in the minimum difference determination.

The global \texttt{RESET} signal is connected to the R (reset) input of the SR flip-flop, and serves to mark the CP as having invalid data. Since the \texttt{RESET} signal is global, all processors are reset to having no valid data signaled. Further, the vector memory locations are \textit{not} affected by the \texttt{RESET} operation; only the the SR flip-flops that indicate the validity of the data in the processor is affected. Because there is no power on reset function in this implementation, it is important that the \texttt{RESET} signal be asserted for not less than three clock periods after powerup. Further, the \texttt{RESET} signal must be released not less than the register setup time $T_{\text{setup}}$ before the rising edge of the clock to assure that the proper state is entered after the reset period. Failure to do this will leave the AP in an unknown state, and all operations using the AP will be suspect, at the least.

There is nothing that prevents the data stored in the CP from being overwritten by subsequent memory accesses. If this happens, the valid data flag is still set, and the new vector is stored in the 32 bit vector memory area. Conversely, there is no way to erase a location and mark it as invalid without resetting the entire AP. For this reason, care should be taken to store codebooks of the same size, since using codebooks of varying size can be difficult.

To control the memory locations that store the vector elements, the local \texttt{store} signal is combined with the global \texttt{sel0} and \texttt{sel1} to form \texttt{load0}, \texttt{load1}, \texttt{load2}, and \texttt{load3}. The value of the \texttt{sel} lines and the store signal combine as follows to generate
The signals that allow the value on the data bus to load into a selected memory location.

\[ \text{load0} = \text{store} \cdot \overline{\text{sel0}} \cdot \text{sel1} \]  
(4.17)

\[ \text{load1} = \text{store} \cdot \text{sel0} \cdot \overline{\text{sel1}} \]  
(4.18)

\[ \text{load2} = \text{store} \cdot \overline{\text{sel0}} \cdot \text{sel1} \]  
(4.19)

\[ \text{load3} = \text{store} \cdot \text{sel0} \cdot \text{sel1} \]  
(4.20)

The use of these \text{load}r signals is described in Section 4.3.1. These signals control into which memory nibble in the byte-wide storage the value on the bus is to written. Since the loads into the memory are byte-wide and four deep, there have to be four lines to control which bit in the nibble is written. Quite simply, these lines can be thought of as combining the value on the \text{sel} lines and the \text{store} line to select the appropriate bit in the memory nibble associated with the bit position in the memory. The logical implementation of this decoding structure is shown in Figure 38.

4.3.3 The Arithmetic Unit

The arithmetic unit does the calculations that are the heart of the cellular processor. It has 3 inputs and one output, as shown in Figure 39. The inputs are from the byte-wide data bus, the byte-wide output of the memory, and the 10 bit-wide input from the accumulator register. The output is the sum of the absolute value of the difference between the the components stored in memory and the input vector, and is stored in the accumulator register.

The arithmetic logic works by adding the one's complement value stored in the
Figure 38: The local decode structure for each CP. This takes the global load, store, sel0, sel1 and local address line and creates the local control signals for the CP.
Figure 39: The computational part of the cellular processor. There are two banks of eight bit ripple carry adders, four banks of eight bit wide memory, eight XNORs, some additional combinatorial logic for the second bank of adders, and a ten bit accumulator register.
memory byte to the byte on the data bus. Since the values stored are two’s complement numbers, the addition of a one’s complement to another value can be transformed into a subtraction. The details of this mathematical function can be found in Appendix B.

To do the subtraction of the stored vector component from the component broadcast on the data bus there is a bank of eight full adders connected in a ripple-carry configuration. The carry-in value of the first full adder is hardwired to zero. The value of the overflow carry of the last is quite important and determines the operation of the second bank of eight full adders. If the overflow carry does not exist, the result is a negative number (in the two’s complement sense). In this case, the result of the subtraction must be negated and because the absolute value is required, the sign bit is dropped. If the overflow carry does exist, it must be added to the least significant digit to get the correct difference. The result of this operation is the absolute value of the difference between the image to be stored and the input data byte. To accomplish this subtraction, the overflow carry is checked and XNORed with the output of the adders, and the result of the XNOR is fed to the second bank of full adders. Rather than perform an additional addition, the carry-out of the MSB of the first bank is connected to the carry-in of the LSB of the second bank of full adders.

The second bank of full adders consists of eight full adders and a combinational circuit (see Section 4.3.4 for details) that behaves much like two half adders to handle the upper two bits. This is because the input to the second bank of adders is the byte-wide output of the first bank of adders and the 10 bit wide input of the accumulator.
This means that the input to the ninth bit is just the carry-out of the eighth bit and the input of the ninth bit of the accumulator. Similarly, the input of the tenth bit is the just the carry-out of the ninth bit and the input of the tenth bit of the accumulator register. Since there can be no overflow, the carry circuitry for the tenth bit of the register isn't required.

This particular circuit determines the clock cycle time of the Associative Processor. The minimum cycle time of the clock is the time that it takes to take the output of the memory bank, feed it to this circuit, have the value on the data bus subtracted and the result added to the accumulator. The worst case delay for this circuit is sixteen full adder delays, plus some extra time for the combinational circuit at the end of the second bank of adders. For this particular design of a full adder, the delay time per stage is approximately 1.2ns as determined by SPICE simulation. To this, the setup time of the accumulator must be added, approximately 6ns. This means that estimated clock period is approximately 30ns.

4.3.4 The Arithmetic Unit endcap Circuit

The last two bits (numbers 8 and 9) of the second stage of the arithmetic unit are slightly different from the other adders in the chip. They are nearly half adders, since the output to the accumulator register position A8 and A9 are

\begin{align*}
A8 &= A8 \oplus C_{T_{\text{out}}} \quad (4.21) \\
A9 &= A8 \cdot C_{T_{\text{out}}} + A9 \quad (4.22)
\end{align*}
Figure 40: The circuit that calculates the values for the $A8$ and $A9$ values of the accumulator register.

where $C_{7_{out}}$ is the carry out of the eighth bit of the second bank of adders and $A8$ and $A9$ are the ninth and tenth bits stored in the accumulator register. The upper two bits of the SAVD are different because there are no inputs from the absolute value of the difference of the two components that are being subtracted. The only input is from the value stored in the accumulator register previously. The other important thing to note is that since we have only four eight bit differences to add, there will never be a carry output that would make an $A10$ required, so there does not need to be a full half adder for the $A9$ value. The logic circuit for this particular circuit is shown in Figure 40.
4.3.5 The Accumulator Register

The accumulator register in the cellular processor consists of a bank of ten modified D flip-flops. Because the accumulator register feeds back onto itself through the second bank of adders, the register must be made from flip-flops. For ease of implementation, a D flip-flop design was chosen. Since the chip was designed to have only single phase clocking, a Triggering mechanism consistent with all the other registers was chosen. The accumulator is a trailing edge triggered design, although the odd number of inverters in the clock powering tree makes the register seem to be leading edge triggered at the chip level.

Several other properties were designed into the accumulator register. The register had to be resettable, but unlike all the other registers on the chip, this one has a synchronous reset. For the accumulator registers there is a special signal out of the AP control logic, the regclr signal, which synchronously resets the accumulator. The accumulator registers are reset whenever the system is in the IDLE state between comparison operations and load operations.

The accumulator register must also be able to hold its present value as long as desired. This is important, since there is a programmable number of wait states during which the minimum difference must be found, so the final value of the sum of absolute value of differences stored in this register may not change during any of the comparison cycles. The mechanism chosen to implement this was to gate the clock with the accum signal from the AP control machine. When accum is high the value in the register follows its input like a normal D flip-flop. When accum is low, the
value loaded into the register is maintained. The accum signal has lower priority than the regclr signal, so the register will clear whenever both are asserted. Since the clock was already gated for this particular subcircuit, the additional delay induced by adding another input to the NAND that gates the clock was minimal.

The gate level logical design for the register is shown in Figure 41. Ten of these register elements were placed in parallel to form the ten bit accumulator register that the design required. As can be seen, the design is trailing edge triggered and each memory element gates the clock.
The use of ten elements in parallel is not an optimal design in terms of area, but it made implementation easier. It would have been better to gate the C signal input to all the register elements just once. This would have made the design of the clock tree (see Section 1.2.3) significantly easier and saved area. As currently implemented the accumulator register takes up 18% of the area of the CP, so optimization here is clearly possible.

4.3.6 The Minimum Difference Subcircuit

This is arguably the most interesting and important piece of the cellular processor and the associative processor. It is the method whereby the CPs communicate with each other and which CPs have the minimum sum of absolute value of differences.

Before discussing the implementation details, consider how the minimum SAVD would be found. A human considering a column of binary numbers, some valid and some not, would begin by deleting all the invalid numbers. The human would then look at the MSB of all the numbers, and if any number had an MSB of 0, the human would delete all the entries with a 1 in the MSB. Of course, if they all had an MSB of 1, then they would all still be valid and have to be compared. Having reduced the search set somewhat (unless all the MSBs were 1 and all the numbers valid), the human would consider the next most significant bit, MSB−1. The human would again look at the column of valid numbers and if any number had a zero in MSB−1 all numbers with a 1 in MSB−1 would be crossed out. If all the entries had a 1 in MSB−1, they would all still be valid. This procedure would continue until the LSB is encountered, at which point the minimum SAVD would remain.
Table 10: The transition table for the minimum difference circuit for a given bit position. See the text for a description of what all the values are.

<table>
<thead>
<tr>
<th>$V_{in}$</th>
<th>$B$</th>
<th>MATCH$_n$</th>
<th>MATCH$_n$</th>
<th>$V_{out}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>Z(X)</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Z(0)</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Z(1)</td>
<td>1</td>
</tr>
</tbody>
</table>

This procedure will yield the minimum number in a set of numbers. There will always be at least one solution to this procedure. There may be many repetitions of the minimum, but there will be only one minimal value. There is some inherent feedback in the calculation of this minimum, but that feedback will drive the system into a completely stable state. The state table of a decision like this might be something like that in Table 4.3.6. Table 4.3.6 is rather difficult to read and the notation is complicated. Consider a bit $N$ in a word of length $M$. For the $N$th bit, $V_{in}$ signals that the bit is to be considered in the minimum difference calculation. $B$ is the value of the bit and MATCH$_n$ is the value on the match bus for this $N$th bit. $V_{out}$ and MATCH$_n$ are the outputs of the circuit, with $V_{out}$ being the valid (not "crossed out") signal for the next most significant bit, $N-1$, and MATCH$_n$ being the value on the MATCH$_n$ bus line. This calculation starts out with the MSB and continues through all $M$ bits in the word.

The states in Table 4.3.6 are in the typical notation, where $X$ means don't care, 1 and 0 mean themselves, but the $Z$ notation is somewhat different. The $Z(0)$ state
in the third line means that this particular element has a high impedance output to the **MATCH**<sub>n</sub> bus line, but some other CP is driving the bus line low. Similarly, **Z(1)** means that this element has a high impedance output to the **MATCH**<sub>n</sub> line, but since no other element is drawing the **MATCH**<sub>n</sub> to ground, the line stays high. **Z(X)** means that the state of the **MATCH**<sub>n</sub> line is unaffected by this particular element and cannot be further specified.

There is only one instance in which this element will drive the **MATCH**<sub>n</sub> line: when **V<sub>n</sub>** is 1 and **B** is 0, this element will draw **MATCH**<sub>n</sub> low. In all other cases the element is a passive listener to the **MATCH**<sub>n</sub> bus line. But when **B** is 0, the circuit both writes to **MATCH**<sub>n</sub> by setting it to 0 and reads the value on the **MATCH**<sub>n</sub> line to propagate **V<sub>out</sub>** to the next stage.

Inherent in this discussion has been the assumption that the **MATCH**<sub>n</sub> line has been driven by some external entity, and that that entity drives the line high unless some element pulls the line low. This is typical of a wired-OR bus and is used in the design of this particular subcircuit. Figure 42 shows how such a circuit can be implemented. The elements that comprise this circuit are some attendant logic that drives a pulldown NMOS transistor. This minimum difference circuit is meant to be cascaded, with each stage getting **V<sub>n</sub>** from the **V<sub>out</sub>** of the previous stage. The **V<sub>n</sub>** for the MSB of the word in the accumulator register comes from the **VALID** bit stored in the control decoder for each CP (see Section 4.3.2 for details). If there is valid data stored in the CP, the **V<sub>n</sub>** signal for the MSB is true, otherwise the **V<sub>n</sub>** signal is false and the CP doesn't participate in the minimum difference determination.
Figure 42: The circuit to detect the minimum difference. The circuits are meant to be cascaded, with $V_{\text{out}}$ for bit $N$ being used as $V_{\text{in}}$ for bit $N-1$. 
The output to the minimum difference circuit and the entire CP is the $V_{out}$ signal of the last minimum difference subcircuit. If this $V_{out}$ is high, then this particular CP has one instance of the minimum difference and this (high) signal is passed to the 32-bit priority encoder that decodes the address of the minimum difference (see Section 4.2.6 for more details on the priority encoder).

The design of the internal logic gates for the minimum difference circuit is straightforward, although the last stage should have a higher output drive capability to drive the capacitance from the last stage to the priority encoder. The difficult part is the sizing of the pulldown NMOS transistors and the pullup PMOS transistors for the MATCH bus. Since this is a wired-OR calculation and is to be done quickly, it is an analog design problem. Because of the large capacitance of a global line like the MATCH line, the speed of the calculation comes from making the pullups and pull-downs large, but must be balanced with power dissipation considerations. Further, the relative sizing of the NMOS to the PMOS transistors is critical to producing good noise margins, as shown in Appendix D.

The concatenation of many of these subcircuits together to form a single minimum difference detector is shown in Figure 43.

4.4 Extensions To Multiple Chips

As shown in Section 4.3.6, the MATCH bus is a wired-OR circuit. This is notoriously difficult to extend to multiple chips without requiring a properly sized external resistor to be used as a pullup and analog I/O pads to properly deal with the MATCH lines. The analog I/O lines add capacitance and electrostatic sensitivity to the system.
Figure 43: An example of an accumulator register feeding a row of minimum difference subcircuits. The inputs to the minimum difference row are the values in the accumulator register, the valid data stored bit from the control subsection of the CP, and the values on the MATCH bus. The output is the $V_{out}$ signal for the LSB of the accumulator and is fed to the 32 bit priority encoder.
Figure 44: The circuit to split the I/O of the wired-OR internal line into an input and output part.

An alternate scheme is to use the extension scheme shown in Figure 44. In this scheme each chip has its own pullup resistors placed internally so that analog I/O is not required. There are twice as many I/O pins required, however, since the inputs are now decoupled from the outputs.

For each line in the MATCH bus in Figure 44 there are now two I/O pins, a MATCH\textsubscript{inn} and MATCH\textsubscript{outn} pin. The MATCH\textsubscript{outn} pin buffers the value currently on the MATCH\textsubscript{n} line and puts it on the output pin. The MATCH\textsubscript{inn} pin takes the value on the input pin and applies it to an N-channel pulldown transistor. This means that
if the input value is a 1, the MATCH line will be pulled down, just as if a CP had pulled that line down.

Using this logic, the circuit that connects the APs together must perform a NAND of each MATCH line to form the MATCH signal. In this way the logical function of the minimum difference circuit presented in Section 4.3.6 can be extended across several APs to form one very large AP. An example of four APs connected together to form a single AP with a maximum codebook size of 128 words is shown in Figure 45. The decoder to find which of the APs had a winning value is optional and can be done as the 32 bit priority encoder described in Section 4.2.4, or an alternate scheme can be used.

The drawback to a scheme such as this is that several extra delays are introduced by the extension mechanism. There is the delay of the buffering of the signal to the pad, the inter-pin delay to the NAND gate, the delay of the external NAND gate, the delay back to the AP input pin, the double buffering to the pulldown transistor, and delay of the pulldown to lower the MATCH line. Some of these delays are quite small, while the delay going from the chip to the external NAND and back can be large and unpredictable. A reasonable estimate for the delay can be found by taking the delay for the pads plus several inverter delays, so the approximate delay introduced by these extensions is 15 ns. For a single chip system there are no extra delays introduced in the system, since the inputs to the extension will be wired to ground and never affect the AP.

An interesting feature that this method of extension of the MATCH bus across
Figure 45: An example of connecting 4 associative processors together to form a single AP with a maximum size of 128 code words. The connections for the tenth and first bits are shown; the other connections are similar.
many chips is that it makes the minimum difference value available to the system. At the very end of the comparison cycle, the value on the MATCH bus is the value of the minimum difference that the system found. This could be useful in a system which does real time adaptive VQ, since this difference could be latched and the error between the input vector and the stored vector found.

4.5 Operational Considerations For Using The AP

This section will bring together some of the design features of the associative processor as implemented. These are meant to be considered as cautions for the use of this particular implementation of the architecture and are not meant as replacements for the detailed descriptions of the components. The desire is to pull the behavior of the components together to make the overall operation of the AP more clear.

4.5.1 Resetting The AP

To reset the AP the RESET pin must be asserted for a minimum of three clock cycles. This is because the reset function is an asynchronous signal and must be held stable for a minimum of two complete clock cycles. The RESET pin must not be released a dff301 setup time, $T_{\text{setup}}$ or approximately 5 ns., before the rising clock edge (see Section 4.2.2 for details on the flip-flops used in the system state register).

The RESET function does not physically erase all the memory locations. Instead, it does two things. First, it resets the internal system state register to the IDLE state and halts all the operations inside the chip. Second, it clears the bit that says that a vector has been stored in a particular cellular processor. It clears this bit in all the
CPs. There is no way to clear the value stored in a single CP; all the CPs must be cleared for one to be cleared, so using a codebook of fixed or expanding size is more efficient.

The AP has no power-on reset function, so the AP must be reset before it is to be used or the contents of the CPs and the system state register will be unpredictable.

4.5.2 Storing Vectors

To store a vector, the SYSSTORE pin is asserted, the one's complement of the first byte component of the vector to be stored is placed on the data pins, and the address of the processor that will store the vector is asserted on the address pin. At the next rising clock edge the AP will move from the idle state to the LPATH0 state, where the value in the data input register will be stored in the first vector component storage location of the processor specified. At the end of the next clock cycle the system will be in the LPATH1 state, and the second component will be stored. This process repeats for each of the components of the vector. It is imperative that the one's complement of the vector be stored for proper operation of the system.

The value of the SYSSTORE pin is not important after the first rising clock edge. The AP assumes that the next three clock cycles will have the appropriate components of the vector to be stored. There is no way to stop this operation once started unless the RESET pin is asserted, at which point the storage operation will stop and the entire AP will be reset.

The address register is rewritable on any given clock cycle, so it is imperative that the operator pay close attention to the internal address that is given during a store
cycle. In particular, the recommended mode of operation is that the address is to be held stable for the entire storage operation.

When a value is stored in any CP, that CP is marked as having valid data. That bit cannot be removed without resetting the entire AP. It is valid, however, to rewrite a memory location and to change the vector stored there. This means that while old vectors cannot be removed, they can be rewritten.

4.5.3 Comparison Operations

A comparison operation starts out much like a storage operation. To begin a comparison operation, the SYSMATCH pin is asserted and the first component of the target vector is set upon the data pins. At the rising clock edge, the system moves from the IDLE state to the MPATH0 state. During the first clock cycle, the absolute value of the difference between this vector component and all the other first vector components is calculated and stored.

At the next rising clock edge, the second component of the vector must be present on the data lines and the system will move the MPATH1 state. During this clock cycle the absolute value of the difference between the second component of the vector and all the stored vectors will be found and added to the difference found in the first clock cycle. This operation repeats two more times until the differences between all four components have been calculated and summed.

After the differences have been calculated, the system begins to compare differences. This can go on from 1 to 4 clock cycles and is selectable based on the value on the WAIT1 and WAIT0 pins. At the end of the wait cycles, the lowest address of
the cellular processor with the minimum difference will be output on the address out
pins. The most significant bit designates if an actual match was found. If the MSB
is not a 1, then the rest of the output has no significance.

The value of the SYSMATCH pin is not important after the first rising clock edge.
Just as in the case of the storage mode, once the processor has begun the comparison
operation it expects to be fed data in a very specific order and it expects all data to
be stable at the times required by the AP. There is no way to stop this operation
once started unless the RESET pin is asserted, at which point the storage operation
will stop and the entire AP will be reset.

4.5.4 Clocking Considerations

There will be a maximum speed at which this circuit can operate. It is expected that
this maximum speed will be determined by the speed at which the absolute value
of the difference between the components can be calculated and added to the preset
sum of absolute value of differences.

There should be no minimum speed at which this chip can operate. All internal
logic is static, so there is no decay of stored data. The buffering of the clock signal
by the clock drivers means that the clock transitions internally should be very sharp,
so except for pathological clocking schemes there should be no minimum clock speed.

With this design, if any one particular computation or clock phase needs to be
increased the external clock can be safely gated and that particular clock period can
be extended. For example, if more than 4 wait states are required, clock stretching
could be employed to increase the amount of time that was given to the comparison
phase of the calculation.

4.6 A Sample Operation

In this section we will explore the behavior of a single AP used in a VQ system. Examples of how the AP is loaded and how it behaves when asked to compare an input vector will be given. The first section will cover how to load the memory. The second section will cover the matching operation in detail.

For this example a single AP of the type described in the previous sections will be used. This means that all the MATCH\textsubscript{inn} pins must be grounded. The codebook will have four codewords:

$$C = \begin{cases} 
(58, 93, 115, 249) & C_0 \\
(254, 231, 125, 187) & C_1 \\
(99, 141, 175, 247) & C_2 \\
(166, 17, 162, 164) & C_3 
\end{cases} \quad (4.23)$$

$$c = (8, 54, 172, 112) \quad (4.24)$$

$C$ is the set of codewords, $C_0$...$C_3$ are the codewords themselves and $c$ is the input vector for which we want to find the nearest codeword in $C$. Each codeword $C_i$ is to be loaded into the corresponding $i^{th}$ cellular processor. The other 28 processors will not be loaded.

4.6.1 Loading the AM with four vectors

To begin the process, the AP must be reset. As explained previously, to reset the system the \texttt{RESET} pin must be asserted for at least three clock periods. While the values of the \texttt{SYSSTORE} and \texttt{SYSMATCH} pins are not important during this time, it

\textsuperscript{3}The codewords were randomly generated using a generator that passes the spectral test.
is best to leave them low during this time so that the AP does not begin processing before we are ready.

After the AP has been reset, the storage operation begins. Imagine that we wish to store $C_3$ first. After the rising edge of a clock at time $\phi_0$ the value 00011 is put on the address pins, 01011001 (the one's complement of the first component of the $C_3$ vector, 166) on the data pins and the SYSSTORE pin is asserted.

At the beginning of $\phi_0 + 1$ the values on the data pins are latched into the data register and broadcast on the data bus to all the processors. The address of the processor in which $C_3$ is to be stored is latched into the address register, decoded and the $a03$ line asserted for storing the data. Since the SYSSTORE pin was asserted, the AP controller has gone from the IDLE state to the LPATHO state. This means that the $se10$ and $se11$ lines are both low, but the store signal to all the processors is high.

The cellular processors also become active during $\phi_0 + 1$. All the CPs see that the store line is high, but only CP$_{03}$ has its address line, $a03$, high, so only it stores the value on the data bus into its memory. Seeing both $a03$ and store high, CP$_{03}$ sets its VALID bit, which enables it to do compares later. CP$_{03}$ chooses the location of the byte to be stored by decoding the $se10$ and $se11$ lines, and seeing that both are zero it places the value on the data bus into the first memory cell of the nibble storage.

During time $\phi_0 + 1$ the values for the next component of $C_3$, 00011 for the address and 11101110 (the one's complement of decimal 17) for the data pins, are asserted.

---

$^4$Time will be measured in clock cycles beginning after the low to high transition, or rising edge, of the clock.
After the storage operation has actually started, the state of the SYSSTORE pin doesn't matter, so the pin can be released, if desired.

At time $\phi_0 + 2$ the values for the second component of $C_3$ are read into the data and address registers and the results broadcast on the data bus and decoded in the address register. The AP controller has gone from state LPATH0 to LPATH1, which means that both $sel_0$ and $store$ are high, and $sel_1$ is low. CP03 is again the only CP active during this time. It sees the $sel$ lines and knows that the value on the data bus is to be stored in the second memory cell of the array of memory locations.

Also during $\phi_0 + 2$ the values for the third component of $C_3$ are readied on the input pins, where 01011101 (the one’s complement of decimal 162) and 00011 are the values for the data and address pins, respectively.

During $\phi_0 + 3$ the third component data value and address value are read into their respective registers. The address register is decoded and $a03$ remains high. The data bus is set to the value in the data register. The AP controller goes from state LPATH1 to LPATH2 and all of $sel_0$, $sel_1$ and $store$ are high. CP03 is the only active processor, and seeing the $sel$ lines it knows to store the value on the data bus in the third memory cell of the array of memory locations.

While the third component is being stored, the values for the fourth component, 01101100 (the one’s complement of 147) for the data lines and 00011 for the address lines, are asserted.

When time has reached $\phi_0 + 4$, 01101100 is put into the data register and 00011 into the address register. The AP controller goes from state LPATH2 to LPATH3,
where both store and sel1 are high, and sel0 is low. This means that 01101100 is stored in the fourth and final memory cell of the array of memory locations in CP03.

At time $\phi_0 + 1$ it is important to realize that the state machine which controls the storage will return to the IDLE state, so the value on the address and data pins is no longer important. This return-to-IDLE particular design means that a maximum of 32 cycles could be wasted when storing data in the entire chip serially, but made for easier simulation and synthesis.

When $\phi_0 + 5$ finally rolls around the system is in the IDLE state and the next vector, $C_0$ for example, could be stored. To store the first component of this vector, 000000 would be placed on the address pins and 11000101 (the one’s complement of 58) would be placed on the data lines, and SYSSTORE would be asserted. The next five clock periods would be used to store the $C_0$ vector just as was done for $C_3$ in the previous example.

### 4.6.2 A sample matching operation

Consider now that the set of $C$ vectors have been loaded into their respective cellular processors and that we wish to initiate a compare operation between all the vectors in $C$ and the vector $c$.

We begin the comparison at time $\phi_1$. At this time we assert the SYSMATCH pin and place the value of the first component of $c$, 00001000 (decimal 8), on the data pins.

At the beginning of $\phi_1 + 1$, which begins on the rising clock edge, the value of decimal 8 will be read into the data register and broadcast to all the CPs. The master
Table 11: The accumulator values at the end of the first clock cycle of the comparison operation.

<table>
<thead>
<tr>
<th>CP Num.</th>
<th>Prev. Accum.</th>
<th>$c_0$</th>
<th>$C_{i1}$</th>
<th>New Accum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>58</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>8</td>
<td>254</td>
<td>246</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>8</td>
<td>99</td>
<td>91</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>8</td>
<td>166</td>
<td>158</td>
</tr>
</tbody>
</table>

AP controller will go from the IDLE state to MPATH0. In the MPATH0 state, the first component of $c$ is subtracted from the first component of each of the $C_i$ stored vectors and the absolute value taken. In the IDLE state the accumulator register was cleared in each of the CPs, so the value in each accumulator register will just be the absolute value of the difference between the first component of $c$ and $C_i$ as shown in Table 11 and Figure 46(a).

While these values are being generated during $\phi_1 + 1$ the value of the second component of $c$, 00110110 (decimal 54), is asserted on the data pins. As it was with the SYSSTORE pin after the storage operation began, the value of the SYSMATCH pin does not matter after the match operation has begun.

During $\phi_1 + 2$ the second component of $c$, decimal 54, has been read into the data register and broadcast to all the CPs. The master AP controller has gone from state MPATH0 to MPATH1. During MPATH1 the second component of $c$ is subtracted from the second component of each of the $C_i$ vectors, and added to the value stored in each of the accumulator registers. At the end of $\phi_1 + 2$ the newly computed value is stored
**Figure 46:** The values of the values on the signal lines of the four Cellular Processors involved in the sample operation just before the rising edge at $\phi_1 + 2$ (a) and at $\phi_1 + 3$ (b).
Table 12: The accumulator values at the end of the second clock cycle of the comparison operation.

<table>
<thead>
<tr>
<th>CP Num.</th>
<th>Prev. Accum.</th>
<th>( c_0 )</th>
<th>( C_{ix} )</th>
<th>New Accum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>50</td>
<td>54</td>
<td>93</td>
<td>89</td>
</tr>
<tr>
<td>1</td>
<td>246</td>
<td>54</td>
<td>231</td>
<td>423</td>
</tr>
<tr>
<td>2</td>
<td>91</td>
<td>54</td>
<td>141</td>
<td>178</td>
</tr>
<tr>
<td>3</td>
<td>158</td>
<td>54</td>
<td>17</td>
<td>195</td>
</tr>
</tbody>
</table>

in the accumulator, as shown in Table 12 and Figure 46(b). During \( \phi_1 + 2 \) the value of the third component of \( c \), 10101100 or 172 decimal, is asserted on the data pins, also.

For the time period \( \phi_1 + 3 \), the third component of \( c \), 172, is latched into the data register and broadcast to all the CPs. The master AP controller has gone from MPATH1 to MPATH2. The CPs will decode the AP controller signals and subtract the third components of their stored vectors from the value broadcast on the data bus, take the absolute value and add it to the current value in the accumulator register. The result will be stored into the accumulator at the end of \( \phi_1 + 3 \). The results are shown in Table 13 and Figure 47(a). During \( \phi_1 + 3 \) the value of the fourth component of \( c \) is placed on the data pins, 01110000 or 112 decimal.

During \( \phi_1 + 4 \) the fourth and final component of \( c \), 112 decimal, is read into the data register and broadcast on the data bus to all the CPs. The master AP controller has gone from state MPATH2 to MPATH3 and the CPs decode the AP controller signals to subtract the fourth component of their stored vectors from the value on the data
Table 13: The accumulator values at the end of the third clock cycle of the comparison operation.

<table>
<thead>
<tr>
<th>CP Num.</th>
<th>Prev. Accum.</th>
<th>$c_0$</th>
<th>$C_{in}$</th>
<th>New Accum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>89</td>
<td>172</td>
<td>115</td>
<td>146</td>
</tr>
<tr>
<td>1</td>
<td>423</td>
<td>172</td>
<td>125</td>
<td>470</td>
</tr>
<tr>
<td>2</td>
<td>178</td>
<td>172</td>
<td>175</td>
<td>181</td>
</tr>
<tr>
<td>3</td>
<td>195</td>
<td>172</td>
<td>162</td>
<td>205</td>
</tr>
</tbody>
</table>
Figure 47: The values of the values on the signal lines of the four Cellular Processors involved in the sample operation just before the rising edge at $\phi_1 + 4$ in (a) and $\phi_1 + 5$ in (b).
Table 14: The accumulator values at the end of the fourth clock cycle of the comparison operation.

<table>
<thead>
<tr>
<th>CP Num</th>
<th>Prev. Accum</th>
<th>$c_0$</th>
<th>$C_n$</th>
<th>New Accum. (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>116</td>
<td>112</td>
<td>219</td>
<td>283 (0100011011)</td>
</tr>
<tr>
<td>1</td>
<td>470</td>
<td>112</td>
<td>187</td>
<td>545 (100010001)</td>
</tr>
<tr>
<td>2</td>
<td>181</td>
<td>112</td>
<td>247</td>
<td>316 (010011100)</td>
</tr>
<tr>
<td>3</td>
<td>205</td>
<td>112</td>
<td>164</td>
<td>257 (0100000001)</td>
</tr>
</tbody>
</table>

bus, take the absolute value of the difference, add it to the previous value in the accumulator register and store that value in the accumulator register at the end of $\phi_1 + 4$. The results are shown in Table 14 and Figure 47(b).

During $\phi_1 + 5$ the value on the data pins is not important. This is because the AP controller has gone from state MPATH3 to MPATH4 and raised the accum signal to all the CPs. When this signal is raised, the accumulator register in the CPs all hold their current value until the accum signal is lowered. At this point, the CPs begin to compare the values in their accumulator registers with each other using the wired-OR MATCH bus. This particular operation is combinatorial and because of the extensions to multiple chips described in Section 4.4 requires an unknown number of delays, there may be multiple clock cycles required to complete the operation. For this description, it is assumed that only one clock cycle will be required to generate a minimum difference.

At the beginning of $\phi_1 + 5$ the comparison phase begins. While the comparison circuitry has been active during all the computations, it is not until the beginning
of $\phi_1 + 5$ that the final sum of absolute value of differences is stable and the address output register is enabled for loading.

To begin the computation, all CPs are involved. First, each CP evaluates its VALID data bit and determines if the data stored in its memory and accumulator register are to be used. In this example, CP04 through CP1A did not have data entered into their memory elements, so they propagate an invalid token to the combinatorial circuit associated with the BIT9 bit of their accumulator registers. This invalid token will be propagated throughout the entire chain, from BIT9 to BIT0 and to the address encoder, so CP04 through CP1A are disabled throughout the rest of the calculation.

CPs 0-3 have all had valid data written to them, so they have valid tokens (the VALID signal from the SR flip-flop is high). The BIT9 comparison circuitry takes this valid input token and uses it to enable its comparison operation. Since the circuitry is valid for all 4 CPs, all 4 examine their values. CP00, CP02, and CP03 see that they have BIT9 values of 0, so they pull the MATCH9 line from its initial value of 1 to 0 and propagate the valid token to the BIT8 comparators.

CP01, which didn’t drive the MATCH9 line, sees that the MATCH9 line has dropped, that it has a 1 stored in its accumulator and thus propagates an invalid token, a 0, to its BIT8 comparator. This invalid token will be propagated through its entire register and finally to the address encoder. This means that CP01 is no longer able to be marked as a winner.

Once the BIT9 comparison circuits have settled, the BIT8 comparison circuits begin to settle. Since only CP00, CP02 and CP03 have valid input tokens for the
BIT8 comparators, only they can affect the MATCH8 line and possibly pass valid token on to the BIT7 comparison circuits. While CP01 BIT8 comparator would have pulled down the MATCH8 line if it had had a valid token, it does not and thus does not participate in the calculation. In this particular example, all the valid CPs see that they have 1's stored in the BIT8 bit of the accumulator register. This means that none of the pulldown transistors in the comparison circuit are activated. The comparison circuit for each CP, seeing that the MATCH8 line is high and that each accumulator register has a 1, passes the valid token on to the BIT7 comparators.

The BIT7 comparators notice that only CP00, CP02 and CP03 have valid data. In each case, the CP notices that it has a valid token and a 0 stored in the BIT7 bit of its accumulator register. This causes the comparison circuit to turn on the pulldown transistor, pulling MATCH7 low, and propagate a valid token to the BIT6 comparators.

The BIT6 comparators act exactly as the BIT7 comparators did, and again CP00, CP02 and CP03 have valid tokens.

The BIT5 comparators taken the input tokens, and CP03 and CP01 note that they have valid tokens and zeroes stored in their BIT5 bits in their accumulator registers, so they pull MATCH5 low. CP02 see that it has a valid input token, a 1 in its BIT5 bit, but that MATCH5 has gone low, so it propagates an invalid token to its BIT4 comparator.

For the BIT4 comparators, only CP00 and CP03 have valid input tokens. CP03 sees that it has a 0 in its BIT4 accumulator register bit, so it pulls MATCH4 low. CP00,
seeing its valid input token, the 1 in its BIT4 accumulator bit and the low value on the MATCH4 bus, propagates an invalid token to its BIT3 comparison circuit.

At this stage, only CP03 still has a valid token. This means that only CP03 can determine what the value will be in the MATCHi lines for $i = 3 \ldots 0$, and that the values of the MATCHi lines will be the value of the equivalent CP03 bit. In other words, CP03 will continue to activate the pulldown transistor whenever a 0 is stored in the accumulator register bit, and will allow the MATCHi line to stay high whenever the accumulator register bit is high.

Once all the match lines have settled and at least one CP has a valid token\(^5\), then the address encoder takes over. The address encoder looks at the token outputs of all the CPs. It takes the first CP and encodes its address. In this case, only CP03 had a valid token at the end of the computation, so the address encoder places 100011 into the output register. The MSB in the output means that there was at least one CP with a minimal difference, and the 00011 means that the address of the first CP with that minimal difference was at address 03. The final state is shown in Figure 48.

### 4.7 Summary

This chapter has shown a detailed view of the architecture implemented for the Associative Processor described earlier. The inner details of the working elements have been described, and a sample operation has been given.

---

\(^5\)There may be more than one CP with a valid token because there will be points equidistant from each pair of points. Section 4.2.6 considers the details of the mechanism that selects which minimum difference is selected.
Figure 48: The values of the values on the signal lines of the four Cellular Processors involved in the sample operation at the end of the matching operation.
CHAPTER V

Prototype Analysis

This chapter will detail the results of the fabrication and testing of the prototype associative processor described in the previous chapter. The prototype AP is designed mainly to show that the architecture is feasible.

5.1 Design Results

A prototype of the associative processor was fabricated in 2μm CMOS using the MOSIS fabrication service. The prototype had 32 cellular processors, and was packaged in a 65 pin Pin Grid Array (PGA) package. The overall floor plan is shown in Figure 49 and the layout is shown in Figure 50. Most of the layout for the chip was done by hand using the MAGIC package from the University of California at Berkeley and Digital Equipment Corporation, as described in the previous chapter. Simulations were run with IRSIM and SPICE3e2 to determine the various operational points.

The overall architecture is very efficient in terms of some of the more costly elements of CMOS design. The chip only used a total of 59 pins, of which 45 were signal I/O pins, and the balance were power and ground pins. The pins and their uses are shown in Table 5.1. The use of 2 sets of pins for address in and address out was somewhat redundant since their functions could easily have been combined, but
Figure 49: The floor plan of the prototype AP.
Figure 50: The layout of the prototype AP.
Table 15: The number and type of the pins used in the Associative Processor.

<table>
<thead>
<tr>
<th>Pin Function</th>
<th>Pin Direction</th>
<th>Pins Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector component data</td>
<td>in</td>
<td>8</td>
</tr>
<tr>
<td>Match data output</td>
<td>out</td>
<td>10</td>
</tr>
<tr>
<td>Match data input</td>
<td>in</td>
<td>10</td>
</tr>
<tr>
<td>System reset</td>
<td>in</td>
<td>1</td>
</tr>
<tr>
<td>STORE function</td>
<td>in</td>
<td>1</td>
</tr>
<tr>
<td>MATCH function</td>
<td>in</td>
<td>1</td>
</tr>
<tr>
<td>System clock</td>
<td>in</td>
<td>1</td>
</tr>
<tr>
<td>Wait state delay</td>
<td>in</td>
<td>2</td>
</tr>
<tr>
<td>CP address in</td>
<td>in</td>
<td>5</td>
</tr>
<tr>
<td>Nearest CP address</td>
<td>out</td>
<td>5</td>
</tr>
<tr>
<td>$V_{dd}$</td>
<td>in</td>
<td>7</td>
</tr>
<tr>
<td>$V_{ss}$</td>
<td>in</td>
<td>7</td>
</tr>
</tbody>
</table>

Combining the functions would have saved only 5 pins and made debugging marginally more difficult. Since I/O pins tend to be among the more expensive and failure-prone parts of chip fabrication, minimizing the number of I/O pins has a positive effect on the overall yield of packaged chips, and thus on the chip cost, as well as the cost of assembling circuit boards. Minimizing the number of pins used in a chip makes the board more easily fabricated by any number of vendors.

Increasing the number of pins used on the AP would complicate the internal design of the AP somewhat, but would significantly simplify system design. For example, loading all the elements into the AP at once would simplify the external system design and would reduce the attention and parsing required of the external control logic at the relatively small cost of adding 24 additional pins, 24 additional registers.
Table 16: The sizes of the elements making up the AP and their contributions to the overall size of the AP.

<table>
<thead>
<tr>
<th>Component</th>
<th>Size</th>
<th>% of AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input data latching</td>
<td>1500 x 344</td>
<td>6</td>
</tr>
<tr>
<td>Input address decoder</td>
<td>98</td>
<td>1</td>
</tr>
<tr>
<td>Nearest address encoder</td>
<td>522 x 1456</td>
<td>4</td>
</tr>
<tr>
<td>MATCH bus extensions</td>
<td>1006 x 138</td>
<td>1</td>
</tr>
<tr>
<td>AP controller</td>
<td>456 x 513</td>
<td>2</td>
</tr>
<tr>
<td>Cellular Processors</td>
<td>706 x 1034</td>
<td>62</td>
</tr>
<tr>
<td>Routing and dead space</td>
<td></td>
<td>22</td>
</tr>
</tbody>
</table>

...to store the additional vector components, and some additional logic to gate the correct signals onto the data path. This would have caused the prototype to exceed its limited budget, but for a commercial application where there would likely be more cellular processors on a given chip, this would have little affect on the price of the system.

The AP itself was fabricated in a 2 μm, single layer polysilicon, double layer metal CMOS process by Orbit Semiconductor using the MOSIS fabrication service. The area of the finished design was 6545 x 6852 μm and the I/O pads were distributed about three sides of the chip. The sizes of the various components used in the construction of the AP are given in Table 5.1. Some comments about the size of the elements used should be made here. Many of the components of the chip listed could be redesigned to be smaller. Their current size is due mainly to the speed required for real-time video operation, and to the fact that some standard cells were used from other sources. If a faster technology is used, the size of the components listed could...
shrink considerably.

Most affected by the use of a faster technology would be the cellular processor. Its current size is quite large so that operations can be completed at clock rates of 33 MHz or higher according to simulations. Further, the register in the CP is larger than necessary and could easily be consolidated into a smaller area that is better behaved in fabrication.

As Table 5.1 shows, the AP makes relatively efficient use of the area allocated. It is important to point out that this architecture scales very well as the number of CPs per chip increases. The control logic of the AP does not change as the number of CPs increases on the AP, while both the input CP address decoder and the nearest-match CP address encoder only grow logarithmically with the increase in number of CPs. This means that incorporation on a larger die of more CPs will increase the relative fraction of area used by the AP for calculation.

The AP also scales well with the increase in the dimension of the vector presented to the AP. For example, an eight component vector AP would require only six additional I/O pins to extend the MATCH lines for multi-chip resolution support. Additional pins to address the CPs inside the AP would only be required if additional CPs were placed on the chip, and the number of address lines would grow logarithmically with the number of CPs on the chip.

The CPs themselves would also scale quite well with increasing dimension. For the eight byte component vector mentioned earlier the memory area would approximately double, but the SAVD subcircuit would only add one additional component. This
means that the overall growth of the CP going from four byte components to eight byte components would only increase the area of the CP approximately 20%. This economical scaling behavior must be balanced by the fact that for constant speed application the cycle time of the components of the AP would be reduced nearly 40% for the doubling in component number. This reduction is easily accomplished by going to more commercial technologies of 1 µm or so, or by using carry lookahead circuitry instead of ripple carry structures. The balance of the maximum dimension of the vectors that can be used in this AP architecture is determined by the speed of the CMOS fabrication process and the real-time requirements of the problem. The balancing of these two demands is left to the implementor of the architecture.

5.2 Results of Testing

Upon receipt of the initial samples of the AP some testing was done at the Ohio State University using a Tektronics DAS 9200 tester. The preliminary tests were repeated and further investigated using a Credence LT1101 tester at IBM Corporation in Boca Raton, Florida. The results reported here will be the data collected at the later site.

The tests on the Credence LT1101 tester required building an interface between the tester and the chip. This was necessary because the Credence LT1101 did not have the ability to accept a chip of this geometry. The Credence LT1101 is a commercial chip tester, while the 65 pin PGA is not a standard commercial size. This interface was a wirewrap board and may have introduced some unwanted parasitics as discussed later. All testing was done at room temperature.
5.2.1 Functional Testing

Functional testing was performed in a low speed mode at 1 MHz. A computer program was written to randomly generate a series of vectors and the control signals to enter them into the AP. Once the randomly generated “codevectors” were stored a new set of vectors and control signals were applied and the results noted.

The functional tests quickly showed that the fifth bit in the input MATCH bus was not behaving as predicted. The line seemed to be stuck at a value that varied from time to time, but tended to be a 1. The cause was found by looking at the layout where I found that there was a small gap between the lines coming from the I/O pad to the line coming from the internal MATCH bus. The gap was also visible under a microscope.

Upon reflection, the gap in the lines was missed by the testing strategy used when simulating the chip. At each stage of the assembly of the CPs the subcomponents were simulated using IRSIM to verify correct function and get a rough idea of the timing. This worked well up until the final assembly of the AP. With 32 CPs, the priority encoder, control unit, etc. the final transistor count exceeded 50,000 and IRSIM ran very slowly and required a very fast machine with a large amount of RAM to complete even the most simple tests. After the AP core was simulated to satisfaction, the I/O ring was wired up. The simulations were not run after the wiring of the I/O ring, which in retrospect was a mistake.

Another error in the external match circuitry was found during this testing: an unwanted feedback path rendered the extension of the wired-OR bus to multiple
Figure 51: A high level logical view of two of the MATCH bus of two APs connected together.

chips ineffective, irrespective of the problem mentioned above. Figure 51 shows how
the problem can occur when extending the bus across two chips. If both nodes
A and B are high, then node C is high. But if node C ever drops to a low state,
then both nodes A and B drop low and can never return to a high state because of
the feedback through the pulldown transistors attached to C. One solution to this
problem is to break the feedback path associated with the MATCH bus by removing
the feedback transistor and instead feed the result of the NAND-ing of the wired-OR
buses directly back to each of the CPs. This would mean adding an additional bus
to the CPs and increasing the logic of the match circuit by adding an additional leg
to the AND circuit, as shown in Figure 52. For this circuit the logic is the same as in the regular match circuit, with the addition of a line called "Match In" that is the result of the external NAND function. Logically, all that happens is that this bit is still a candidate for the minimum difference if one of two conditions is met: (a) the value stored in the accumulator is zero and the previous bit was valid, or (b) the accumulator bit, the input from the external NAND function, and the MATCH line are high, and the previous bit was valid. This extension of the feedback line throughout the entire chip will naturally slow things down slightly, but the difference should be minimal if drivers are properly sized during layout.

Other than the errors in the MATCH extensions between APs, the chip was found to perform as desired. Functionally, it performed the minimum distance calculation well and seemed to behave correctly. The lack of the MATCH bus working as designed
meant that the chip could not perform the multichip matching function as desired, and that a software assist would be necessary if the AP was ever to be used in a system requiring more than 32 memory elements. For example, in a 128 codeword system, the system software would be responsible for determining which of the codevectors in the 4 APs was really the closest to the input vector. This is computationally possible even with a moderate speed microprocessor, so this particular defect is not insurmountable.

5.2.2 At-Speed Testing

Since the target application for the AP was to be a video subsystem, it was hoped the AP could run at video speeds. For TV running at CCIR 601 resolution there are 720 pixels on a line. In NTSC systems the horizontal scan rate is 15.75 KHz, which means a time of 81.8 ns/pixel. This time includes the blanking and horizontal synchronization times, which is reasonable in a system that assumes that buffering will be required on both ends, as is done in MPEG and other systems. The AP must execute 6 to 9 clock cycles in the period of 4 pixels, which means that the clock frequency for the AP must be 17 to 26 MHz. The spread in operating frequencies is due to the fact that the AP is programmable to allow up to 4 clock cycles for the final winner determination.

In fact, 26 MHz operation is by far the worst case assumption. Many broadcast operations commercial providers are preparing to use non-maximum CCIR resolutions for most programming. Various experiments have shown that resolutions of 352 x 240 at 30 frames per second yield “VCR quality” recordings. The idea of “VCR
quality” is that it is the minimum acceptable level for television broadcast since it is slightly inferior to normal broadcast resolutions, which in turn are inferior to the full CCIR 601 resolution of 720 × 180. In the case of what is thought to be the “normal” mode of operation in digital television at “VCR quality” levels, the AP would have to operate at 8.4 to 12.6 MHz, again depending on how many cycles are required to choose the nearest codeword.

For at-speed testing, the AP was run again on the Credence LT1101 tester. Codewords and input vectors were randomly chosen using the computer program mentioned before. The Credence LT1101 tester has a programmable clock speed which was set from 8 to 26 MHz for the tests.

The results of the testing showed that the AP could operate at frequencies up to approximately 19 MHz, depending on the care with which the system design was done. This result was nearly half that predicted by IRSIM. IRSIM is a digital simulation tool included with the MAGIC! design package. IRSIM is simpler than SPICE and is more conservative than SPICE in its estimates of circuit performance but can be easily used in larger circuits. IRSIM showed that clock speeds in excess of 35 MHz should have been achieved.

The question of why the desired performance wasn’t achieved required further investigation of the AP. Initial testing was done with a bare board and wires connecting the AP to the tester. This setup developed problems of signal and clock bounce at approximately 15 MHz. The symptoms of this clock bounce were that the AP would trigger on both clock edges. When high-speed bypass capacitors were added
the maximum frequencies rose again to approximately 19 MHz before clock bounce again became a problem.

The reasons for the clock bounce could be many. SPICE simulation of the input pad combined with the first stage of the clock repowering tree is relatively sensitive to noise at approximately 2.3 Volts. The reasons for this sensitivity have to do both with the pad design and the high gain of the initial inverter of the clock powering tree. Since the input pad is not Schmitt-triggered, any noise centered about 2.3 V could cause the pad to oscillate. Any sufficiently high pin bounce, as might happen without bypass capacitors in a high impedance CMOS system, would also cause double clocking. Secondly, the high gain of the input stage of the clock tree makes the whole circuit very susceptible to slight oscillations about the transition point of 2.3 Volts, since even small oscillations would be amplified to full, rail-to-rail voltage swings.

An alternate scheme for the double-clocking could be a large amount of clock skew between the data bus latches and the CP registers. This is unlikely, since SPICE simulation indicated that unless capacitance and process values deviated significantly from those measured on test circuits elsewhere on the wafer the clock skew should have been under 1 ns.

The matching time, as measured by the number of clocks to wait until matching occurred could not be measured because of the limitation on the clock frequency. In all cases the matched value was correct in a single clock cycle. This was an indication that to the best of the measurements of the tester, the wired-OR match circuitry
functioned as desired. This is an important fact, since this method of finding the minimum difference is one of the key contributors to the overall speed of the AP.

The coarseness of the time base prevented many tests from being run. For example, it would have been interesting to find how the layout affected the time it took for the minimum difference circuit to function, and to find how long it took for the priority decoder to work. None of these results showed any variations based on location because of the coarseness of the time base.

As a final note, the AP was not as cool as it could have been. The chip warmed noticeably to the touch, even when doing nothing. This is because the pullup resistors were permanently tied high inside the AP. A more efficient design might have made the pullup resistors active only during the comparison phases, although simulation would be required to determine the length of time that the pullup resistors would require to charge the large capacitance of the minimum difference bus lines.

5.3 Summary

The AP functioned nearly as designed. An oversight and incorrect circuitry rendered the extension to multiple chips circuit untestable due to a metal gap in an I/O line. The speed of the AP was limited to relatively slow speed, far slower than that for which it was designed. This was due to the selection of I/O pads for the clock tree and to the very high gain of the initial stage of the clock repowering tree. However, some very important results were still attained:

- The structure of the AP was validated.
• The CPs were found to function correctly and be able to communicate well across the data bus, and to accumulate correctly.

• The wired-OR match circuitry for finding the minimum difference behaved as designed. This was important since it is one of the key mechanisms for increasing the overall speed of vector quantization.

• That an AP could be built in an effective manner, utilizing a designer's time well.

• Showing that cost effective, fast vector quantization can be achieved using even relatively slow CMOS processes.
CHAPTER VI

Recommendations, Conclusions and Future Work

The Associative Processor presented represents a growing trend in VLSI to integrate processor functions with memory functions in order to speed overall system performance. This tighter coupling of memory and arithmetic functions is visible in such diverse systems as RISC CPUs, where large caches are used, to multiprocessor supercomputers. In both cases memory is more tightly coupled to a processor to increase the performance of that processor and the system overall. In cases where the memory and the processor are tightly coupled, and where many of these pairs are used, the communications network between the processors can become the limiting factor in system performance.

6.1 Major Accomplishments of this Work

Much of the work of this research has been devoted to examining systems from all possible levels to determine how to implement a system most economically. This meant that the conventional models for digital image compression were examined and the merits of each were examined. The conventional methods were found to have varying success, with the most promising of the popular methods, MPEG 2, having great potential, but several potentially damaging drawbacks. While MPEG
2 generated the best overall compression. It was not optimally matched to digital broadcast for several reasons. First, MPEG 2 decoders are very complicated pieces of equipment since they require a large amount of buffering and fast DCT processors to be in each decoder. Secondly, they use scalar quantization and Huffman encoding for key portions of the MPEG 2 data stream. If these key portions are distorted, the MPEG 2 data stream could be useless for prolonged periods of time.

The merits of the MPEG 2 system still weigh heavily towards its use, however. The very large compression through both spatial and temporal techniques is very attractive. Vector Quantization offers very good spatial compression and much higher noise immunity than the DCT, scalar quantization and Huffman coder combination in the MPEG 2 specification. VQ also offers the ability to make very inexpensive decoders, since the decoding technique is just a table lookup, which can be performed in real time by very inexpensive and unspecialized hardware. The chief drawback to VQ is its much higher computational complexity in the encoder. A method to overcome this high computational cost makes VQ a very attractive technology, since with fewer broadcasters and more decoders even a small savings in the decoder technology can easily offset a much more expensive encoder.

This dissertation has shown an architecture called the Associative Processor in which a VQ encoder could be designed for fast operation. This was accomplished in several ways. First, the problem was defined as how to do VQ in a real time or near real time system. To make this leap in speed, the problem was attacked at several levels to ease computational requirements. The target system, the human
visual system, was highly nonlinear, so the strict requirement of Euclidean distance was relaxed to the more tractable Minkowski or Manhattan distance. This meant that expensive floating point operations were not required, so a parallel architecture with many processors on a single chip was possible.

With a parallel architecture it is imperative to choose an architecture for the sub-processors that can be efficiently implemented since this structure will be replicated many times. The architecture given here uses a subtractor and accumulator to efficiently calculate the Manhattan distance between a stored vector and an input vector. This architecture grows slowly in complexity as the length of both the vector and the vector components increase.

Even in a parallel architecture, the determination of the minimum distance can take some time if done in a purely digital fashion. Such a solution can take a number of comparisons nearly equal to the size of the codebook stored in the AP. To overcome this problem, the architecture presented here shows a unique method for using a wired-OR bus as an extensible, fast minimum determination technique.

As final proof of this concept, a prototype was implemented. All analog and all digital methods were evaluated, but the final system came out strongly weighted towards a digital implementation. This was because while the analog approach was roughly the same size as the digital solution for the given prototype, the analog approach would not scale as well with increasing component resolution. The analog issues were not completely eliminated, however, since the use of a wired-OR circuit for the minimum determination depended strongly on analog computation to determine
settling times.

The prototype chip was implemented in 2μm silicon using an Orbit Semiconductor two level metal CMOS process through the MOSIS service. The chip held 32 subprocessors on a single die and was designed to operate at 33 MHz for use in a digital video system. The chip was tested and although there were a few errors in the layout of the chip, the basic architecture was validated.

The use of multilevel analysis throughout this problem shows how optimization of a large system can be accomplished. The idea of digital video broadcasting was examined and the weak points identified. An alternate technology was proposed which would be more economical if certain barriers could be overcome. An architecture to overcome these problems was proposed that used techniques from both digital and analog circuit analysis. A working prototype showed the soundness of the solution. The overall effect shows how cooperation throughout the spectrum of technology can accomplish a goal that may seem impossible.

6.2 Discussion

The AP architecture was chosen to be efficient as the dimension of the system vectors changed. As such, it may not always be possible to operate the AP in a real time mode for a given problem, but the architecture will always yield a very high speedup to a given VQ problem. It is important to note that while much of the discussion of the design of the AP was driven by digital video and its requirements, those requirements are often similar to those of other systems. VQ has applications to many different problems in control, image processing, systems theory, and many other fields and
most of them also could use the AP architecture as a hardware speed boost to their systems.

The relatively simple AP architecture and CP design can be modeled and designed in very high level languages such as VHDL. As an example, see the VHDL in Appendix E. This VHDL is synthesizable with the Synopsys VHDL compiler and yields a very efficient, very fast CP in 0.8 μm technology. The small size of the CP shows that a single chip solution for a 256 codeword system is possible. Such a chip using the AP architecture would be space limited, not pin limited. This means that fewer than 60 pins for packaging could be used. The small pin count package is very desirable since using more than approximately 168 pins would raise system cost to an implementor since most board vendors have difficulty and charge more for systems with pad spacings less than 0.2 mils, and the cost of board manufacture for smaller pin count packages drops sharply with the decrease in pin count. The cost for a 256 codeword, single chip solution is approximately $9 in a die image of approximately 6.8mm × 6.8mm in a vendor 0.8 μm process in quantities of more than 50,000 per year. The simulated speed for this AP is much higher than that of the 2μm design; a rough estimate shows that a 50 MHz part should be easily attainable.

While a VHDL description of the AP is possible, it is generally not optimal. Using latches for memory, as was done in Appendix E is very inefficient. The SRAM-like structures used in the prototype are much more area efficient and as fast. It is also difficult to describe the wired-OR bus in VHDL. VHDL has no capacity to bidirectionally signal over a single I/O bus, so the input and outputs must be split
and a tristate buffer used rather than the simpler pulldown transistor. Not being able to use VHDL well is not as much of a drawback as it might seem. The small size and simplicity of the CPs lend themselves to easy and efficient custom designs.

The results of the AP design considerations and prototype testing have yielded several lessons useful for future implementors, both at the commercial and research level. In no particular order, some of the lessons that have been brought to the forefront through this prototype are:

- Schmidt triggered receivers on clock inputs would greatly help the speed characteristics of the AP. Clock tree design with more attention to packaging issues should be attempted. For example, adding a single repowering element at the accumulator register would provide superior clock performance and balancing.

- It is important when designing at the polygon level to run sufficient simulations to fully exercise the connection of one level to another. While the internal core of the AP was thoroughly simulated, the full chip was not. This oversight allowed an important part of the design to be untestable.

- The tradeoff of design complexity between analog and digital domains must be examined before an architecture is selected. In the case of the AP, the fact that the largest part of the CPs grows logarithmically with increasing vector dimension when done digitally is very important. The point at which the digital implementation is more efficient will vary with technology, however, and should be evaluated at each step.
The optimization of the CP area is a key element of efficient implementation of the AP architecture. The prototype had only 32 CPs on it, but efficient implementation of the register and combinatorial elements could easily have reduced the size of the CP, and thus increased the number of CPs on a given size die. Had more time been spent optimizing the layout of the CP, it is possible that 50% more CPs could have been included on the same size die.

All of these issues can be resolved with proper planning, although not all results will obvious beforehand.

As mentioned before, an associative processor of the type shown here is not only suitable for vector quantization. It is useful in many other applications in which the determination of the nearest vector is very important. For example, symbol detection for QPSK and QAM systems is a very difficult problem requiring sometimes intricate electronics. A chip such as the AP could serve as a maximum likelihood detector for such a system, rather than requiring techniques like forward error correction. With 64 QAM and 256 QAM being proposed for cable broadcast systems, the AP could provide a very good detection subsystem. The application of the AP is even more advantageous in systems such as code-division multiple access (CDMA) where nonlinear codes such as the Nordstrom–Robinson codes provide superior noise characteristics and signal density but are more difficult to decode.

Another area of future research would be a real-time VQ codebook generator. For example, the AP already generates the minimum difference and the address of the nearest vector. The encoding system could use these values as a way of adapting a
Figure 53: A simple encoding system that uses the minimum difference and address generated by the AP to encode a new, incremental codebook in a real-time fashion.

A new codebook in parallel to the encoding of the input signal. A simple, relatively easy to implement idea is shown in Figure 53. In this system, the codeword is selected from the AP outputs. The vector that generated that AP output is applied to the codeword and the codeword is adapted via the appropriate algorithm. Such a system could be implemented with a CAM for data storage and a DSP for the adaptation algorithm. If the adaptation algorithm is too intensive multiple cycles may be taken to adapt the codebook. Specialized hardware could also be employed for the adaptation. Just as special attention was required to match the AP to realizable hardware, special attention would be required for an efficient codebook generator.

An alternate architecture could use a version of the AP in a cached-codebook type
solution, where the most probable codewords would be searched first, then the second, etc. In a situation like this all that would be required is some additional memory to store the additional codewords and some muxes to direct this new memory in the combinatorial unit. This has the advantage of getting larger codebooks at a relatively low incremental cost of adding memory cells. For the prototype system adding an additional codevector would only have been a 16% area penalty, for example. This modified AP architecture is an example of trading off multiple temporal passes against larger die size implications.

The AP is also useful for a whole set of neural network problems. Since VQ is essentially a process of learning to partition the quantization space into a discrete set of representative vectors, the AP will serve to quickly find the correct partition. The problem may be something like modeling currency trading, where the axes are the value of the dollar against the yen, pound, franc, and peso. In such a system, an AP could quickly determine the appropriate state, which would be updated by some neural network algorithm to produce some output. The AP architecture is not specific to video signals and can be generalized to many of the self-organization problems to which VQ is applied.

As far as VQ for video, several areas need to be explored farther to fully optimize the system. Some of the areas of exploration might be:

- Explore the use of image smoothing techniques to improve the efficiency of the encoding. For example, the Cinepak algorithm from SuperMac technologies uses a Gaussian blur filter to smooth the result of the quantization process and
present a better image. Such a filter could be applied using a multi-tap filter to the input image to smooth the image before compression, possibly resulting in a smoother image and better quantization results.

• Investigate the behavior of predictor-based systems and their interaction with VQ systems. A good predictor will reduce the variance of the input vectors, which will in turn result in better VQ performance.

• Explore the trade-offs involved in using larger VQ blocks. As the vector dimension increases the number of codewords required will grow exponentially. This will limit the effective application dimension for the AP, and alternate architectures and methods will have to be used. Some of the sub-optimal techniques described earlier will become better candidates for implementation, possibly by extending the memory associated with each CP. For example, an iterative search could be done using block 1 of memory, then block 2, etc., as shown in Figure 54.

• More precisely scale the point at which the digital vs. analog tradeoff occurs for a general technology. For 2\(\mu\)m technology the analog solution is smaller until approximately 9 bits, but this should be tend to decrease for smaller geometry fabrications.

• An analysis of more algorithms which might benefit from VQ technology is in order. There are many uses of scalar quantization in use today which could be modified to use the more efficient VQ techniques, especially when real time
Figure 54: An iterative solution using a modified AP architecture in which multiple codevectors are associated with each CP to find the minimum difference in a system. Each codevector will be made of multiple components, so the selector is more complicated in this case.
vector quantization may be possible using a hardware accelerator.

- Alternate AP architectures using pipelining should be explored. The AP as designed only uses parallelization techniques to improve speed. The idea of pipelining some of the operations, especially during the minimum difference determination phase, holds significant promise of reducing the cycle time of the design. For example, by double buffering the accumulator in the AP the generation of the distance could be going on at the same time as the minimum distance calculation.

All of these areas could result in more applications for architectures of the type presented earlier, and some of the more theoretical ones deserve more attention since efficient VQ techniques can be implemented at reasonable cost.

6.3 Conclusion

Many engineering problems can at first glance be daunting in scope and scale, but through careful design they may often be overcome. The work in this dissertation has shown that a proper balance between costs can drive a system to a solution that may appear difficult, but is rewarding. The MPEG 2 algorithm, for example, can be improved by using vector quantization since the decoders for a MPEG 2/VQ hybrid would be far simpler. This cost savings in the large number of decoders is possible only at the cost of relatively higher computations at the broadcast center. The work here showed an architecture called the Associative Processor which utilized algorithms like the Manhattan distance measure which allowed an optimized, parallel solution to
be presented. This AP also utilized a very fast minimum difference detector circuit. The overall optimizations allowed a prototype solution to reach video speeds even in a rather old technology.

The usefulness of the AP architecture is by no means limited to digital video. The architecture scales efficiently to systems where the vectors may have more components and higher resolution. This means that the AP can be used as a general purpose hardware accelerator for many problems such as neural network design, image processing, and the like.

This thesis has attempted to illustrate the efficiencies that can be achieved when system architect and system implementor can effectively communicate their abilities, liabilities and desires. A difficult problem was examined, and the driving factors at the levels of economics, systems design, chip design, and circuit design were considered and a proposed architecture was designed, simulated, and implemented. This interaction was crucial in optimizing the overall benefits of the system and the interaction between the algorithm and implementation is a critical theme that will continue to become more important in the future.
Appendix A

Proof the Generalized Distance Measure is a Metric

Consider the space $R^n_p$ of all ordered $n$-tuples $x = (x_1, \ldots, x_n)$ of real numbers with the generalized distance measure

$$d_p(x, y) = \left( \sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \quad (A.1)$$

where $p$ is a fixed number and $p \geq 1$. To show that this qualifies as a valid metric the following three properties must hold:

1. $d(x, y) = 0$ if and only if $x = y$.
2. $d(x, y) = d(y, x)$, i.e. the function is symmetric.
3. The triangle inequality holds: $d(x, z) \leq d(x, y) + d(y, z)$.

The first two properties are obvious upon inspection. The third property is more difficult and will be covered in the remainder of this exercise.

Let

$$x = (x_1, \ldots, x_n), y = (y_1, \ldots, y_n), z = (z_1, \ldots, z_n) \quad (A.2)$$

be three points in $R^n_p$, and let

$$a_i = x_i - y_i, b_i = y_i - z_i \quad \forall i \in [1, \ldots, n]. \quad (A.3)$$
Then the triangle inequality becomes the form known as Minkowski’s inequality:

\[
\left( \sum_{i=1}^{n} |a_i + b_i|^p \right)^{1/p} \leq \left( \sum_{i=1}^{n} |a_i|^p \right)^{1/p} + \left( \sum_{i=1}^{n} |b_i|^p \right)^{1/p}.
\] (A.4)

The inequality is obvious for \( p = 1 \); the more difficult proof is for \( p > 1 \), which is the remainder of this section.

To show that A.1 is true, we must first show that Hölder’s inequality

\[
\sum_{i=1}^{n} |a_i b_i| \leq \left( \sum_{i=1}^{n} |a_i|^p \right)^{1/p} \left( \sum_{i=1}^{n} |b_i|^q \right)^{1/q}
\] (A.5)

where

\[
\frac{1}{p} + \frac{1}{q} = 1
\] (A.6)

and \( p > 1 \) and \( q > 1 \) is true. The equation A.5 is homogeneous in that if it is true for points \( a \) and \( b \), it is true for \( \lambda_1 a \) and \( \lambda_2 b \), where \( \lambda_1 \) and \( \lambda_2 \) are arbitrary real numbers.

This simplifies A.5 down to proving only the more manageable case where

\[
\sum_{i=1}^{n} |a_i|^p = \sum_{i=1}^{n} |b_i|^p = 1.
\] (A.7)

If we use the condition in A.7 then A.5 becomes

\[
\sum_{i=1}^{n} |a_i b_i| \leq 1.
\] (A.8)

To show this condition, consider the areas \( S_1 \) and \( S_2 \) shown in Figure 55. The curve

\[
\eta = \xi^{p-1}, \quad \text{or} \quad \xi = \eta^{q-1}
\] (A.9)

serves to delimit the two regions. From simple calculus,

\[
S_1 = \int_{a}^{b} \xi^{p-1} d\xi = \frac{a^p}{p}, \quad S_1 = \int_{0}^{a} \eta^{q-1} d\eta = \frac{b^q}{q}.
\] (A.10)
Figure 55: A simple parameterization of a partitioning curve.
From the figure it is obvious that

$$S_1 + S_2 \geq ab$$  \hfill (A.11)

for any $a, b \geq 0$, so we can write

$$ab \leq \frac{a^p}{p} + \frac{b^q}{q}$$  \hfill (A.12)

Setting $a = |a_i|$, $b = |b_i|$, summing $i$ from 1 to $n$ we get Equation A.8, which proves Equation A.5.

To prove Minkowski's inequality A.1, which started this whole process, begin by considering the identity

$$(|a| + |b|)^p = (|a| + |b|)^{p-1}|a| + (|a| + |b|)^{p-1}|b|$$  \hfill (A.13)

Setting $a = a_i$, $b = b_i$ and summing $i$ from 1 to $n$, we get

$$\sum_{i=1}^{n}(|a_i| + |b_i|)^p = \sum_{i=1}^{n}(|a_i| + |b_i|)^{p-1}|a_i| + \sum_{i=1}^{n}(|a_i| + |b_i|)^{p-1}|b_i|.$$  \hfill (A.14)

We apply A.5 with the fact that $q(p - 1) = p$ to get

$$\sum_{i=1}^{n}(|a_i| + |b_i|)^p \leq \left(\sum_{i=1}^{n}(|a_i| + |b_i|)^p\right)^{1/p} \left(\left[\sum_{i=1}^{n} |a_i|^p\right]^{1/p} + \left[\sum_{i=1}^{n} |b_i|^p\right]^{1/p}\right)$$  \hfill (A.15)

which in turn yields

$$\left(\sum_{i=1}^{n}(|a_i| + |b_i|)^p\right)^{1/p} \leq \left(\sum_{i=1}^{n} |a_i|^p\right)^{1/p} + \left(\sum_{i=1}^{n} |b_i|^p\right)^{1/p},$$  \hfill (A.16)

which immediately yields Minkowski's inequality, thereby proving the triangle inequality and the fact that Equation A.1 is in fact a metric.
Appendix B

Subtraction using \((n - 1)\)'s complement arithmetic

The definition of an \((n - 1)\)'s complement of a number \(A\) is

\[
(A)_{n-1,c} \equiv n^i - n^{-j} - A
\]  

(B.1)

where

\[
i = \text{number of digits in the integer portion of } A \quad \text{(B.2)}
\]

\[
j = \text{number of digits in the fractional portion of } A \quad \text{(B.3)}
\]

\[
n = \text{radix number} \quad \text{(B.4)}
\]

For a 2's complement number, finding the 1's complement is very easy: complement each digit (0 or 1) of the number up to the most significant digit specified.

To perform subtraction using \((n - 1)\)'s complements use the following procedure.

Assume the numbers to be subtracted are \(A\) and \(B\) and the result is to be \((A - B)\).

1. Add \(A\) to the \((n - 1)\)'s complement of \(B\).

2. Check the state of the overflow carry on the most significant bit.

   (a) If an overflow carry exists, the result is positive and the overflow carry is to be added to the least significant digit of the result.
(b) If the overflow carry does not exist, the result is a negative number. The result of the addition must be complemented and a minus sign added to the front of the result.

For use in the cellular processor, we have 2's complement numbers used throughout, and we want to find the absolute value of the difference of the two numbers. Since \(|A - B| = |B - A|\), it is easiest to store the complemented number in the memory. Further, because we do not care about the sign bit, we only have to do an XNOR of the overflow carry with the result of the addition to generate the correct value for the subtraction, with one important caveat: the overflow carry must be connected to the carry in of the adder that takes the result of the subtraction and adds it to the value in the accumulator register.
Appendix C

Input Decks For Automatically Synthesized Components

The following are decks that are used in the synthesis of various components of the Associative Processor.

C.1 State Assignments for the AP Control Machine

The following is the input deck for determining the optimal state assignments of the PLA that drives the state machine for the Associative Processor. It is designed to be an algorithm that uses an exhaustive search to find the minimal implementation of the PLA. This is important since the PLA is implemented in pseudo-NMOS and will draw more power than typically required. This particular deck is to be fed to the nova state assignment program with the options -e ie -r to force the exhaustive search.

The output of this program will be a listing with the optimal state assignments.

C.1.1 nova Program to Generate Optimal State Assignments

# Find optimal state assignments for the Associative memory
# controller. Use nova -e ie -r
#
# Author: Rich Kaul Date: 2.26.92
#
# Inputs Present State Next State Outputs

172
# Match Store Reset cycle sel\{1,0\} accum load reg clear

<table>
<thead>
<tr>
<th>State</th>
<th>Store</th>
<th>Reset</th>
<th>Cycle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>idle</td>
<td>idle</td>
<td>00</td>
<td>00-00</td>
</tr>
<tr>
<td>001</td>
<td>idle</td>
<td>idle</td>
<td>00</td>
<td>00-00</td>
</tr>
<tr>
<td>111</td>
<td>idle</td>
<td>mpath0</td>
<td>00</td>
<td>00-00</td>
</tr>
<tr>
<td>111</td>
<td>lpath0</td>
<td>lpath1</td>
<td>01</td>
<td>01-1-</td>
</tr>
<tr>
<td>111</td>
<td>lpath1</td>
<td>lpath2</td>
<td>11</td>
<td>11-1-</td>
</tr>
<tr>
<td>111</td>
<td>lpath2</td>
<td>lpath3</td>
<td>10</td>
<td>10-1-</td>
</tr>
<tr>
<td>111</td>
<td>mpath0</td>
<td>mpath1</td>
<td>00</td>
<td>00-00</td>
</tr>
<tr>
<td>111</td>
<td>mpath1</td>
<td>mpath2</td>
<td>01</td>
<td>01-1-</td>
</tr>
<tr>
<td>111</td>
<td>mpath2</td>
<td>mpath3</td>
<td>11</td>
<td>11-1-</td>
</tr>
<tr>
<td>111</td>
<td>mpath3</td>
<td>mpath4</td>
<td>10</td>
<td>10-1-</td>
</tr>
<tr>
<td>111</td>
<td>mpath4</td>
<td>idle</td>
<td>--</td>
<td>--001</td>
</tr>
<tr>
<td>110</td>
<td>mpath4</td>
<td>mpath5</td>
<td>--</td>
<td>--001</td>
</tr>
<tr>
<td>110</td>
<td>mpath5</td>
<td>mpath4</td>
<td>--</td>
<td>--001</td>
</tr>
<tr>
<td>110</td>
<td>mpath5</td>
<td>idle</td>
<td>--</td>
<td>--001</td>
</tr>
<tr>
<td>110</td>
<td>mpath6</td>
<td>mpath5</td>
<td>--</td>
<td>--001</td>
</tr>
<tr>
<td>110</td>
<td>mpath5</td>
<td>mpath6</td>
<td>--</td>
<td>--001</td>
</tr>
<tr>
<td>110</td>
<td>mpath6</td>
<td>idle</td>
<td>--</td>
<td>--001</td>
</tr>
<tr>
<td>110</td>
<td>mpath7</td>
<td>mpath6</td>
<td>--</td>
<td>--001</td>
</tr>
<tr>
<td>110</td>
<td>mpath6</td>
<td>mpath7</td>
<td>--</td>
<td>--001</td>
</tr>
<tr>
<td>110</td>
<td>mpath7</td>
<td>mpath6</td>
<td>--</td>
<td>--001</td>
</tr>
<tr>
<td>111</td>
<td>mpath7</td>
<td>idle</td>
<td>--</td>
<td>--001</td>
</tr>
</tbody>
</table>

C.2 The BDSYN code to generate the PLA

The following code is fed to bdsyn to generate the BLIF (Berkeley Logic Interchange Format) statements that determine the PLA to be simulated. The state diagram for this particular circuit is given in Figure 22. The reset signal is not mentioned in this deck because the reset path is taken care of by the use of resettable registers. These
registers provide the inputs which determine the next state of the machine. Since the control machine is a Moore machine, the output of the PLA will always be correct as long as the reset signal is applied and the PLA will not change state, nor will any invalid signals be output.

It is important that this deck be processed with the -z option. This option forces all variables not set in a state to be set to zero. This is extremely important for correct operation of the state machine.

After the BLIF is generated with bdsyn, it is fed to the octpla program with the -nofold option. That option prevents PLA folding (both column and row) from being done on the PLA. This had several benefits, including a smaller size and correct operation since without the -nofold option incorrect PLAs were frequently generated.

Finally, the BLIF can also be used with the anchor program to generate a PLA. anchor is the program that the OCTTOOLS authors suggest rather than octpla. However, at the time of this writing the anchor program produces larger and incorrect PLAs for this particular example and is not recommended.

C.2.1 The BDLS to generate the State Controller

! This is my state controller for the AM, done as a PLA. There are 2 ! distinct paths for the controller, depending on if the sequence to ! be executed is a load instruction or a match instruction.
!
! Note: reset is not required, since it is wired into the state
MODEL am_controller
  nextState<3:0>, muxout<1:0>, load, accum, regclear =
  store, match, waits<1:0>, presentState<3:0>;

BEHAVIOR;

! state assignments
constant IDLE = 0;
constant LPATH0 = 7, LPATH1 = 6, LPATH2 = 5, LPATH3 = 4;
constant MPATH0 = 11, MPATH1 = 10, MPATH2 = 9, MPATH3 = 8,
  MPATH4 = 15, MPATH5 = 14, MPATH6 = 13, MPATH7 = 12;

! Constants
constant HI = 1, LO = 0;
constant FIRST = 0, SECOND = 1, THIRD = 3, FOURTH = 2;

PORT
  store input,       ! load scratch reg into memory
  match input,      ! match scratch reg against memory
  waits<1:0> input, ! how many cycles to allow convergence
  presentState<3:0> input,

  muxout<1:0> output, ! mux controls (depend on state only)
  load<0> output,    ! load values into memory
  accum<0> output,    ! accumulate differences
  regclear<0> output, ! reset accum registers
  nextState<3:0> output;

ROUTINE associative_memory_controller;

  nextState = IDLE;

  SELECT presentState FROM
    [IDLE]:
      BEGIN
        muxout = FIRST;
load = LO;
regclear = LO;
IF ( store ) THEN
    nextState = LPATH0
ELSE IF ( match ) THEN
    nextState = MPATH0
ELSE
    nextState = IDLE;
END;

[LPATH0] :
BEGIN
    muxout = FIRST;
    load = HI;
    nextState = LPATH1;
END;

[LPATH1] :
BEGIN
    muxout = SECOND;
    load = HI;
    nextState = LPATH2;
END;

[LPATH2] :
BEGIN
    muxout = THIRD;
    load = HI;
    nextState = LPATH3;
END;

[LPATH3] :
BEGIN
    muxout = FOURTH;
    load = HI;
    nextState = IDLE;
END;

[MPATH0] :
BEGIN
muxout = FIRST;
load = LO;
accum = HI;
regclear = HI;
nextState = MPATH1;
END;

[MPATH1]:
BEGIN
muxout = SECOND;
load = LO;
accum = HI;
regclear = HI;
nextState = MPATH2;
END;

[MPATH2]:
BEGIN
muxout = THIRD;
load = LO;
accum = HI;
regclear = HI;
nextState = MPATH3;
END;

[MPATH3]:
BEGIN
muxout = FOURTH;
load = LO;
accum = HI;
regclear = HI;
nextState = MPATH4;
END;

[MPATH4]:
BEGIN
load = LO;
accum = LO;
regclear = HI;
IF ( waits EQL 0 ) THEN
nextState = IDLE
ELSE
  nextState = MPATH5;
END;

[MPATH5]:
BEGIN
  load = LO;
  accum = LO;
  regclear = HI;
  IF (waits EQL 1) THEN
    nextState = IDLE
  ELSE
    nextState = MPATH6;
  END;

[MPATH6]:
BEGIN
  load = LO;
  accum = LO;
  regclear = HI;
  IF (waits EQL 2) THEN
    nextState = IDLE
  ELSE
    nextState = MPATH7;
  END;

[MPATH7]:
BEGIN
  load = LO;
  accum = LO;
  regclear = HI;
  nextState = IDLE;
END;

ENDBEHAVIOR;
ENDMODEL am_controller;
C.3 Synthesis decks for use with the 32 line priority decoder.

The first of these decks, shown in Section C.3.1, forms the combinational logic for the priority decoder. This is only combinational logic since bdsyn cannot generate memoried circuits. This deck implements the logical equations of Equation 4.14. Again, the BLIF was generated by using bdsyn -z -A to zero the unmentioned signals in the states generated.

By trial and error it was determined that a standard cell layout of the resulting machine was much faster and more compact than the equivalent PLA. To reduce the size of the standard cell implementation, the deck shown in Section C.3.2 was used. It selected the Mississippi State University Standard Cell package, version 2.2 from the Lager1V toolset, ran a boolean minimization and did some cursory checking of the resulting output.

C.3.1 Priority BDLS source

! This is the logic for the priority decoder. Use this with the ! flipflops to build a clocked output latching 32 entry priority ! encoder.
!
! Author: Rich Kaul Date: 2.21.92

MODEL priologic
  final<5:0> =
    zero<3:0>, one<3:0>, two<3:0>, three<3:0>;
ROUTINE priodecode;
    IF ( zero<0> AND three<0> ) THEN
        final<0> = 1
    ELSE IF ( zero<1> AND three<1> AND NOT three<0> ) THEN
        final<0> = 1
    ELSE IF ( zero<2> AND three<2> AND NOT three<1> AND
                NOT three<0> ) THEN
        final<0> = 1
    ELSE IF ( zero<3> AND three<3> AND NOT three<2> AND
                NOT three<1> AND NOT three<0> ) THEN
        final<0> = 1;

    IF ( one<0> AND three<0> ) THEN
        final<1> = 1
    ELSE IF ( one<1> AND three<1> AND NOT three<0> ) THEN
        final<1> = 1
    ELSE IF ( one<2> AND three<2> AND NOT three<1> AND
                NOT three<0> ) THEN
        final<1> = 1
    ELSE IF ( one<3> AND three<3> AND NOT three<2> AND
                NOT three<1> AND NOT three<0> ) THEN
        final<1> = 1;

    IF ( two<0> AND three<0> ) THEN
        final<2> = 1
    ELSE IF ( two<1> AND three<1> AND NOT three<0> ) THEN
        final<2> = 1
    ELSE IF ( two<2> AND three<2> AND NOT three<1> AND
                NOT three<0> ) THEN
        final<2> = 1
    ELSE IF ( two<3> AND three<3> AND NOT three<2> AND
                NOT three<1> AND NOT three<0> ) THEN
        final<2> = 1;

    IF ( (three<1> AND NOT three<0>) OR
         (NOT three<2> AND NOT three<0>) ) THEN
        final<3> = 1;

    IF ( NOT three<0> AND NOT three<1> ) THEN
final<4> = 1;

IF ( three<0> OR three<1> OR three<2> OR three<3> ) THEN
final<5> = 1;

ENDROUTINE priodecode;

ENDBEHAVIOR;
ENDMODEL priologic;

C.3.2 Optimization script for misII

This optimization script performs algebraic minimization on an input file. It uses the
MSU standard cells and the misII program to generate optimized logic.

# Boolean minimizer script for use with the MSU standard cells
# This deck will take an input BLIF and perform minimization and
# cursory checking after the minimization. It is very important
# to check the output more thoroughly with a better simulator
# after synthesis.
#
# Author: Rich Kaul Date: 2.12.92
#
set OCT-CELL-VIEW physical
set OCT-CELL-PATH $OCTTOOLS/tech/scmos/msu/stdcell2_2
set OCT-TECHNOLOGY scmos
set OCT-EDITSTYLE SYMBOLIC
set OCT-VIEWTYPE SYMBOLIC
#
# Read the stdcell descriptions
#
read_library stdcell2_2.genlib
#
# Here are the boolean optimizer steps
#
el 10
sim1 *
  asb
glx
  asb
e1 0
  asb
glx
  asb
e1 0
sim2 *
gd *
  asb
e1 -1
  asb
sim2 *
e1 -1
#
  # Map the logic to standard cells
  #
  map
  phase -g
  print_stats
  print_gate -s
  #
  # Verify the result
  #
  verify logic.blif
Appendix D

Relative Sizing of Wired–OR Transistors

A wired–OR circuit like that in Figure 56 has a single pullup transistor and possibly many pulldown transistors. The structure is roughly equivalent to the use of a depletion mode NMOS transistor inverter, and like the depletion mode NMOS inverter it has the disadvantage that it always dissipates power. The advantage is that the circuit provides a very easy and quick method of communicating between many computational elements.

Figure 56: A possible wired–OR circuit. Note the single pullup and many pulldowns.
To consider the effects of sizing, the output trip level of the inverter has to be considered. The worst case for the noise margin and other calculations comes from the case where only one of the pulldown transistors is on and all others off. From this point on, this is the case that will be considered. For a set of inverters to be useful, we need to be able to cascade them, and so we need to have

\[ V_o = V_{in} = V_{trip} \]  \hspace{1cm} (D.1)

where \( V_{trip} \) is the gate threshold voltage (the voltage at which the inverter is metastable). To optimize the noise margin, we should set the trip voltage midway between the power rails, so we want

\[ V_{trip} = \frac{V_{DD}}{2}. \]  \hspace{1cm} (D.2)

At \( V_{trip} \) the NMOS device is in saturation and the PMOS device is in the linear region. Assuming that the substrates of the NMOS and PMOS transistors are tied to their sources, we can write that the current in the NMOS transistor is

\[ I_{dsn} = \frac{\beta_n}{2} (V_{trip} - V_{in})^2 \]  \hspace{1cm} (D.3)

and for the PMOS transistor we have

\[ I_{dsp} = \beta_p \left[ (-V_{DD} - V_{tp})V_{ds} - \frac{V_{ds}^2}{2} \right]. \]  \hspace{1cm} (D.4)

If we set \( I_{dsn} = I_{dsp} \) and solve for \( V_{trip} \) we get that

\[ V_{trip} = V_{in} + \sqrt{\frac{2\beta_p}{\beta_n} \left[ (-V_{DD} - V_{tp})V_{ds} - \frac{V_{ds}^2}{2} \right]} \]  \hspace{1cm} (D.5)
If we plug in the fairly typical values of $V_{trp} = V_{DD}/2$, $V_{in} = |V_{ip}| = V_{DD}/5$, and $V_{DD} = 5$ volts, then we find that

$$\frac{\beta_p}{\beta_n} = \frac{1}{6} \quad (D.6)$$

In a typical technology where $\beta_p = \beta_n/2$, the ratio of the $(W/L)_n$ to $(W/L)_p$ of the transistors should be 3/1. To improve the noise margin a 4/1 ratio may be used, which is similar to the measure suggested for NMOS ratio logic [37]. Ratios lower than 3/1 are not suggested for circuits that are to be cascaded.
Appendix E

VHDL Descriptions of AP Components

The following are two VHDL descriptions that form the meat of the Associative Processor: the control state machine and Cellular Processor. These files are meant to be compiled with the Synopsys VHDL compiler and synthesizer. A complete synthesis of the AP cannot be done in VHDL using the Synopsys toolset, since the synthesizer cannot handle the wired-or logic that is used in the bus resolution for the entire chip. As such, the pullup resistor will have to be instantiated in any over-all design for a complete replacement AP.

The goal of this section is to provide a framework which could easily be expanded to a full 256-codeword AP using any one of a number of commercial, sub-micron standard cell or sea-of-gates processes.
E.1 The Control State Machine

The VHDL presented here performs similar functions to the PLA/register combination used in the prototype chip. It is somewhat different in that the outputs to this machine come directly off state latches to minimize the clock skew. This is more inefficient than the setup used in the prototype chip, but since this design is targeted more towards a commercial sea-of-gates design using a 0.8μm process, the inefficiency is insignificant.

-- State - Associative Processor state machine

--
-- This is the state machine for the Associative Processor.
-- Two paths, a load path and a match path. Synchronous
-- reset.
--

-- Library load time.
library IEEE;
use IEEE.std_logic_1164.all;

--
-- Declare who we are
--
entity AP_State is
  port (clk, match, store, reset : in std_logic;
        mtchnum : in std_logic_vector(1 downto 0);
        load, regclr, accum : out std_logic;
        mux : out std_logic_vector(1 downto 0));
end;

--
-- This is the synthesizable version of the state machine.
--
architecture synth of AP_State is
  type State_Type is (IDLE, LPATH0, LPATH1, LPATH2, LPATH3,
    MPATH0, MPATH1, MPATH2, MPATH3, MPATH4,
    MPATH5, MPATH6, MPATH7);
signal State, Next_State : State_Type;

begin

  -- There are two processes in this thread. The first generates
  -- the registers, the second does the state assignment and output
  -- logic generation. It is done this way to make this chunk of
  -- logic safer for synthesis.

  -- This is the section of logic dealing with the generation of
  -- the state flipflops.
  process begin
    wait until elk'event and elk = '1';
    State <= Next_State;
  end process;

  -- This section is the combinatorial logic for the next state and
  -- output assignment.

  process (match, store, reset, clk) begin

    -- Default state assignment
    Next_State <= State;

    -- Synchronous reset
    if (reset = '0') then
      Next_State <= IDLE;
    else

      -- This is the core logic of the state machine. There are two
      -- main branches, the load path which loads 4 bytes of memory
      -- in each cellular processor sequentially. The match path
      -- generates the control signals such that the stored elements
      -- in each cellular processor are subtracted from the value
      -- on the data bus, then added to the value stored in the
      -- accumulator. The match path can be extended based on
-- the value on the mtchnum input. The paths of the
-- state machine are otherwise not affected by inputs after
-- they have been started (except by reset, of course).

case State is

-- The idle state of the state machine. This is the
-- only state sensitive to the store and match signals.

when IDLE =>
  if (store = '1') then
    Next_State <= LPATH0;
  elsif (match = '1') then
    Next_State <= MPATH0;
  else
    Next_State <= IDLE;
  end if;
load <= '0';  -- Load memory elements off
regclr <= '0';  -- Accumulator register reset on
accum <= '0';  -- Accumulator register clock off
mux <= "--";  -- Memory element selected don't cared

-- This is the path that generates the control signals
-- to load the value on the data bus into the selected
-- memory location in the selected cellular processor.

when LPATH0 =>
  Next_State <= LPATH1;
load <= '1';  -- Load memory elements on
regclr <= '-';  -- Accumulator register don't cared
accum <= '-';  -- Accumulator register don't cared
mux <= "00";  -- First memory element selected

when LPATH1 =>
  Next_State <= LPATH2;
load <= '1';  -- Load memory elements on
regclr <= '-';  -- Accumulator register don't cared
accum <= '-';  -- Accumulator register don't cared
mux <= "01";  -- Second memory element selected
when LPATH2 =>
Next_State <= LPATH3;
load <= '1';       -- Load memory elements on
regclr <= '-';     -- Accumulator register don't cared
accum <= '-';     -- Accumulator register don't cared
mux <= "11";      -- Third memory element selected

when LPATH3 =>
Next_State <= IDLE;
load <= '1';       -- Load memory elements on
regclr <= '-';     -- Accumulator register don't cared
accum <= '-';     -- Accumulator register don't cared
mux <= "10";      -- Fourth memory element selected

-- This is the path that does the global difference
taking, accumulating and comparison.

when MPATH0 =>
Next_State <= MPATH1;
load <= '0';       -- Load memory elements off
regclr <= '1';     -- Accumulator register clear off off
accum <= '1';     -- Accumulator changes on
mux <= "00";      -- First memory element selected

when MPATH1 =>
Next_State <= MPATH2;
load <= '0';       -- Load memory elements off
regclr <= '1';     -- Accumulator register clear off
accum <= '1';     -- Accumulator changes on
mux <= "01";      -- Second memory element selected

when MPATH2 =>
Next_State <= MPATH3;
load <= '0';       -- Load memory elements off
regclr <= '1';     -- Accumulator register clear off
accum <= '1';     -- Accumulator changes on
mux <= "11";      -- Third memory element selected

when MPATH3 =>
Next_State <= MPATH4;
load <= '0'; -- Load memory elements off
regclr <= '1'; -- Accumulator register clear off
accum <= '1'; -- Accumulator changes on
mux <= "10"; -- Fourth memory element selected

-- This is the comparison phase of the match path. We set
-- the number of cycles the comparison will take with the
-- mtchnum value.
-- Up to 4 cycles can be taken.

when MPATH4 =>
  if (mtchnum = "00") then
    Next_State <= IDLE;
  else
    Next_State <= MPATH5;
  end if;
load <= '0'; -- Load memory elements off
regclr <= '1'; -- Accumulator register clear off
accum <= '0'; -- Accumulator changes off
mux <= "--"; -- Memory element selection DCed

when MPATH5 =>
  if (mtchnum = "01") then
    Next_State <= IDLE;
  else
    Next_State <= MPATH6;
  end if;
load <= '0'; -- Load memory elements off
regclr <= '1'; -- Accumulator register clear off
accum <= '0'; -- Accumulator changes off
mux <= "--"; -- Memory element selection DCed

when MPATH6 =>
  if (mtchnum = "10") then
    Next_State <= IDLE;
  else
    Next_State <= MPATH7;
  end if;
load <= '0'; -- Load memory elements off
regclr <= '1'; -- Accumulator register clear off
accum <= '0';  -- Accumulator changes off
mux <= "--";   -- Memory element selection DCed

when MPATH7 =>
  Next_State <= IDLE;
  load <= '0';    -- Load memory elements off
  regclr <= '1';  -- Accumulator register clear off
  accum <= '0';  -- Accumulator changes off
  mux <= "--";   -- Memory element selection DCed

  -- This is the emergency case. We should generate an error
  -- when we're in this state, but that will be left to a
  -- future version. Right now, just return to the idle state.

when OTHERS =>
  Next_State <= IDLE;

end case;

end if;

end process;

end synth;

E.2 The Cellular Processor

This is a VHDL description of the cellular processor. This is somewhat more complex
than the CP on the prototype chip because VHDL does not allow a graceful way to
reuse the same bus for inputs and outputs. As such, the inputs and outputs are
logically separated.

-----------------------------------------------
-- Cell - Associative Processor cellular processor
--
-- This is the cellular processor description in VHDL.

-- Library load time.
Library IEEE;
USE IEEE.Std_Logic_1164.all;

entity cellp is
port (  compare, reset, store, clk: in std_logic;
       min: out std_logic;
       phase: in std_logic_vector (1 downto 0);
       mbus: inout std_logic_vector (9 downto 0);
       dbus: in std_logic_vector (7 downto 0));
end cellp;

architecture synth of cellp is
begin

--
-- This section stores the data and does the additions and
-- subtractions.
--
process
signal SUM: std_logic_vector (9 downto 0);
signal VALID: std_logic;
begin

block
variable DATA0, DATA1, DATA2, DATA3:
   std_logic_vector (7 downto 0);
variable TEMP: std_logic_vector (7 downto 0);
begin
   wait until clk'event and clk = '1';

-- Resetting the accumulator and valid data bit have absolute
-- priority
if (reset = '0') then
   SUM <= "0000000000";
   VALID <= '0';
DATA0 := "00000000";
DATA1 := "00000000";
DATA2 := "00000000";
DATA3 := "00000000";
else
-- Store has the next level of priority.
if (store = '1') then
    VALID <= '1';
    case phase is
       when "00" => DATA0 := DBUS;
       when "01" => DATA1 := DBUS;
       when "11" => DATA2 := DBUS;
       when "10" => DATA3 := DBUS;
       when others => NULL;
    end case;
else
-- When all else fails, do the arithmetic based on the cycle
-- information in phase.
    case phase is
       when "00" =>
           TEMP := DATA0 - DBUS;
           SUM <= "00" & TEMP;
       when "01" =>
           TEMP := DATA1 - DBUS;
           SUM <= SUM + TEMP;
       when "11" =>
           TEMP := DATA2 - DBUS;
           SUM <= SUM + TEMP;
       when "10" =>
           TEMP := DATA3 - DBUS;
           SUM <= SUM + TEMP;
       when others =>
           NULL;
    end case;
end if; -- if (STORE)
end if; -- if (RESET)
end block;

--
-- This is the matching circuitry.
MATCH: block

variable C: std_logic_vector (9 downto 0);
begin

for i in 0 to 9 loop
  if (i = 9) then
    C(i) := VALID and (not (SUM(i) and (not mbus(i))));
    if ((VALID = '1') and (SUM(i) = '0')) then
      mbus(i) <= '0';
    else
      mbus(i) <= 'Z';
    end if;
  end if; -- msb

  if (i = 0) then
    min <= C(i+1) and (not (SUM(i) and (not mbus(i))));
    if ((C(i+1) = '1') and (SUM(i) = '0')) then
      mbus(i) <= '0';
    else
      mbus(i) <= 'Z';
    end if;
  end if; -- lsb

  if ((i < 9) and (i > 0)) then
    C(i) := C(i+1) and (not (SUM(i) and (not mbus(i))));
    if ((C(i+1) = '1') and (SUM(i) = '0')) then
      mbus(i) <= '0';
    else
      mbus(i) <= 'Z';
    end if;
  end if; -- other bits

end loop; -- ending for loop generate
end block; -- MATCH

end process;
end synth;
BIBLIOGRAPHY


